THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :
Université Toulouse 3 Paul Sabatier (UT3 Paul Sabatier)

Discipline ou spécialité :

Systèmes Informatiques et Systèmes Embarqués

Présentée et soutenue par :
Samir ALILI

le : jeudi 21 avril 2011

Titre :

Interaction décisionnelle Homme-Robot : planification de tâche pour un robot interactif en environnement humain

---

Ecole doctorale :
Systèmes (EDSYS)

Unité de recherche :
LAAS-CNRS

Directeur(s) de Thèse :

Rachid ALAMI

Rapporteurs :
François Charpillet
Dominique Duhaut

Autre(s) membre(s) du jury

Peter Ford Dominey
Palanque Philippe

Résumé

Cette thèse aborde la problématique de la décision partagée homme-robot dans une perspective de résolution interactive de problème à laquelle prennent part l'homme et le robot. Le robot et l'homme poursuivent des objectifs communs et doivent déterminer ensemble les moyens de les réaliser (les capacités et les compétences de chacun étant différentes). Les questions à traiter concernent ce partage des rôles, le partage d'autorité dans l'exécution d'une tche (prise d'initiative), les connaissances à exhiber afin que l'un et l'autre puissent jouer un rôle optimal dans la résolution du problème commun.

Nous avons développé un planificateur de tâche nommé HATP (Human Aware Task Planner). Ce planificateur est conçu sur la base de la planification hiérarchique qu'on a enrichie avec des règles sociales. Il permet de produire des plans qui sont socialement acceptables, c'est-à-dire des plans qui rendent lisibles les actions et les intentions du robot. Le planificateur a également la capacité de planifier pour le robot et l'humain tout en garantissant l'optimalité pour chacun d'eux.

Nous nous sommes également intéressés à une approche hybride, qui mixe la planification de tâche à la planification géométrique. Cette approche permet au robot d'avoir un contrôle sur la séquence d'actions qu'il produit mais également sur la façon de la réaliser. Ce qui permet de traiter le problème de l'interaction homme-robot de manière plus fine mais également sur plusieurs niveaux.

Mots-clés

Planification de tâche hiérarchique, Planification hybride, Planification pour la coopération, Interaction homme-robot, comportement social d'un robot.

Abstract

This thesis addresses the problem of the shared decision between human and robot in the perspective of interactive problem solving that involved human and robot. The robot and human share common goals and must work together to identify how to realize (the capacity and the competence of each one are different). Issues to be addressed concerning this division of roles, sharing of authority in the execution of a task (taking initiative), to exhibit the knowledge such that both can play an optimal role in the resolution of common problems.

We developed a task planner named HATP (Human Aware Task Planner). This planner is based on hierarchical task planning that is enriched with social rules. It can produce plans that are socially acceptable that means plans that make legible the actions and intentions of the robot. The planner also has the ability to plan for the robot and humans while ensuring optimality for each.

We are also interested in a hybrid approach that mixes between task planning and geometrical planning. This approach allows the robot to have control over the sequence of actions that it produces, but also on how to achieve it. Thereby treat the human-robot interaction problem more cleverly, but also on several levels.

Keywords

Hierarchical task planning, Hybrid planning, Planning for cooperation, Human robot interaction, Robot social behavior.

A mes parents.

A mon épouse et mon fils.

A mes frères et sœurs.

Remerciements

Les travaux présentés dans ce manuscrit ont été effectués au sein du groupe Robotique et InteractionS (RIS) du Laboratoire d'Analyse et Architecture des Systèmes (LAAS).

Je tiens tout d'abord à remercier Messieurs Raja Chatila, Directeur du LAAS-CNRS, pour m'avoir accueilli dans ce laboratoire et permis de réaliser ce travail.

Il m'est particulièrement agréable de remercier Monsieur Rachid Alami, Directeur de Recherche au LAAS/CNRS de Toulouse, pour avoir encadré ma thèse. Je lui témoigne toute ma gratitude pour la confiance qu'il m'a accordée et ses conseils scientifiques très pertinents. Malgré des responsabilités de direction du groupe RIS, il a toujours témoigné un vif intérêt à la réussite de mes travaux par des encouragements, tant professionnels que personnels.

Je remercie Mr François Charpillet et Mr Dominique Duhaut d'avoir accepté de rapporter ma thèse. Je remercie également Mr Peter Ford Dominey et Mr Philippe Palanque d'avoir accepté d'examiner cette thèse.

Je remercie également tous les membres du pôle Robotique permanents et doctorants qui m'ont témoigné soutien et sympathie. Un remerciement spécial pour Julien Guitton et Mathieu Warnier qui m'ont vraiment apporté toute leur aide pendant la rédaction de cette thèse.

Mes sincères remerciements s'adressent aussi à l'ensemble du personnel des services, Secrétariat, Edition-Documentation et Magasin du LAAS, je pense essentiellement à Natacha Ouwanssi, Camille Cazeneuve et Christian Berty.

Je remercie tous mes collègues et amis que j'ai côtoyés pendant toutes ces années et avec qui je garde de très bons souvenirs. Je commence par Nizar, Ali, et Amit, je leur dis : rédigez et arrêtez de penser à le faire. A Moktar, le plus grand des (em...) je lui dis bonne chance pour la suite. Xavier c'est bientôt ton tour, mon pote. Et tous ceux qui nous ont déjà quittés : Luis, Akin, Auréli, Vincent et Assia.

Ensuite, je poursuis avec ceux qui ont toujours été là pour moi : Mohamed Matmat (Hamada), Hamida Hallil, Ahmed Benyahia et enfin Hamza Semma.

Je ne saurais terminer sans remercier très profondément mes parents Mr Abd-El-Rahmen Alili et Mme Farida Alili qui m'ont continuellement soutenu et encouragé dans mes démarches. Mon épouse Fahima Cheikh et mon fils Sammy, sans qui je n'aurais jamais tenu jusque là. Ma sœur Chenda et mon frère Remy pour leurs aides continuelles et leurs témoignages d'affection et de tendresse durant toutes ces années. Naturellement, mes respectueux remerciements s'adressent également à mes chers grands parents ainsi qu'à l'ensemble de ma famille.

Enfin, que tous ceux qui se sentent délaissés me pardonnent, il s'agit très certainement d'une omission involontaire tant la liste est longue, remercier nominativement chacune des personnes s'avère une composition délicate à laquelle je me suis exposé.

Samir Alili

Table des matières

Liste des figures

Liste des tableaux

Liste des Algorithmes

1

Introduction

L'interaction homme robot est passée par plusieurs phases, tout au long de son histoire. Cela a commencé par l'apparition des automates mécaniques au XVI siècle, puis vient la révolution informatique avec l'apparition des ordinateurs dans les années 40 qui ont donné naissance au robot à proprement parler. On a commencé par les robots industriels. Puis les robots autonomes. Et maintenant les robots autonomes interactifs et cohabitant avec des humains.

Cette cohabitation humain-robot impose des contraintes sur les comportements possibles ; c'est-à-dire qu'un robot doit pouvoir agir tout en assurant la sécurité physique de ses partenaires humains, mais également se comporter de manière socialement acceptable par la société humaine.

Fong et .al dans [Fong 03] définit un robot social interactif comme un partenaire ou un assistant. Cela implique qu'il doit présenter un certain degré d'adaptabilité et de flexibilité pour conduire des interactions avec un large éventail d'humains [1]. Le robot doit donc prendre en compte l'humain à tous les niveaux. Du niveau fonctionnel jusqu'au niveau décisionnel. Au niveau fonctionnel le robot doit être capable de produire des mouvements sûrs, fluides et confortables pour son partenaire, sinon celui-ci risque de se lasser de lui et décline toute interaction avec ce dernier. Au niveau décisionnel, le robot doit être capable de produire des comportements qui

1. Socially interactive robots operate as partners, peers or assistants, which means that they need to exhibit a certain degree of adaptability and flexibility to drive the interaction with a wide range of humans

respectent les règles de vie des humains et qui correspondent à son statut.

La sociabilité du robot impose donc que celui-ci agisse de manière cohérente et intelligible comme ce qui a été défini dans [Nourbakhsh 05], c'est-à-dire qu'il doit agir de façon compréhensible pour ses partenaires humains. Pour cela le robot devra, dans la mesure du possible, se comporter comme un être humain. Lorsque deux êtres humains collaborent entre eux pour réaliser une tâche commune, ils se répartissent les efforts nécessaires à sa réalisation en fonction des aptitudes de chacun et de leurs états d'esprit courant. Si on souhaite donc qu'un robot puisse agir ainsi, il devra intégrer ces contraintes dans son système décisionnel. De plus, la façon dont chacun des partenaires réalise sa part de la tâche peut influencer le ressenti des autres sur le bon déroulement des opérations. Un partenaire agissant de manière non prédictible, c'est-à-dire effectuant des actions désordonnées, pourra susciter un sentiment d'incompréhension chez les autres partenaires et ainsi engendrer une remise en cause de la réalisation de la tâche. Afin de garantir une bonne sociabilité du robot, il convient donc d'intégrer ces règles de comportement dans le système de planification du robot.

Dans cette thèse, nous nous intéressons à la prise en compte de l'humain par le robot au niveau décisionnel et plus précisément au niveau de la planification de tâche. C'est au niveau de la planification que se prend la décision de comment réaliser la tâche, c'est-à-dire trouver la séquence d'actions qui permettra au robot d'atteindre son but. C'est le choix de l'ordre des actions qui déterminera le comportement du robot. Pour cela nous nous sommes intéressés à ce qu'on a appelé "*planification avec contraintes sociales*". Dans cette approche, le robot cohabite avec l'humain, c'est-à-dire qu'il partage le même espace et travaille en coopération avec lui. Le planificateur va produire des plans qui sont réalisables pour le robot comme pour l'humain, mais en faisant attention à respecter certaines contraintes sociales, ce qui va rendre l'interaction plus agréable pour l'humain.

Ce mémoire est articulé en cinq chapitres. Le premier chapitre va donner une vue générale sur l'état de l'art de l'interaction homme-robot où nous allons constater que pour une interaction homme-robot, l'humain doit être pris en compte à différents niveaux. Cependant, nous allons nous intéresser essentiellement à l'interaction humain-robot au niveau décisionnel où nous décrirons quelques approches existantes que nous illustrerons par des exemples.

Le second chapitre présente le modèle HTN de notre planificateur nommé HATP (Human Aware Task Planner), qui peut produire des plans dits socialement acceptables. Nous allons illustrer la représentation du monde utilisée et le langage de HATP. Nous allons également définir le modèle de tâche utilisé, et présenter notre définition d'un problème de planification et un plan solution dans HATP. Nous allons illustrer par la suite notre extension sur le domaine de planification. Nous allons voir des exemples de plan, et nous en déduirons des règles qu'on appellera règles de comportement ou règles sociales, qui vont nous permettre de donner au robot un comportement qui soit acceptable par les humains. Après avoir identifié les critères

qui influencent la qualité sociale d'un plan, nous allons vous présenter notre métrique pour l'évaluation sociale de ce plan, qui se base sur une analyse multicritère très utilisée en industrie pour l'aide à la décision. Enfin, nous allons présenter l'algorithmique du planificateur avec l'implémentation de l'heuristique A* intégrée au sein du planificateur. Nous allons conclure le chapitre par quelques critiques de ce modèle.

Le troisième chapitre présente une extension du modèle HATP. Pour cette approche nous allons présenter un modèle qui prend les humains en compte pendant le processus de panification, mais qui prend également en compte leurs croyances propres sur les états du monde. Nous allons commencer par une vue globale sur l'état de l'art et les approches qui traitent de cette problématique. Ensuite, nous présenterons le modèle du planificateur. Ceci en détaillant la structure de base des croyances qui permettent de prendre en charge les différentes croyances des différents agents. Par la suite nous allons présenter les extensions sur le modèle de tâche du planificateur, et nous allons voir notamment l'introduction d'un nouveau type d'action supplémentaire qui est l'action de communication. Nous poursuivrons par la description des modifications apportées sur l'algorithme de planification et finirons par des exemples produits par le planificateur afin d'améliorer l'interaction homme-robot.

Le quatrième chapitre présentera une autre extension du modèle HATP. Pour cette approche, nous allons présenter un modèle qui prend en compte l'humain au niveau symbolique et au niveau géométrique. Nous commencerons par donner une vue générale des travaux existants qui traitent de cette problématique. Ensuite, nous allons donner notre représentation du monde qui prend en considération les deux types de représentation géométrique et symbolique. Nous poursuivrons avec la description de notre modèle de tâche où nous allons donner le détail du modèle des actions, et nous expliquerons comment sont pris en charge les faits géométriques à ce niveau, tout en gardant une représentation générique des actions. Ensuite, nous ferons de même pour les tâches de haut niveau où nous verrons notre processus de propagation de contrainte et son influence sur la planification géométrique. Nous allons décrire par la suite l'algorithme qui permet de construire un plan avec deux composantes. Une composante discrète pour la partie symbolique et une composante continue pour la partie géométrique. Nous observerons également les différents niveaux de backtrack et leur gestion. Nous finirons par les exemples qui illustrent l'intérêt de la fusion de ces deux approches.

Le cinquième chapitre présente les résultats obtenus avec le planificateur HATP. Nous allons voir l'influence des règles sociales sur la qualité des plans produits et sur le temps de calcul du planificateur. Nous présenterons également les résultats qui démontrent l'efficacité de l'heuristique de guidage. Nous illustrerons les performances du planificateur sur un robot réel, intégré dans une architecture dédiée à l'interaction homme-robot, et également son intégration dans une autre architecture dédiée au multi-agent.

Liste des publications liées à cette thèse

– *S. Alili, A. K. Pandey, E. A. Sisbot and R. Alami*, "**Interleaving Symbolic and Geometric Reasoning for a Robotic Assistant**", ICAPS Workshop on Combining Action and Motion Planning (CAMP 2010).

– *M. Ali, S. Alili, M. Warnier and R. Alami*, "**An Architecture supporting Proactive Robot Companion Behavior**", AISB 2009, Edinburgh, April 2009, Scotland.

– *S. Alili, M. Warnier, M. Ali and R. Alami*, "**Planning and Plan-execution for Human-Robot Cooperative Task achievement**", 19th International Conference on Automated Planning and Scheduling, September 19-23, 2009, Thessaloniki, Greece.

– *S. Alili , R. Alami , V. Montreuil* "**A Task Planner for an Autonomous Social robot**", DARS 2008 - Distributed Autonomous Robotic Systems 2008, Tsukuba (Japon), 17-18 Novembre 2008.

– *A. Clodic, H. Cao, S. Alili, V. Montreuil, R. Alami, and R. Chatila*, "**Shary : a supervision system adapted to Human-Robot Interaction**", International Symposium on Experimental Robotics, ISER 2008, Athens , July 2008.

2

Interaction Homme-Robot

2.1 État de l'art

On peut voir de plus en plus de robots autonomes ou semi-autonomes qui aident les humains dans leur vie de tous les jours. Par exemple, le robot guide de musée [Shiomi 06] au musée des sciences d'Osaka ou [Clodic 06] à la cité de l'espace à Toulouse. Le robot Valérie [Gockley 06] à l'Université de Carnegie Mellon qui est un robot réceptionniste. Les robots transporteurs d'objets utilisés dans les industries [Huttenrauch 02]. Mais ces applications restent restreintes à des interactions de courte durée et ces systèmes peuvent être vus comme des automates de haut niveau qui ont des primitives d'interaction très développées.

Ces dernières années, l'interaction homme-robot et la collaboration homme-robot sont devenus des sujets de recherche très actifs dans différentes disciplines et à différents niveaux. Par exemple, au niveau sociologique plusieurs travaux essaient d'évaluer les réactions d'humains interagissant avec un robot. Ces études se focalisent sur différents aspects physiques et comportementaux du robot. Par exemple, les travaux de [Hiolle 09] [Saint Aimé 10] essayent d'évaluer les réactions d'un humain face à différents profils interactifs d'un robot. Il y a aussi les travaux de [Cortellessa 08] qui évaluent l'influence du cadre culturel des humains dans leur

interaction avec des robots, mais également les réactions des humains face à un comportement proactif et protecteur d'un robot. On a aussi étudié l'influence de l'aspect extérieur d'un robot sur l'interaction comme dans les travaux [van Vugt 07].

D'autres recherches visent la prise en compte des humains au niveau fonctionnel, c'est-à-dire prendre en compte des humains au moment de l'exécution des actions du robot. Ce type d'approche tente de donner au robot un comportement dit socialement acceptable, et cela en ne considérant plus les humains comme de simples obstacles à éviter. On peut citer les travaux de [Pacchierotti 05] qui prend en compte comme contrainte le comportement spatial d'un humain pour calculer les déplacements du robot.

Il y a aussi les travaux de [Sisbot 07a], [Pandey 09] qui a développé un planificateur de trajectoire dédié à la prise en compte des humains présents dans le voisinage du robot. Ce planificateur génère et adapte en ligne des trajectoires qui ne surprennent pas l'humain. Pour ce faire, ces trajectoires respectent certaines contraintes dont celle de rester toujours dans le champ de visibilité de l'humain. Les travaux de [Luis F. Marin-Urias 08] complètent les travaux précédents. Ils ont permis de développer un générateur de configuration pour un robot qui travaille en coopération avec des humains. Ils permettent de calculer des configurations dites sûres, et socialement acceptables pour un robot qui effectue une tâche jointe et proche de l'humain en terme de distance. La tâche à réaliser est prise en compte comme contrainte. En effet, la configuration pour un robot qui donne un objet à un homme sera différente de la configuration pour poser un objet. Pourquoi ? Parce que la présence de l'humain engendre des contraintes supplémentaires telles que la distance de sécurité, la distance d'interaction, etc. D'autre travaux [Trafton 05] et [Marin-Urias 08] se sont intéressés à la faculté de l'être humain de se mettre à la place de l'autre (Perspective Taking). Cette faculté permet au robot de mieux gérer les requêtes de l'humain en se mettant à sa place pour résoudre d'éventuelles ambiguïtés.

On peut citer aussi certaines recherches portant sur l'attitude physique d'un robot lorsqu'il est engagé dans une conversation : Ceci joue un rôle important dans l'aspect social de son comportement. Lorsqu'un robot dialogue avec un partenaire humain il se doit de respecter des règles d'engagement et de désengagement durant le déroulement de la conversation. Les travaux de [Sidner 03b] et [Sidner 03a] mettent en évidence l'importance du suivi des mouvements de la tête et de l'orientation du regard lors d'une discussion. On voit là encore que la réalisation d'une action élémentaire du robot se doit d'être exécutée en tenant compte de la présence des humains à proximité du robot.

D'autres approches choisissent d'équiper le robot d'un modèle cognitif ce rapprochant par certains aspects de celui de l'humain. Cette approche intéresse plus particulièrement toute la communauté ACT-R [1] (architecture cognitive pour la reproduction et la compréhension du modèle cognitif humain). Les travaux de [Kennedy 07] s'inscrivent dans ce cadre : celui-ci essaie en effet de faire le lien entre la représentation spatiale numérique d'un robot et la représentation

1. http ://act-r.psy.cmu.edu/

spatiale symbolique des humains en construisant une carte cognitive. Cette dernière contient les positions relatives et les orientations entre les éléments importants de l'environnement. Le but ici est de faciliter la communication entre humain et robot, et de permettre au robot d'identifier plus aisément les objets ou les personnes dont l'humain parle.

D'autres travaux se sont intéressés à une reproduction partielle du modèle de représentation de connaissance chez l'humain en se basant sur une représentation hiérarchisée de l'environnement [Galindo 04]. Dans cette approche, l'environnement dans lequel évolue le robot est modélisé à l'aide d'un graphe multi-hiérarchique annoté. L'avantage de cette approche réside en ce sens qu'elle permet au robot de manipuler des concepts de haut niveau. Cela a pour effet de faciliter la communication avec les humains, mais cela constitue également une source d'information supplémentaire pour le robot qui l'utilise pour le contrôle de l'exécution comme l'explique [Bouguerra 07].

Nous nous intéressons dans cette thèse à la prise en compte de l'humain au niveau délibératif. Là aussi il y'a beaucoup d'approches différentes qu'on peut classer en quatre familles :

- Planification d'agenda.
- Ordonnancement et allocation de ressource.
- Synthèse de plan.
- Planification probabiliste, contingente et autre.

2.1.1 La planification d'agenda

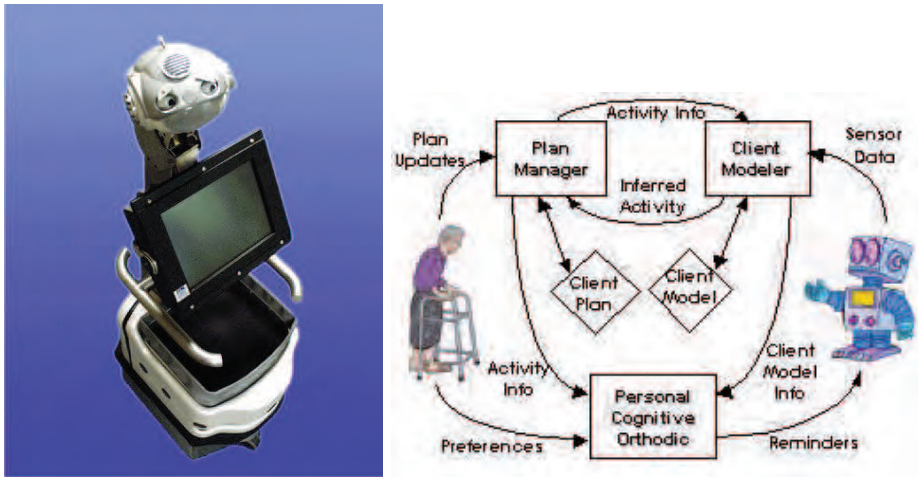
Dans ce type de planification, le modèle de tâche est simplifié et réduit à une triplette $< N, C_t, C_p >$ où :

- N définit le nom de la tâche.
- C_t représente un ensemble de contraintes temporelles propres à la tâche N. Ces contraintes temporelles précisent les intervalles pendant lesquels les temps de début et de fin associés à une tâche peuvent avoir lieu.
- C_p représente un ensemble de contraintes de précédence entre la tâche N et les autres tâches présentent dans le domaine. Ces contraintes de précédence déterminent l'écart temporel entre la fin d'une tâche et le début des autres tâches.

On peut noter que le modèle ne fait pas de raisonnement sur la faisabilité ou la finalité des tâches, mais uniquement un raisonnement temporel, car il n'inclut pas les notions de précondition pour la réalisation ou l'effet de la tâche réalisée. Le rôle du planificateur dans ce modèle est de trouver un ordonnancement temporel pour toutes les tâches présentes dans le domaine en respectant les contraintes temporelles et de précédence. Ainsi, on peut dire que ce type de planification est dépendant, car le raisonnement sur la faisabilité et la finalité des tâches reste à la charge de la personne qui spécifie les contraintes.

Une des applications qui utilise ce type de planification est le robot Pearl ou Autominder (figure 2.1.a) [Pollack 03] de l'université du Michigan. C'est un robot interactif destiné à porter assistance aux personnes âgées et aux personnes qui souffrent d'une défaillance de la mémoire, en gérant leur emploi du temps et en leur rappelant les tâches importantes qu'elles doivent effectuer (prise de médicament, heure du déjeuner, etc.).

La figure 2.1.b illustre l'architecture de l'Autominder. Elle est composée de trois éléments :



(a) Pearl robot assistant pour personne âgée (b) Schéma de fonctionnement du robot Pearl

FIGURE 2.1 – Le robot Pearl et son schéma de fonctionnement

– **Plan manager** : C'est un agenda évolutif intelligent [Pollack 99]. Le plan manager est une librairie de plans de l'humain dans laquelle il peut ajouter, supprimer, ou modifier les contraintes sur les tâches.

– **Client Modeler** : c'est un module responsable de deux fonctions : (1) supervision des actions de l'humain en utilisant les données des capteurs du robot et également en utilisant un réseau bayésien dynamique. (2) apprentissage du modèle de l'humain (préférence, habitude, etc.).

– **PCO (Personal Cognitive Orthotic)** : C'est un module responsable de la gestion de l'agenda de l'humain. C'est lui qui de décide quoi et quand rappeler la tâche à faire. En se basant sur le plan produit par le plan manager et la préférence de l'humain. C'est un module basé sur la planification par réécriture (PbR) [Ambite 98].

Le principe de fonctionnement de l'Autominder est illustré sur la figure 2.2. Le Plan manager reçoit en entrée les tâches quotidiennes de son partenaire humain (certaines tâches sont fournies par l'administrateur, par exemple le médecin, et d'autres par le patient lui-même), ainsi qu'un ordre de priorité sur les tâches et un ensemble de contraintes sur les temps et l'ordre de réalisation de ces tâches. Le planificateur utilise le formalisme DTP (Disjunctive Temporal Problem) détaillé dans [Pollack 02], qui détermine un ordonnancement consistant pour toutes les tâches de l'humain en respectant toutes les contraintes spécifiées. Le plan produit par le

planificateur est un 4-uplet $< S, O, L, B >$ où S représente les différentes étapes du plan (les actions que l'humain doit faire) ; O, L, B représentent respectivement les contraintes d'ordre, les liens causaux et les contraintes d'instanciation. Le système est capable d'adapter le plan en ajoutant ou en supprimant des tâches en fonction des requêtes qu'il reçoit de la part de l'humain partenaire ou bien de la part de l'administrateur. Le robot est alors en charge de la supervision de son partenaire humain. Dans le cas où l'humain oublie une tâche, le robot intervient pour le lui rappeler.

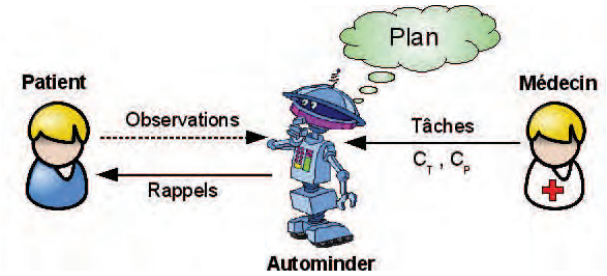


FIGURE 2.2 – Principe de fonctionnement du robot Autominder

2.1.2 L'ordonnancement et l'allocation de ressources

Ce type de planification est inspiré de la planification d'agenda, le modèle a été étendu en rajoutant la notion de ressource. Le modèle de tâche est décrit par 4-uplet $< N, C_t, C_p, C_r >$ où :

– N représente le nom de la tâche. Le nom d'une tâche est unique, car c'est l'identifiant de la tâche.
– C_t représente un ensemble de contraintes temporelles propres à la tâche N. Ces contraintes temporelles indiquent les intervalles pendant lesquels la tâche peut être réalisée.
– C_p représente un ensemble de contraintes de précédence entre la tâche N et les autres tâches présentes dans le domaine. Ces contraintes de précédence déterminent l'écart temporel entre la fin d'une tâche et le début des autres tâches.
– C_r représente un ensemble de contraintes de ressource. Ces contraintes indiquent l'ensemble des ressources que la tâche va mobiliser avant, pendant et après la réalisation de la tâche.

Comme dans le modèle précédant le planificateur doit trouver un plan solution qui respecte l'ensemble des contraintes.

Une des applications qui utilise ce type de planification est le système d'exploitation HRI/OS [Fong 06]. Ce système a été développé en collaboration avec la NASA pour faciliter la communication entre humains et robots dans le cadre de la réalisation collaborative de tâches. Ce système fournit un cadre générique permettant aux différents agents en présence de communiquer entre eux afin qu'ils puissent s'informer de leurs aptitudes propres, de leurs tâches courantes et de l'état d'avancement de leurs actions.

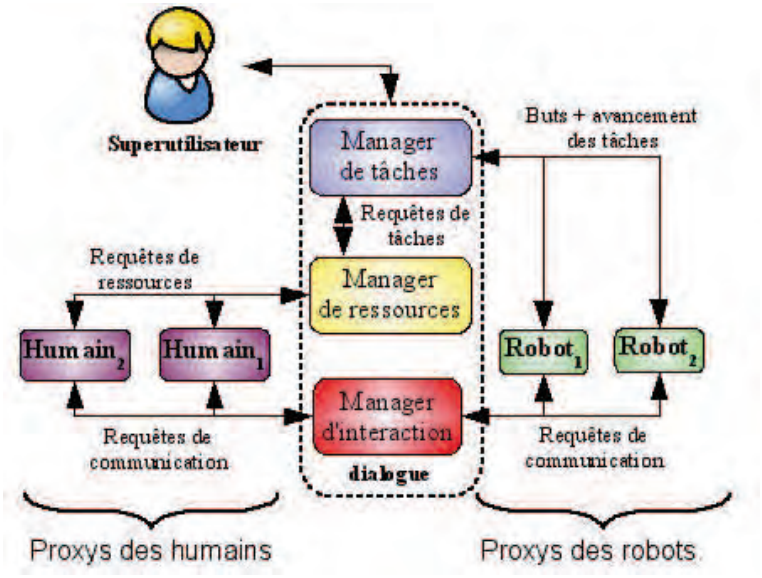


FIGURE 2.3 – Schéma de fonctionnement du système d'exploitation HRIOS

Le schéma de fonctionnement du système d'exploitation est représenté sur la figure 2.3. Chaque agent (humain, robot) est schématisé par un proxy qui gère toutes les requêtes. L'architecture a trois modules principaux qui sont :

– **Le manager de ressource :** Pour chaque tâche à réaliser le manager des ressource doit établir une liste ordonnée des agents ayant les capacités nécessaires en fonction de la situation courante (la position, les ressource, le temps, etc.).

– **Le manager de tâche :** Il utilise la liste produite par le manager de ressource pour envoyer des requêtes de réalisation dans l'ordre de la liste.

– **Le manager d'interaction :** il gère les communications entre agents.

La planification se fait au niveau du manager de ressource qui va déterminer l'ordre de réalisation et l'allocation des tâches, en prenant en compte les ressources, les contraintes et les compétences de chaque agent. Comme précédemment, le système ne gère pas lui-même les buts à réaliser, ils sont laissés à la charge d'un superviseur humain qui gère le bon fonctionnement du système.

2.1.3 La synthèse de plans

Dans ce mode de planification, on enrichit le modèle de tâche en y intégrant les notions de préconditions et d'effets pour/de la réalisation de la tâche. Le modèle de tâche est décrit par $< N, P, E >$ où :

– N représente le nom de la tâche. Il est utilisé comme son identifiant. Le symbole N est unique et il est associé à une seule tâche.

– P représente un ensemble de préconditions qui déterminent les conditions de réalisation

de la tâche.
– E représente l'ensemble des effets de la réalisation de la tâche sur le monde.

Ce type de planification peut être décliné en deux variantes : la planification réactive et la planification délibérative. La planification réactive repose sur l'établissement d'un plan général qui va être instancié et adapté au fur et à mesure de l'exécution, alors que la planification délibérative vise à établir une succession d'actions à réaliser puis à contrôler le bon déroulement de ces actions lors de l'exécution.

2.1.3.1 La planification réactive

Ce type de planification est destiné à créer des applications multi-agents, en se basant sur les théories de la coopération entre agents hétérogènes telle que la théorie de l'intention jointe ou les SharedPlans. Dans cette approche, le planificateur démarre le processus de planification à partir d'un plan complet, qui est décrit de manière générique. Ce plan va être instancié au fur et à mesure de l'exécution des différentes tâches du plan.

Théorie de l'intention jointe

La théorie de l'intention jointe [Cohen 90], [Cohen 91], [Levesque 90], a été développée afin de définir un cadre formel permettant d'établir des buts communs entre plusieurs agents. Cette théorie suggère qu'une action jointe est plus qu'un ensemble d'actions individuelles et qu'il est nécessaire de prendre en compte les croyances communes et individuelles de chaque agent pour former une équipe. Par conséquent, elle définit des buts persistants et la nécessité d'établir des engagements qu'il faut maintenir tant que tous les intervenants de la tâche ne sont pas avertis de sa terminaison ou de son échec. Elle définit la notion de but commun persistant $JPG(\Theta, p, q)$ où Θ représente l'équipe associée à la réalisation du but commun p sous les conditions de pertinence q. Un but commun persistant $JPG(\Theta, p, q)$ sera valide tant que les conditions suivantes seront satisfaites :

– Chaque membre de Θ a la croyance que le but p est achevé.
– Chaque membre de Θ a p comme but.
– Chaque membre μ de Θ a la croyance mutuelle que chacun des membres de Θ a p comme but individuel jusqu'à ce que p soit mutuellement reconnu comme achevé.

Un but p est considéré comme achevé s'il est terminé avec succès, avec échec, ou abandonné, ce dernier cas étant déterminé par l'éventualité selon laquelle les conditions de pertinence q ne sont plus valides. Un but individuel $WAG(\mu, p, \Theta, q)$ pour un membre $\mu \in \Theta$ est valide si une des conditions suivantes est satisfaite :
– μ a la croyance individuelle que p n'est pas encore achevé et a la croyance individuelle que p doit être achevé.

– μ a la croyance individuelle que p est achevé (par un succès, un échec ou un abandon) et s'efforce de rendre cette croyance commune à tous les membres de Θ, c'est-à-dire transformer cette croyance individuelle en une croyance mutuelle.

Une des applications, dérivée de la planification réactive et le la théorie de l'intention jointe, est l'infrastructure *STEAM* (Shell TEAMwork) décrite dans [Tambe 97], [Scerri 03]. Dans cette approche, chaque agent (humain, robot) a un ensemble de croyances individuelles et mutuelles conformément à la théorie de l'intention jointe. Chaque agent va raisonner sur un plan réactif hiérarchique, appelé TOP (Team Oriented Plan) [Scerri 04].

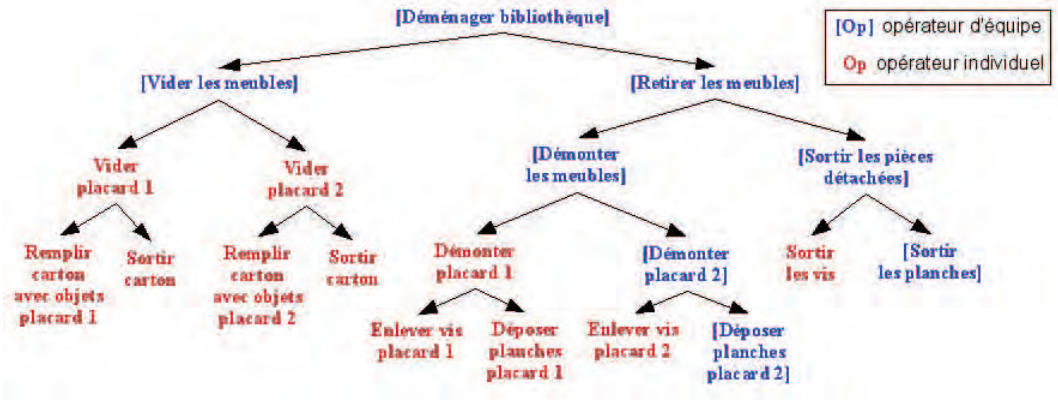


FIGURE 2.4 – Exemple de Team Oriented Plan.

La figure 2.4 illustre un exemple d'un plan générique TOP. Sur un plan TOP on distingue deux types d'opérateurs : (1) les opérateurs individuels qui ne concernent qu'un agent, (2) les opérateurs d'équipe qui concernent tous les agents impliqués ou un sous-groupe. Chaque opérateur est composé de trois parties :

– **Les règles de précondition :** Elles définissent les conditions de réalisation des tâches.

– **Les règles d'application :** Elles décrivent les sous-buts à réaliser pour satisfaire l'opérateur courant (description de la structure hiérarchique).

– **Les règles de terminaison :** Elles définissent les conditions menant à la fin de l'opérateur (échec, succès ou abandon).

Considérons une équipe d'agents qui doit réaliser l'opérateur d'équipe "*déménager la bibliothèque*" : Les agents commencent par établir un but commun persistant (JPG) sur cet opérateur. Pour cela, les agents s'échangent des informations qui servent à construire les croyances mutuelles et individuelles. Une fois le but commun persistant établi, les agents consultent le TOP et appliquent les recettes fournies par celui-ci. La tâche "*déménager la bibliothèque*" sera décomposée en deux opérateurs d'équipe "*vider les meubles*" et "*retirer les meubles*". L'opérateur "*retirer les meubles*" sera traité une fois que l'opérateur "*vider les meubles*" sera achevé. Lorsqu'un agent est affecté à un opérateur individuel, il utilise ses propres croyances et la recette fournie par le TOP pour construire un plan et, une fois le plan établi, l'agent commence son exécution jusqu'à ce qu'il obtienne la croyance individuelle que l'opérateur

associé est terminé. Lorsqu'un opérateur d'équipe est achevé, les agents de l'équipe ayant participé à sa réalisation s'échangent des informations afin d'établir les croyances mutuelles nécessaires à la fin du JPG associé à l'opérateur d'équipe.

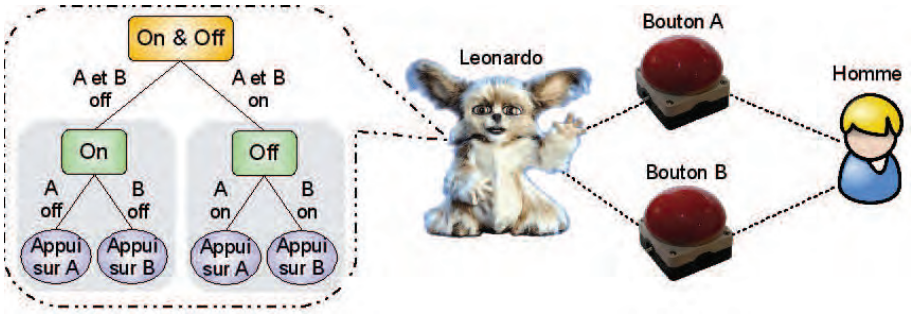


FIGURE 2.5 – Exécution de tâches avec le robot Leonardo

Un autre exemple d'utilisation de la planification réactive est le robot Leonardo [Breazeal 04]. Développé au sein du MIT, ce robot est doté de capacité interactive suivante(s) : expression faciale, communication gestuelle, dialogue, etc. Il est capable d'apprentissage collaboratif avec ses partenaires humains. Lors de son apprentissage, le robot construit un plan réactif hiérarchique figure 2.5. Leonardo utilise ses capacités interactives pour établir un but commun persistant (JPG), mais aussi pour construire ses croyances mutuelles et individuelles lui permettant de construire un plan consistant.

2.1.3.2 La planification délibérative

A la différence de la planification réactive, la planification délibérative consiste à générer un plan complet avant de se lancer dans son exécution. Ce type de planification présente l'avantage de produire des plans optimisés puisque la totalité du plan est produite avant l'exécution. Les applications existantes utilisant la planification délibérative pour la collaboration entre agents hétérogènes peuvent se classer en deux catégories : (1) les systèmes permettant à l'homme d'intervenir dans le processus de planification et (2) les systèmes utilisant la planification pour proposer des actions adéquates à l'homme.

Planification à initiative mixte

Les entrées d'un planificateur classique sont le domaine qui décrit l'ensemble des actions possibles et le problème qui décrit le but à satisfaire comme l'illustre la figure 2.6. Avec ces deux descriptions, le planificateur va trouver de façon autonome une solution qui satisfait le but. Dans l'approche de la planification mixte, le système prend en considération les recommandations et les désirs de l'utilisateur (humain). En effet, ce type de planificateur est basé généralement sur des structures hiérarchisées, qui vont être raffinées suivant les recommandations des utilisateurs.

Le planificateur *PASSAT* ([Myers 02]) est un exemple de planificateur à initiative mixte.

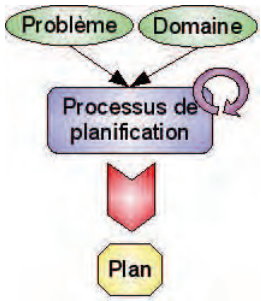


FIGURE 2.6 – Planificateur ¡¡ standard ¿¿

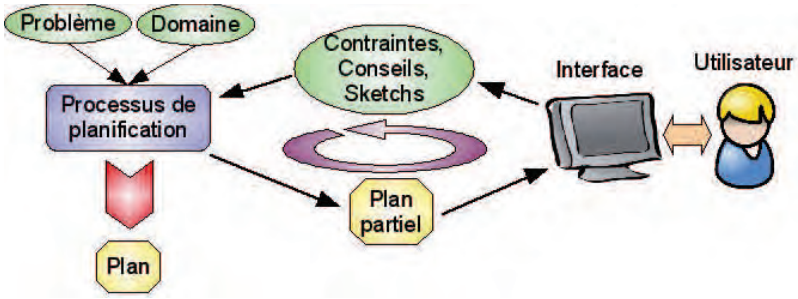


FIGURE 2.7 – Planificateur à initiative mixte

Ce planificateur est basé sur le formalisme Hierarchical Task Network (HTN). Ce formalisme utilise des actions de bas niveau (les opérateurs) et un ensemble de tâches de haut niveau, appelées méthodes. Les méthodes peuvent être décomposées en un ensemble partiellement ordonné d'opérateurs et/ou de méthodes formant ainsi une structure hiérarchique de tâches. Ainsi, lors du processus de planification, un planificateur basé sur le formalisme HTN va développer une structure arborescente jusqu'à ce que les feuilles de cette structure soient toutes des opérateurs. Le planificateur *PASSAT* permet à l'utilisateur de guider l'évolution de l'arbre de raffinement pendant le processus de planification, c'est-à-dire de l'aider à choisir parmi les décompositions possibles pour les méthodes qui restent encore à développer. Pour cela, *PASSAT* a été doté d'un système robuste de traitement de *sketch* [Myers 03]. Comme il a été défini dans [Myers 97], un *sketch* est un ensemble de tâches définissant une structure hiérarchique partiellement ou totalement instanciée. Le processus de planification cherche alors à produire une structure arborescente incluant totalement ou partiellement le *sketch*. L'objectif final est alors de trouver la structure arborescente qui soit la plus compatible possible avec le *sketch* défini par l'utilisateur.

Un autre exemple est le système *TRIPS-CAMPS* [Burstein 03] qui est une application dans le domaine du transport aérien pour l'US Air Force ou [Bresina 05] pour la gestion des missions Mars rover.

La planification délibérative a également été utilisée pour permettre à un agent logiciel d'aider un partenaire humain dans ses prises de décision. Cette approche est basée sur la théorie des plans partagés.

Théorie des plans partagés

La théorie des plans partagés formalise les intentions nécessaires à la constitution d'un plan permettant la réalisation des tâches désirées. Cela permet donc de formaliser la composante intentionnelle du dialogue, lorsque celui-ci vise à déterminer la répartition des tâches entre agents collaboratifs. Cette théorie présente l'avantage majeur de faire apparaître explicitement les notions de plans en cours d'élaboration (plans partiels) et de plans intégralement déterminés (plans complets).

La théorie des plans partagés ([Grosz 96], [Grosz 98]) est basée sur la notion d'intention de réaliser une tâche. Cette théorie a été développée pour permettre d'établir formellement la composante intentionnelle du discours lorsque celui-ci porte sur la réalisation de tâches par une équipe d'agents. Elle présente l'avantage majeur de faire apparaître explicitement les notions de plans en cours d'élaboration (plans partiels) et de plans intégralement déterminés (plans complets).

La théorie des plans partagés utilise la notion de recette. Une recette est un ensemble de sous-tâches à effectuer pour réaliser une tâche de plus haut niveau. De ce fait, les recettes permettent de construire une hiérarchie de tâches. La réalisation d'une tâche T par une équipe Θ en utilisant la recette R_T consiste alors à répartir les sous-tâches composant R_T parmi les membres de Θ. Afin de représenter les états mentaux des agents au fur et à mesure du déroulement du dialogue, on utilise quatre méta-prédicats : FIP, PIP, FSP et PSP. Ces méta-prédicats sont respectivement associés à des plans individuels complets, des plans individuels partiels, des plans partagés complets et des plans partagés partiels.

Le méta-prédicat FIP (*Full Individual Plan* en anglais) représente l'état mental d'un agent G lorsqu'il a déterminé un plan P_α grâce à une recette R_α complète permettant de réaliser la tâche α, c'est-à-dire qu'il a un plan individuel complet. FIP(P_α,G,α,R_α) est valide si :

1. G connaît une recette R_α pour réaliser la tâche α et chaque sous-tâche β_i de R_α,
2. une des conditions suivantes est vraie :
   - ➢ β_i est une action élémentaire et G a l'intention de réaliser β_i lui-même,
   - ➢ β_i est une tâche de haut-niveau et FIP(P_{β_i},G,β_i,R_{β_i}) est valide,
   - ➢ G a l'intention de sous-traiter la réalisation de β_i à un autre agent.

Le méta-prédicat PIP (*Partial Individual Plan* en anglais) représente l'état mental d'un agent G utilisant un plan P_α pour réaliser la tâche α, en ayant trouvé uniquement une solution partielle (c'est-à-dire que P_α est incomplet). PIP(P_α,G,α) est valide si :

1. G a la croyance qu'il existe une façon de réaliser α mais ne possède actuellement qu'une recette partielle (il a alors l'intention de compléter cette recette).
2. Pour chaque sous-tâche β_i de la recette partielle, une des conditions suivantes est vraie :
   - ➢ G a l'intention de réaliser β_i lui-même mais n'a qu'un plan partiel pour le faire,
   - ➢ G a l'intention de sous-traiter β_i à un autre agent mais n'a pas de plan complet pour réaliser l'acte de sous-traitance,
   - ➢ G n'a pas encore l'intention franche de réaliser β_i (il n'a qu'une intention potentielle).

Le méta-prédicat FSP (*Full Shared Plan* en anglais) représente l'état mental d'un groupe d'agents GR lorsque ce groupe a déterminé un plan P_α, grâce à une recette R_α complète permettant de réaliser la tâche α. FSP(P_α,GR,α,R_α) est valide si :

1. le groupe GR a la croyance mutuelle que tous les membres de GR sont impliqués dans la réussite de la tâche α,

2. le groupe GR a la croyance mutuelle des sous-tâches β_i à réaliser pour réaliser la tâche α.
3. pour chaque sous-tâche β_i une des conditions suivantes est vraie :
   - ➢ La tâche β_i est une tâche nécessitant un seul agent et un membre de GR va la réaliser,
   - ➢ La tâche β_i est une tâche nécessitant plusieurs agents et un sous-groupe de GR va la réaliser,
   - ➢ La tâche β_i est une tâche nécessitant un seul agent et le groupe GR a l'intention de sous-traiter la réalisation de β_i à un autre agent extérieur à GR,
   - ➢ La tâche β_i est une tâche nécessitant plusieurs agents et le groupe GR a l'intention de sous-traiter la réalisation de β_i à un groupe d'agents extérieurs à GR.

Le méta-prédicat PSP (*Partial Shared Plan* en anglais) représente l'état mental d'un groupe d'agents GR utilisant un plan P_α pour réaliser la tâche α, en n'ayant actuellement trouvé qu'une solution partielle (c'est-à-dire que P_α est incomplet). PSP(P_α,GR,α) est valide si :

1. le groupe GR a la croyance mutuelle que tous les membres de GR sont impliqués dans la réussite de la tâche α,
2. le groupe GR a la croyance mutuelle qu'il existe une recette permettant de réaliser la tâche α, mais la recette actuelle n'est que partielle. Les membres de GR ont un FSP pour réaliser la recette partielle.
3. Pour chaque sous-tâche β_i dans la recette partielle, une des conditions suivantes est vraie :
   - ➢ β_i ne nécessite qu'un agent, un agent membre de GR, a l'intention de réaliser β_i mais n'a qu'un plan partiel pour la réaliser,
   - ➢ β_i nécessite plusieurs agents, un sous-groupe de GR a l'intention de réaliser β_i mais n'a qu'un plan partiel pour la réaliser,
   - ➢ β_i ne nécessite qu'un agent, le groupe GR a décidé de sous-traiter la réalisation de β_i à un autre agent, mais le groupe GR n'a qu'un plan partiel pour réaliser l'acte de sous-traitance,
   - ➢ β_i nécessite plusieurs agents, le groupe GR a décidé de sous-traiter la réalisation de β_i à un autre groupe, mais le groupe GR n'a qu'un plan partiel pour réaliser l'acte de sous-traitance,
   - ➢ le groupe GR n'a pas encore délibéré à propos de β_i et aucune décision n'a encore été prise sur qui réalisera β_i.

Une des applications qui utilise la théorie des plans partagés est le système CoLLAGEN pour agent collaboratif ([Rich 97], [Rich 98]). Le fonctionnement de ce système peut être illustré par la figure 2.8. L'objectif est de permettre à un agent logiciel d'apporter son aide à un partenaire humain qui est en train d'utiliser une application sur un ordinateur. Afin de garantir un fonctionnement efficace et une interaction de bonne qualité, une entité appelée "*manager d'interaction*" est en charge d'analyser le comportement du partenaire humain afin de lui proposer une aide adaptée. Le *manager d'interaction* scrute les actions que le partenaire humain effectue dans l'application, et analyse les différentes communications entre ce dernier et

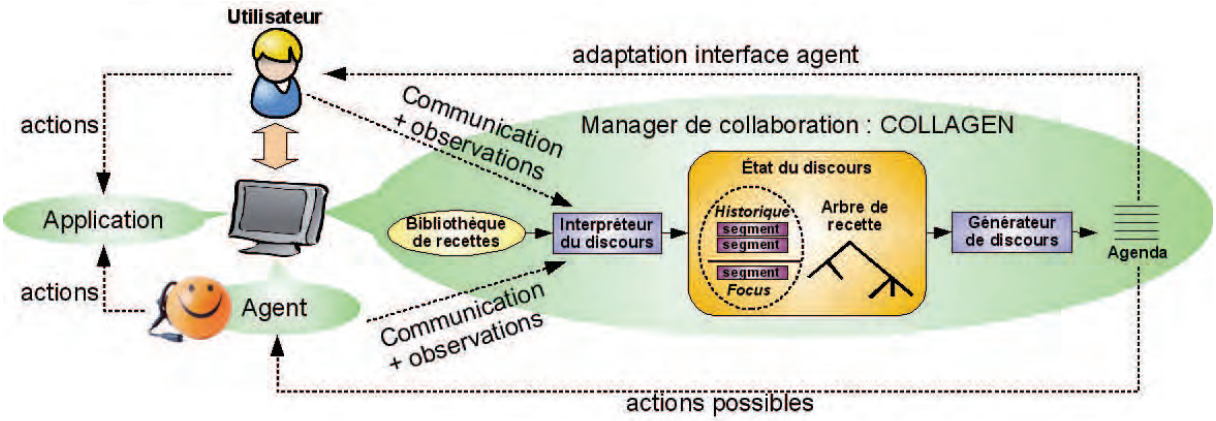


FIGURE 2.8 – Structure de l'agent COLLAGEN

l'agent logiciel grâce à un module d'interprétation du discours. Un module est alors en charge de maintenir l'état de la composante intentionnelle du partenaire humain en conservant l'historique des communications (en se focalisant sur les plus récentes) et en construisant un arbre de recettes représentant le *plan partagé partiel* (PSP) courant. Cette structure permet d'établir les actions qui restent à effectuer et permet donc d'établir une liste d'actions affectées de priorité, qui sont stockées dans l'agenda. Ce dernier est alors utilisé par l'agent soit pour agir directement sur le logiciel, soit pour adapter la communication entre l'agent et son partenaire humain. Ce principe est répété jusqu'à obtention par l'humain d'un *plan individuel complet* (FIP).

2.1.4 Planification probabiliste, contingente et autre

Dans cette section nous allons donner une brève idée des autres techniques utilisées pour l'interaction homme robot. Il y a ce qu'on appelle la planification probabiliste. Dans ce type d'approche, on s'intéresse aux aspects incertains et aux facultés limitées des robots à observer le monde qui les entoure. On retrouve les systèmes MDP (Processus Décisionnel de Markov) et POMDP (Processus Décisionnel de Markov Partiellement Observable) [BEYNIER .A 08], qui s'intéressent à des situations où les robots ou les agents ne savent pas si leurs actions vont aboutir, et pour le POMDP en ayant une observabilité partielle de certains états. Un exemple de ce type d'approche est fourni par les travaux de [Broz 08] qui modélise un sous-problème d'interaction homme-robot par un POMDP temporisé. Citons également [Beynier 06] qui fait de la gestion multi-agent avec un MDP décentralisé et aussi [Karami 10] qui modélise des situations d'interaction homme-robot avec un POMDP et qui évite les conflits entre humain et robot en essayant de calculer l'intention de l'homme.

Il y a aussi les approches dites planification conditionnelle. Dans ce type d'approche, on produit, comme pour la planification délibérative, un plan complet avec des branches supplémentaires qui sont développées ou seront développées pour certains faits dont on n'est pas sûr. On peut citer comme exemple le travail de [Bouguerra 04],

D'autres approches considèrent les cas d'une planification avec événement externe, comme

dans [Blythe 96] qui utilise un réseau bayésien pour évaluer la robustesse d'un plan produit face à des événements externes qui peuvent modifier les états du monde. Cette idée a été reprise par [Cirillo 10] qui l'a étendue vers un modèle POMDP pour la planification homme-robot. Dans cette approche, le robot a un agenda de tâche qu'il doit réaliser. Pour pouvoir le faire, il prend comme entrée les tâches des humains qu'il assiste.

Dans cette thèse, nous ne nous intéressons pas aux aspects gestion de l'incertain et de l'indéterminisme. Nous sommes plus dans l'optique d'une planification déterministe pour produire des plans collaboratifs entre un homme et un robot, tout en imposant au robot des contraintes de comportement. Nous nous situons donc plus du côté de la planification délibérative. Pour cela, nous n'allons pas détailler davantage ces approches.

2.2 Conclusion

Dans ce Chapitre, nous avons donné une vue globale de l'état de l'art en interaction homme-robot. Nous avons pu constater qu'il existe plusieurs approches qui traitent le problème de l'interaction homme-robot sur plusieurs niveaux. Nous nous sommes intéressés plus précisément à la prise en compte de l'homme au niveau décisionnel. Nous avons pu considérer plusieurs modèles conçus pour l'interaction avec des humains. Tous ces modèles utilisent la planification comme moteur central pour la production de plan pour l'interaction homme-robot.

Dans le chapitre suivant, nous allons nous inspirer de ces approches pour concevoir notre planificateur pour l'interaction. Ce planificateur doit donner au robot une grande autonomie décisionnelle, ce qui implique que la planification d'agenda et la planification et l'ordonnancement de tâche sont à écarter, puisqu'elles nécessitent une intervention constante des humains. Le planificateur doit également produire des plans pour le robot et pour l'humain. Il est nécessaire que ces plans soient optimaux tant pour le robot que pour l'humain. De ce fait, la planification réactive est également à écarter. Il ne reste plus que la planification délibérative qui permet de construire des plans qui sont optimaux et qui donnent au robot l'autonomie décisionnelle nécessaire pour une coopération efficace avec des humains.

3

Le planificateur dédié à l'interaction homme-robot (HATP)

Dans le chapitre précèdent, nous avons pu constater qu'il existe plusieurs approches prenant en considération l'homme dans la construction et la réalisation de leur plan tant au niveau fonctionnel qu'au niveau décisionnel. La prise en compte de l'humain dans la réalisation des actions permet de garantir un comportement socialement acceptable au niveau local (i.e. lorsque le robot exécute l'action). La prise en compte de l'humain au niveau décisionnel se traduit par la gestion des plans de de celui-ci, soit la production de plans qui aident le robot à collaborer avec lui pour la réalisation d'une tâche.

Toutefois, aucune de ces approches ne prend explicitement en compte l'impact social du plan produit par le robot. En effet, la plupart des approches présentées se focalisent sur la situation présente ou sur l'utilisation de script d'actions qui sont exécutées par le robot. Cependant, cela ne garantit pas que la série d'actions planifiées ne fasse apparaître un comportement global ou une situation qui pourrait être jugé comme socialement inacceptable. Pour cela, il faut que le processus permettant au robot de déterminer les actions à réaliser (i.e. le planificateur) intègre explicitement des notions de règles sociales, afin de produire des plans dans lesquels le comportement global du robot permettra d'assurer une bonne qualité de l'interaction homme-

robot.

Ce chapitre présente la description d'un planificateur que nous appelons HATP (Human Aware Task Planner), qui permet une socialisation du robot partenaire d'humains. Ce planificateur doit être capable de satisfaire les contraintes suivantes :

– **La prédictibilité** : le planificateur doit produire des plans qui donnent au robot un comportement cohérent au regard d'un observateur humain.
– **L'expression de l'intention** : le planificateur doit fournir au robot une structure de plans qui reflète l'intention associée à la séquence d'actions qui a été planifiée.
– **La planification mixte** : le planificateur doit produire des plans tant pour le robot que les humains partenaires.
– **La sociabilité** : le planificateur doit produire des plans socialement acceptables, c'est-à-dire qui respectent des conventions de comportements humains.
– **Le contrôle des coûts** : le planificateur doit intégrer la notion des coûts engendrés par les différentes actions du robot ou de l'humain. Cela permettra au planificateur de produire des plans plus cohérents et efficaces.

Pour réaliser ce planificateur et satisfaire ces contraintes, nous allons dans ce chapitre répondre à trois questions :

1. Quel formalisme de planification choisir ?
2. Quelles sont les extensions sur le domaine de planification ?
3. Comment évaluer la qualité sociale d'un plan ?

Il est important de noter que ce travail a été entamé avec la thèse de Vincent Montreuil [Montreuil 08].

3.1 Quelle formalisme de planification choisir ?

Le principe de fonctionnement d'un planificateur de tâche est illustré par la figure 3.1. Il repose sur trois éléments :
– **Le domaine :** il contient la description de toutes les tâches qui peuvent être réalisées.
– **La base de faits :** elle contient la description de l'état du monde sous forme de faits. En effet, elle décrit les états de toutes les entités existantes, mais également toutes les relations que celles-ci peuvent avoir. Elle est considérée comme l'état de départ ou l'état initial de la planification.
– **Le problème :** ou **le but** il représente une spécification d'un sous-état d'arrivée que le planificateur doit atteindre, sa représentation diffère en fonction des planificateurs. Il peut prendre la forme d'une conjonction de faits qui doivent être satisfaits ou bien de tâches qui doivent être réalisées.

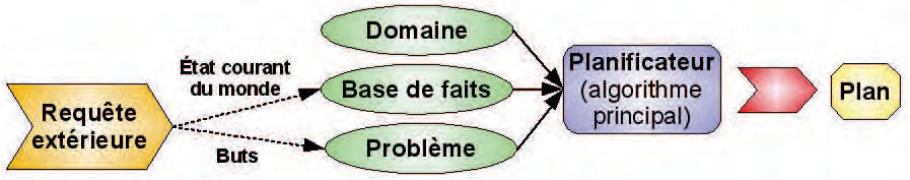


FIGURE 3.1 – Principe de fonctionnement d'un planificateur de tâche

L'objectif d'un planificateur est de trouver une combinaison d'actions appartenant à son domaine et qui relie un état initial (la base de faits) à un état d'arrivée (le problème). Alors, à chaque fois que le robot a besoin de réaliser une tâche ou d'atteindre un but, il fait appel à son planificateur de tâches en lui fournissant un état initial et un état but. En fonction des tâches du domaine, le planificateur va, soit produire une séquence de tâches qu'on appelle "plan" et qui relie l'état initial à l'état final, soit indiquer qu'il n'existe pas de plan pour relier ces deux états.

Il existe dans la littérature de nombreux formalismes de planification, les plus utilisés sont la recherche dans l'espace d'état, la recherche dans l'espace des plans partiels et la planification HTN (hierarchical task network). Cette section n'a pas pour but de faire un état de l'art de la planification de tâches. Pour un état de l'art complet, nous renvoyons le lecteur à l'ouvrage de Ghallab et al [Ghallab 04]. Parmi les nombreux formalismes existants, notre choix s'est porté sur la planification HTN pour les raisons suivantes :

– Un des premiers critères de choix d'un planificateur pour l'interaction homme-robot est le contrôle. La manière de définir un but dans les modèles non HTN est d'énumérer la liste des faits qu'on désire voir réalisés ou la liste des faits qu'on ne souhaite pas voir réalisés, mais cela n'est pas suffisant pour avoir le contrôle sur la construction du plan. En effet, tous les plans générés vont avoir ce qu'on appelle des effets de bord, qui sont des effets additionnels aux effets désirés. Le seul moyen d'avoir le contrôle, c'est de définir le but en contraignant tous les effets existants. Cette solution est clairement irréaliste. En revanche, les HTN, avec leurs structures hiérarchisées, offrent un meilleur contrôle sur la réalisation des tâches. Sauf que, du fait de ces structures, l'expressivité du but se trouve réduite.

– Les travaux sur la théorie de plans partagés, plus précisément la modélisation de l'intention pour le dialogue, ont démontré qu'il est possible de modéliser l'intention de chaque agent dans des structures hiérarchisées [Lochbaum 98]. Ces mêmes structures sont également utilisées pour le contrôle d'exécution des tâches du robot [Ingrand 96]. Il convient donc de retenir ce formalisme ou les tâches de haut niveau sont exprimées explicitement.

Nous allons maintenant détailler le modèle de planification HATP en donnant la description des différents éléments qui le composent (base de faits, domaine, problème et algorithme). Tout au long de la description, nous ferons une comparaison entre l'approche HATP et celle du planificateur SHOP2 [Nau 03].

```
factdatabase
{
    // Etape 1 : Définition des types d'entités
    define entityType Lieu;
    define entityType Objet;

    // Etape 2 : Définition des attributs
  define entityAttributes Agent
  {
    static atom string type;
    static atom number maxObjets;
    dynamic atom Lieu posTopo;
    dynamic set Objet objets;
    statique atom number handicap;
    statique atom number disposition;
  }

  define entityAttributes Object
  {
   dynamic atom Agent possesseur;
   static atom string type;
  }

  define entityAttributes Lieu
  {
    dynamic set Lieu composition;
  }

    // Étape 3 : Création des entités

    Salon = new Lieu;
    Chambre = new Lieu;
    Cuisine = new Lieu;
    Appartement = new Lieu;
    Jido = new Agent;
    Tom = new Agent;
    Verre1 = new Objet;
    Verre2 = new Objet;
    Bouteille = new Objet;

    // Etape 4 : Initialisation des attributs
    Jido.type = "robot";
    Jido.posTopo = Salon;
    Jido.objets <<= Verre1;
    Jido.objets <<= Verre2;
    Jido.maxObjets=2;
    Jido.handicap=0;

    Tom.type = "humain";
    Tom.posTopo = Salon;
    Tom.handicap=10;


    Verre1.type = "coffeeGlass";
    Verre1.possesseur = Jido;

    Verre2.type = "waterGlass";
    Verre2.possesseur = Jido;

    Appartement.composition<<=Salon;
    Appartement.composition<<=Cuisine;
    Appartement.composition<<=Chambre;
}
```

FIGURE 3.2 – Exemple de déclaration d'une base de faits avec la syntaxe HATP

3.2 Description de la structure de la base de faits

La base de faits est une structure qui décrie l'état courant du monde. Elle est composée de faits. Il existe plusieurs représentations de ces faits selon les différents planificateurs. Un fait peut être représenté par un prédicat du premier ordre, utilisé généralement dans les langages de planification (STRIPS[Fikes 71], ADL[Pednault 94], PDDL[McDermott 98]), ou par une variable d'état si on est en planification numérique ou temporelle.

Dans HATP, la base de faits est représentée par un l'ensemble fini de variables d'état appelé "entités" $Wb = \{En_1, En_2, ..., En_n\}$ avec $n \in \mathbb{N}$. Chaque entité est unique, a un type et possède un ensemble d'attributs définis par la description du domaine. Ainsi, dans HATP, un fait est défini comme l'état d'un attribut appartenant à une entité à un instant donné. La figure 3.2 donne un exemple de construction de la base de faits. Cette description se déroule en quatre étapes :

1. **Définition des types d'entités** : cette étape permet de définir les types d'entités qui vont être utilisés pour décrire l'état du monde. Dans HATP, le Type agent est prédéfini, il représente les agents qui peuvent opérer des changements sur l'état du monde.

2. **Définition des attributs** : cette étape permet d'associer à un type d'entité un ensemble d'attributs. Les attributs vont décrire l'état courant associé à une entité. Le type d'un attribut est décrit par un tuple

$$AttriType =< Ct_1 \wedge Ct_2 \wedge Ct_3 >$$

   Où :
   - $Ct_1 = \{Static \vee Dynamic\}$ détermine si la valeur de l'attribut peut varier ou non durant le processus de planification. Si la valeur de l'attribut peut évoluer, il est de type *Dynamic*, sinon il est de type *Static*.

   - $Ct_2 = \{Atom, Set\}$ détermine si l'attribut est une donnée atomique ou un ensemble de valeurs.

   - $Ct_3 = \{Numeric, Boolean, String, Entity\}$ détermine si l'attribut est un nombre, une chaîne de caractère, un booléen ou bien un type entité.

3. **Déclaration des entités** : cette étape permet de créer les entités qui vont représenter le monde.
4. **Initialisation de la base de faits** : cette étape permet d'initialiser les valeurs des attributs associés aux entités déclarées précédemment.

Dans l'exemple sur la figure 3.2. On définit un environnement constitué d'un appartement qui est lui-même composé de trois lieus (Cuisine, Chambre, salon), on définit aussi deux agents (Jido

Symbole	**définition**	**Exemple**
=	Opérateur d'affectation	Entite.attribut=2
==	Test d'égalité	Entite.attribut==2
! =	Test d'inégalité	Entite.attribut !=2
<<=	Insertion d'un élément dans un ensemble	Entite.AtrributEnsemble <<= Entite2
=>>	Suppression d'un élément d'un ensemble	Entite.AtrributEnsemble =>> Entite2
>>	Test d'appartenance à un ensemble	Entite2 >> Entite.AtrributEnsemble
! >>	Test de non appartenance à un ensemble	Entite2 ! >> Entite.AtrributEnsemble
$+; -; /; *; >; <$	Opérateurs mathématiques de base	
Fonction	**Définition**	**Exemple**
Call	fonction de calcul mathématique	Entite.attribut = Call(Entite.attribut + 2)
ForAll(type ; condition)	Fonction qui retourne un ensemble d'entités d'un certain type, qui respectent certaines conditions et leur applique certains changements	A=ForAll(Agent ; A.type=="Robot"){effets}
Select(type ; condition)	Fonction qui retourne un ensemble d'entités d'un certain type et qui respectent certaines conditions	A=Select(Agent ; A.type=="Robot")

Table 3.1 – Liste des opérateurs et fonctions utilisés dans le planificateur HATP

et Tom) qui peuvent évoluer dans ces lieux. Les agents sont décrits par un ensemble d'attributs. Par exemple, chaque agent a un type (Jido est de type robot), une position topologique (Jido est au salon), une capacité de portage (Jido ne peut porter que deux objets à la fois), une liste d'objets qu'il porte (Jido porte Verre1 et Verre2), leur degré d'handicap (Tom a un degré d'handicape de 10% et Jido est de 0%).

HATP et SHOP2 utilisent des syntaxes différentes pour la description de la base de fait. SHOP2 utilise la syntaxe et les fonctionnalités du langage Lisp. Comme HATP, il peut exprimer des symboles variables ou constants, des nombres et des listes (ensemble dans le cas de HATP). Ils ont tous deux un pouvoir expressif équivalent. Mais HATP se différencie par sa représentation particulière des états du monde, qui permet de regrouper toutes les informations concernant une entité dans une même variable. Cela facilite l'accessibilité aux données, mais permet également de construire un modèle de l'entité en question.

Prenant l'exemple d'une entité de type agent, on peut constater sur la description d'un agent ce fait sur plusieurs niveaux. On décrit son état physique, comme sa capacité et ses handicaps avec les attributs "*maxObjets*" et "*handicap*" qui représentent respectivement le nombre maximum d'objets qu'il peut porter et le degré de son handicap. On décrit son état par rapport à l'environnement avec les attributs "*PosTopo*" qui représentent sa position et l'attribut "*objets*" qui énumère la liste des objets qu'il a en main. On peut exprimer son désir, comme le désir de participer à la tâche avec l'attribut "*disposition*". Naturellement ce modèle n'est pas complet, mais il va l'être avec la définition des actions et leur modèle de coût.

3.2.1 Syntaxe de HATP

La syntaxe de HATP est fortement inspirée des langages de programmation orientées objet comme C++ ou java. Nous avons défini plusieurs opérateurs et fonctions qui vont servir pour l'initialisation de la base de faits, mais également dans la définition des préconditions et des effets des actions. Sur le tableau 3.1 nous pouvons voir la liste de tous les opérateurs et fonctions utilisées dans HATP.

3.3 Description du domaine de planification de HATP

Le domaine de planification contient le descriptif des tâches que les agents peuvent réaliser. HATP est basé sur le formalisme HTN [Nau 03]. On peut distinguer deux types de tâches ; Les actions élémentaires ainsi que les tâches de haut niveau appelées "**méthodes**".

3.3.1 Description d'une action dans HATP

Les actions HATP sont décrites par un tuple

$$Action =< Name, Par, Precond, Eff, Co, Du, >$$

Où :

– $Name$: représente le nom de l'action. C'est un symbole unique qui identifie l'action.

– Par : représente les paramètres de l'action. C'est une liste typée d'entités.

– $Precond$: représente les préconditions de l'action. Ces préconditions doivent être vérifiées avant l'application de l'action. Une précondition est une conjonction de faits. Elle est décrite par un ensemble de tests portant sur la valeur de certains attributs associés aux entités présentes dans les paramètres. Les tests peuvent être des tests d'égalité où d'inégalité $(==, !=)$ ou des tests d'appartenance ou non appartenance à un ensemble $(>>, !>>)$. Il est nécessaire de noter que la définition des préconditions supporte aussi la disjonction entre groupe de faits et cela en utilisant le symbole *OR*.

– Eff : représente les effets de l'action. Ils définissent les changements que le monde va subir si l'action est réalisée. Dans HATP un effet est défini comme changement de valeur sur les attributs des entités présentes dans la liste des paramètres. Les opérateurs qui peuvent être utilisés sont l'opérateur d'affectation $(=)$ et ajout/retrait d'un élément d'un ensemble $(<<=, =>>)$. En plus des opérateurs cités, on peut utiliser les fonctions Forall et Call. Le Forall permet d'accéder à des attributs d'entités non présentes dans les paramètres de l'action. La fonction Call a la même structure que la fonction "CALL" du langage PDDL. Cette fonction nous permet d'effectuer des calculs mathématiques sur des attributs numériques. On a aussi défini les effets conditionnels, c'est-à-dire des effets qui ne s'appliquent que si certaines

conditions qui leur sont associées sont vérifiées.

– *Co* : représente une fonction coût associée à l'action, elle permettra d'évaluer le coût lié à l'application de l'action. Cette fonction peut prendre comme arguments les paramètres de l'action ou bien des attributs liés aux entités présentes dans les paramètres, C'est une fonction qui est positive et bornée, c'est-à-dire qu'elle ne renvoie que des valeurs positives et ces valeurs ont une borne maximale [0,MAX].

– *Du* : représente une fonction de calcul du temps associée à une action, elle permet d'évaluer la durée moyenne de l'application de l'action. À l'instar de *Co*, la fonction de calcul du temps prend comme argument les paramètres de l'action ou bien des attributs d'entités présentes dans la liste de paramètres. Cette fonction est aussi bornée et positive comme pour la fonction calcul de coût.

On peut distinguer deux types d'actions. Les actions individuelles et les actions jointes [Sebanz 06]. Les actions individuelles sont les actions qui ne font intervenir qu'un seul agent. Par opposition, les actions jointes font intervenir plusieurs agents. Au niveau de la planification, une action jointe est similaire à toute autre action, sauf qu'elle présente la particularité d'être un point de convergence entre les sous-plans des agents concernés par l'action. De ce fait, cette action est un point névralgique du plan. En effet, les actions jointes exigent une attention particulière au moment de la réalisation du plan, car elles nécessitent une forte interaction/synchronisation entre les participants, d'autant plus s'ils sont de types différents (humain, robot). Pour ces raisons, il est nécessaire de faire cette distinction au niveau du planificateur, pour pouvoir en informer l'exécuteur du plan.

```
action Donner(Agent A1, Agent A2, Objet Obj)
  {
    precondition
    {  A1 != A2;
       A1.posTopo == A2.posTopo;
      Obj >> A1.objets;
      A2.chargé == false;
      Obj.détenu == true;
      Obj.détenteur == A1;};

    effects
    { A1.objets =>> Obj;
      A2.objets <<= Obj;
      Obj.détenteur = A2;
      A1.chargé = false;
      A2.charge=Call(A2.charge + 1);
      IF {A2.charge==maxObjets;}
      {A2.chargé = true;}
    };

    cost{Cout_de_donner(A1,A2)};
    duration{Duree_de_donner(A1, A2)};
  }
```

FIGURE 3.3 – Exemple de description de l'action joint Donner dans HATP.

On peut voir sur les figures 3.3 et 3.4 les déclarations des actions au niveau d'HATP. Si on examine l'action *Donner*, il apparaît que c'est une action jointe. Elle fait intervenir deux agents **A1**, **A2** et un objet **Obj**. Cette action permet de faire passer l'objet **Obj** de l'agent **A1** à l'agent **A2**. Cela ne peut se faire que si les préconditions de l'action sont respectées :

- L'agent **A1** est différent de l'agent **A2**.
- **Obj** est dans la liste des objets que possède l'agent **A1**.
- Le détenteur de l'objet **Obj** est bien l'agent **A1**.
- Les positions topologiques des deux agents sont les mêmes.
- L'agent **A2** n'a pas atteint sa charge maximale.

Lorsque l'action est réalisée, **Obj** est retiré de la liste des possessions de **A1** pour être ajouté à la liste de **A2**, et l'attribut *détenteur* de **Obj** est mis à jour. L'attribut *chargé* des deux agents est mis à jour. En ce qui concerne l'agent **A1**, comme il a donné l'objet, il ne peut plus être à vrai, alors il est mis à faux. Pour l'agent **A2**, on calcule sa charge actuelle. Si celle-ci atteint le nombre maximal d'objets que l'agent peut porter, alors l'attribut *chargé* est mis à jour.

```
action Soulever(Agent Ag, Objet Obj, Meuble F)
{
  preconditions
  {
    Ag.posTopo == F.posTopo;
    Obj.sur==F;
    Obj.possesseur=NULL;
  };

  effects
  { Obj.sur=NULL;
    Obj.possesseur=Ag;
    Ag.objets <<= Obj;
    Forall(OBJ)=(Objet; {OBJ.sous==Obj;};)
    {OBJ.visible=true;
     OBJ.sur=F;
     OBJ.possesseur=NULL;}
  };

  cost{Cout_de_soulever(Ag, Obj)};
  duration{Duree_de_soulever(Ag, Ag.posTopo, P2)};
}
```

FIGURE 3.4 – Exemple de description d'une action individuelle "Soulever" dans HATP.

L'action *Prendre* illustrée par la figure 3.4 est une action qui permet à un agent **Ag** de prendre un objet **Obj** sur un meuble **F**. Pour se faire, il est nécessaire que l'agent **Ag** soit à la même position que le meuble **F** ($Ag.posTopo == F.posTopo$). Il est obligatoire que l'objet **Obj** soit sur la table et qu'il n'ait pas de possesseur ($Obj.sur == F; \wedge Obj.possesseur = NULL;$).

Lorsque l'action est réalisée, l'objet **Obj** est ajouté à la liste des possessions de l'agent **Ag** et son attribut *possesseur* est mis à **Ag** pour indiquer que son possesseur est l'agent **Ag**, de même pour son attribut *sur* est mis à *NULL*, ce qui signifie qu'il n'est plus sur le meuble **F**. On

note que cette action peut avoir des effets supplémentaires sur des entités qui ne sont pas dans la liste des paramètres de l'action. Cela est réalisé avec la fonction Forall. Tout objet **OBJ**, qui satisfait la condition d'être sous l'objet **Obj** (*OBJ.sous==Obj ;*), se verra affecté par les effets supplémentaires de l'action, c'est-à-dire qu'il deviendra visible (*OBJ.visible=true ;*), son attribut *possesseur* sera mis à *NULL* et son attribut *sur* sera mis à *F*.

En comparant les définitions des actions dans SHOP2 [Nau 03] et dans HATP, on observe que, dans les deux approches, une action possède un nom, une liste de paramètres, une liste de préconditions, une liste d'effets et un coût associé à l'action. Les deux approches présentent plusieurs différences :

– Au contraire de SHOP2, HATP utilise des variables typées. Ce qui lui permet d'être plus efficace lors de l'instanciation des paramètres de l'action.

– Dans les préconditions, la gestion des disjonctions dans SHOP2 est beaucoup plus fine que dans HATP, car il permet des disjonctions entre prédicats, ce que ne permet pas HATP. De plus, il peut inclure des variables ne se trouvant pas dans la liste des paramètres dans ces tests des préconditions.

– SHOP2 peut faire appel à des fonctions de calculs externes, ce qui lui permet de faire de la planification numérique.

3.3.2 Description d'une méthode dans HATP

Dans les domaines de planification HTN, les tâches de haut niveau sont appelées *méthodes* [Nau 03]. Une méthode peut être décomposée en listes partiellement ordonnées de sous-tâches. Ces sous-tâches sont soit des méthodes soit des actions. Les méthodes HATP sont décrites par un tupl

$$M =< Name, Par, goal, D >$$

Où :

– $Name$ comme pour les actions $Name$ représente le nom de la méthode, c'est un symbole qui sert à identifier la méthode.

– Par représente la liste de paramètres de la méthode. Cette liste contient un ensemble d'entités.

– *goal* c'est une liste de conditions qui définit le but ou les méta-effets associés à la méthode. Si ces conditions sont respectées, alors la méthode est déjà réalisée et elle est réduite à un ensemble vide.

– D représente une liste de couples $< P_i, TL_i >$. Cette liste définit l'ensemble des décompositions possibles pour la méthode. Ainsi, pour chaque décomposition i, si la précondition P_i est valide alors la liste de tâches partiellement ordonnées TL_i est une façon

possible de réaliser la méthode.

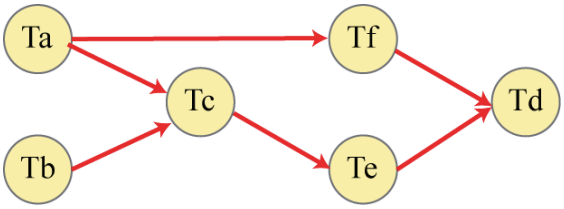


FIGURE 3.5 – Liste de tâches partiellement ordonnées

```
1 : Ta;
2 : Tb;
3 : Tc > 1, > 2;
4 : Te > 3;
5 : Tf > 1;
6 : Td > 5, > 4;
```

FIGURE 3.6 – Description de la liste de tâches partiellement ordonnées

Pour la description d'une liste de tâches partiellement ordonnées TL_i, nous avons opté pour la description qui est souple et lisible. A chaque tâche présente dans la liste, on associe un identifiant numérique. Les liens causaux sont décrits grâce à des relations de précédences entre les identifiants numériques des tâches. Ces relations de précédences sont décrites au moyen du symbole $>$.

Pour illustrer cette représentation, on considère l'exemple illustré par les figures 3.5-3.6. Dans cet exemple on peut constater que les tâches `Ta` et `Tb` sont indépendantes puisqu'elles n'ont aucune contrainte, donc elles peuvent être réalisées en parallèle. Par contre, la tâche `Tc` est précédée par `Ta` et `Tb`, ce qui signifie que la tâche `Tc` ne peut être réalisée qu'après la réalisation de `Ta` et `Tb`. La tâche `Tf` est contrainte par `Ta`, `Te` par `Tc` et enfin `Td` qui est précédée par `Te` et `Tf`.

La figure 3.7 illustre la déclaration d'une méthode nommée *TransmetreObjet*. Cette méthode permet de transmettre un objet **Obj** de l'agent **A1** à l'agent **A2**. Le but de la méthode est décrit au niveau de la condition *goal*. Si cette condition est respectée, alors cela signifie que la tâche est déjà réalisée et qu'on n'a pas à aller plus loin. Dans le cas contraire, on a deux façons de réaliser la tâche :

1. L'agent **A1** pose l'objet **Obj** quelque part pour l'agent **A2**. Cette décomposition est définie par la méthode *PoserQuelquePart*. Elle est autorisée si sa précondition est valide, c'est-à-dire si **Obj** est bien détenu par **A1**.

2. L'agent **A1** va aller donner l'objet **Obj** à l'agent **A2**. Comme pour la première décomposition ceci est défini par la méthode *AllerDonner*. Elle est aussi autorisée si sa précondition est valide.

On peut observer au niveau de la première décomposition, l'apparition d'une nouvelle variable produite par la fonction $SELECT$ dans la déclaration de la tâche $PoserQuelquePart(A1, A2, Obj, F)$. Cette fonction permet de sélectionner un ensemble de variables d'un certain type et qui satisfont certaines conditions. Dans notre exemple, la variable

```
method TransmettreObjet(Agent A1, Agent A2, Objet Obj)
  {
    goal
    { A1 != A2;
      Obj >> A2.objets;
      Obj.possesseur == A2;
    };

    //decomposition 1
    {
      preconditions
      { A1 != A2;
        Obj >> A1.objects;
        Obj.possesseur == A1;
      };

      subtasks
      {F = SELECT(Meuble, {F.libre==true;});
      1:PoserQuelquePart(A1, A2, Obj, F); };
    }

    //decomposition 2
    {
      preconditions
      { A1 != A2;
        Obj >> A1.objects;
        Obj.possesseur == A1;
      };

      subtasks
      { 1:AllerDonner(A1, A2, Obj); };
    }
  }
```

FIGURE 3.7 – Exemple de description de méthode

F doit être de type *Meuble* et son attribut *libre* doit être à vrai (c'est-à-dire que le meuble peut accueillir un objet). De ce fait, une décomposition qui comprend la fonction $SELECT$ ne va pas générer qu'une seule branche, mais N branches modulo la taille de l'ensemble de variables produites par la fonction.

En comparant les déclarations des méthodes dans SHOP2 et HATP, on s'aperçoit qu'il y a les différences suivantes :

– HATP intègre à la description de la méthode la description de son but. Avec SHOP2 on peut arriver au même résultat en ajoutant une décomposition supplémentaire, la précondition de cette décomposition va représenter le but de la méthode et sa liste de tâches va être un ensemble vide. Dans HATP, si le test du but de la méthode est valide, on ne teste pas les décompositions de celle-ci, puisque la tâche est déjà réalisée. Cependant, l'algorithme de SHOP2 va tester la validité de toutes les décompositions associées à la méthode.

– L'expression des contraintes de précédences dans SHOP2 est beaucoup plus lourde que dans HATP. En effet, Dans SHOP2, pour exprimer la précédence entre tâches, on utilise des expressions spécifiques (: $unorderd\, tasklist_1\ \ldots\ taskliste_n$), (: $orderd\, tasklist_1\ \ldots\ taskliste_n$). Le mot clé : *orderd* spécifie que la liste de tâches doit être réalisée dans l'ordre de la déclaration des tâches. Par opposition, le mot clé : *unorderd* indique que les tâches peuvent être réalisées dans n'importe quel ordre.

3.3.3 Structure de donnée d'un plan HATP

En planification classique, un plan est défini comme une séquence d'actions partiellement ou totalement ordonnées qui, à partir d'un état initial, mène à un état but. Ces actions sont déterminées séquentiellement, puis elles sont parallélisées pour produire un plan.

Dans HATP un plan a deux formes complémentaires et indissociables. Un plan séquentiel qu'on appelle *projection temporelle* et le plan hiérarchique qu'on appelle *arbre de raffinement*. La figure 3.8 illustre la structure d'un plan HATP.

Un problème HATP est représenté par un ensemble de tâches de haut niveau. La décomposition de ces tâches et de leur descendance va produire une structure hiérarchique que HATP va conserver jusqu'à atteindre le niveau des actions élémentaires. Alors, on va appeler cette structure *Arbre de raffinement*. Cette structure hiérarchisée, comme noté précédemment, va faciliter l'exécution en permettant au robot de vérifier le bien fondé de ses tâches à plusieurs niveaux d'abstraction. Elle permettra également au robot de justifier ses activités et ses intentions en se référant aux tâches de haut niveau. Cependant, il ne suffit pas de décrire complètement un plan. En effet, déterminer comment vont être réalisées les tâches de haut niveau ne permet pas de connaître l'ordre de réalisation des actions. Pour cela, l'*arbre de raffinement*

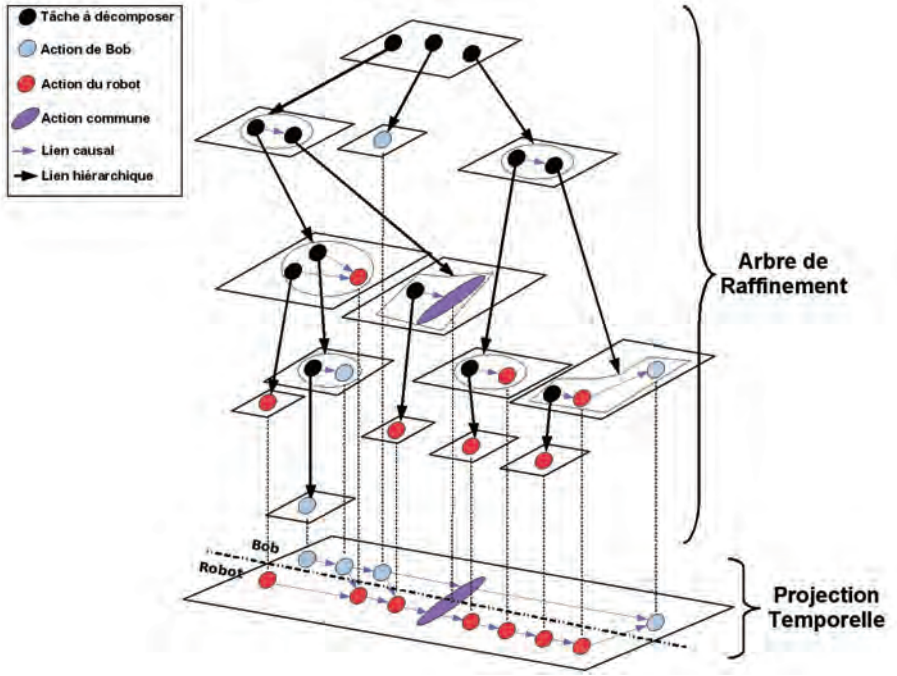


FIGURE 3.8 – Structure de plan dans HATP

et enrichi avec la structure appelée *projection temporelle*.

La *projection temporelle* est un plan parallélisé qui définit dans quel ordre les actions seront exécutées, elle fournit aussi une estimation des temps de début et de fin pour chacune des actions présentes dans le plan. À partir de l'*arbre de raffinement* et des contraintes de précédence produites par HATP, la *projection temporelle* est calculée en utilisant un ordonnanceur temporel. Celui-ci est basé sur un algorithme STNs (Simple Temporal Network) qui permet d'engendrer tous les ordonnancements possibles.

La structure de donnée d'un plan dans SHOP2 et HATP est très différente. Dans SHOP2, un plan est défini comme une séquence d'actions partiellement ordonnées, ce qui rejoint notre définition de la *projection temporelle*. Cependant, HATP se distingue avec sa définition de la structure d'*arbre de raffinement*. En effet, cette structure supplémentaire va être avantageuse, pour l'interaction homme-robot, en permettant au robot de révéler son intention et d'expliquer ses actions, mais également pour le contrôle de l'exécution du plan et la re-planification, qui seront abordés dans la section suivante.

3.3.4 Structure d'un problème de planification HATP

Comme pour tout planificateur HTN, un problème dans HATP est décrit par une liste de tâches partiellement ordonnées. La figure 3.10 illustre la syntaxe utilisée et sa représentation graphique. On observe que la liste de tâches est décrite dans un nœud avec un identifiant, cela indique que toutes les tâches se trouvent au même niveau dans la hiérarchie. La liste de tâches est décrite comme une liste de tâches dans une décomposition, c'est-à-dire que chaque tâche a

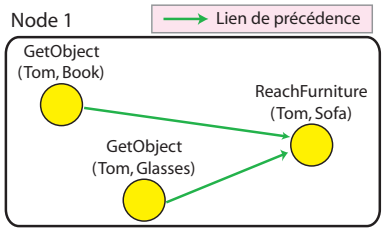


FIGURE 3.9 – Exemple de problème HATP

```
// Description des noeuds
node:1 {
    1:GetObject(Tom, Book);
    2:GetObject(Tom, Glasses);
    3:ReachFurniture(Tom, Sofa)>1, >2;
}
```

FIGURE 3.10 – Description associée

FIGURE 3.11 – Problème HATP avec contraintes de décompositions

```
// Description des noeuds
node:1 {
 1:GetObject(Tom, Book);
 2:GetObject(Tom, Glasses);
 3:ReachFurniture(Tom, Sofa)>1, >2;
 }

node:2 {
 1:GetObjectFromFurniture(Tom, Book, Shelf);
 }

node:3 {
 1:GetObjectFromFurniture(Jido, Glasses, Table)
 2:Give(Jido, Tom, Glasses) >1;
 }

// Description des liens hiérarchiques
 1:1: decomposition = node 2;
 1:2: decomposition = node 3;
```

FIGURE 3.12 – Description associée avec contraintes de décompositions

un identifiant numérique qui va servir à créer les relations de précédences entre tâches. Le but décrit dans l'exemple est le suivant : l'humain Tom souhaite prendre un livre, ses lunettes et s'asseoir sur le sofa.

Cette représentation n'est pas adaptée pour un robot interagissant avec des humains, Du fait que le planificateur ne peut pas prendre en considération les désirs et les volontés de ses partenaires humains. Pour cela, nous avons donné à HATP la capacité de planifier à partir d'une structure hiérarchique partiellement spécifiées comme celle définie dans le planificateur PASSAT [Myers 02], Ce qui permet d'imposer au problème plus de contrainte et ainsi d'accroître l'expressivité de la définition du but.

Les figures 3.11 et 3.12 illustrent le même exemple que précédemment. Cependant, dans ce cas, on prend comme hypothèse que l'humain *Tom* veut aller lui-même chercher le livre et que le robot *Jido* aille lui chercher les lunettes. On observe que les tâches de haut niveau (1) est (2) sont contraintes par les décompositions décrites dans les nœuds respectifs (2) et (3). Pour faire la liaison entre les différents nœuds, on utilise la syntaxe $\underline{Id_i : Id_j : \ decomposition = node \ Id_k}$ qui se traduit par le fait que la tâche identifiée par Id_j qui se trouve dans le nœud Id_i est décomposée en une liste de tâches décrite par le nœud Id_k.

Comme on l'a constaté dans l'exemple précédent, on peut utiliser une structure hiérarchisée pour définir un problème de planification pour HATP. Cette structure hiérarchisée est appelée plan partiel, car c'est un *arbre de raffinement* partiellement spécifié. Pour rappel, Dans la section précédente, on a identifié l'*arbre de raffinement* comme étant un plan solution.

Il paraît clair que la définition d'un problème de planification dans HATP est beaucoup plus riche que celle utilisée dans SHOP2, du fait qu'il exploite la notion de "plan partiel" pour la définition des problèmes de planification. L'utilisation de plans partiels dans HATP apporte deux avantages majeurs. (1) Elle permet d'améliorer l'interaction entre l'humain et le robot, et cela en permettant au partenaire humain d'exprimer ses souhaits sur la façon de réaliser le plan. (2) Elle permet d'accroître la réactivité du robot, et cela en permettant une re-planification à partir d'un plan déjà existant.

3.4 Extension sur le domaine de planification

Dans cette section, nous allons aborder la description de l'extension que nous avons ajoutée à la définition du domaine de planification pour pouvoir produire des plans socialement acceptables.

3.4.1 Critères sociaux pour l'interaction

Pour pouvoir produire des plans qui soient socialement acceptables, il est impératif que ces plans respectent certains critères sociaux pour l'interaction. Dans la suite de cette section nous allons commencer par identifier et illustrer certains critères que nous avons identifiés, et par la suite nous donnerons leur description dans le domaine de planification de HATP.

Le coût et le temps de la réalisation

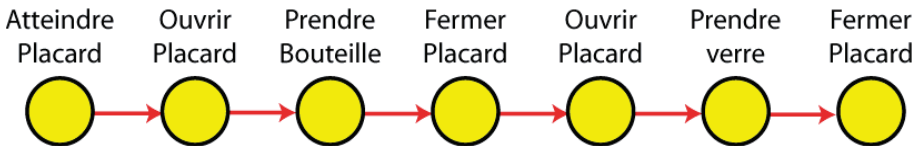


FIGURE 3.13 – plan produit sans prise en compte de critère

La prise en compte de ces deux critères est nécessaire pour l'interaction homme-robot. En effet la réalisation d'une tâche simple qui perdure trop dans le temps devient lassante pour l'humain. Il en est de même pour les plans qui ne sont pas optimaux, ils peuvent susciter l'incompréhension de l'humain. La figure 3.13 illustre un plan où le robot doit aller chercher un verre et une bouteille se trouvant tous les deux dans un placard. Du fait que le robot ouvre et ferme la porte plusieurs fois pour prendre le verre et la bouteille, son but n'apparaît pas

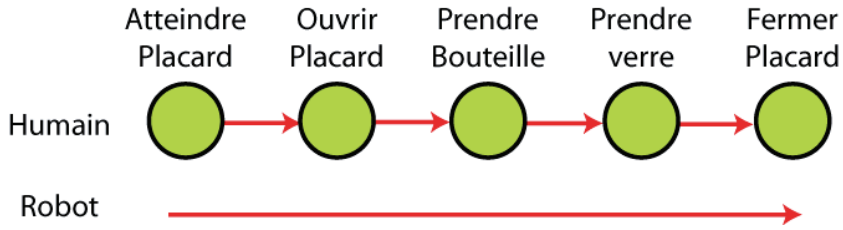


FIGURE 3.14 – Exemple avec une mauvaise gestion des efforts

clairement, voire est incompréhensible.

La prise en compte du coût et du temps de réalisation d'un plan durant la planification permet de produire des plans plus efficaces. En effet, un plan efficace en termes de coût et de temps, améliore la qualité de l'interaction homme-robot en évitant de susciter des sentiments d'incompréhension et de lassitude chez l'humain. Ces deux critères sont intégrés au noyau du planificateur, et l'utilisateur n'a aucun contrôle sur eux.

La gestion des efforts

Un planificateur qui ne prend en compte que l'optimalité des coûts peut produire des plans où les efforts ne sont concentrés que sur un seul agent. L'exemple de la figure 3.14 illustre un plan ou le but est que l'humain souhaite boire, dans cet exemple nous considérons que le coût des actions et proportionnellement inverse aux compétences de l'agent et que l'humain et plus compétent pour réaliser certaines tâches. Nous pouvons constater que ce plan et tout à fait réalisable, mais peut susciter une incompréhension de la part de l'humain du fait que toutes les tâches lui sont affectées. En effet les humains, quand ils travaillent en équipe, essaient de partager les tâches équitablement entre tous les membres de l'équipe. Dans le cas d'une équipe homme-robot il est normal que le robot produise plus d'effort que l'humain du fait qu'il est à son service. Le but de cette règle est de réguler les efforts entre humain et robot et d'imposer le fait qu'un robot doit fournir plus d'effort qu'un humain même si celui-ci est plus compétent pour réaliser la tâche.

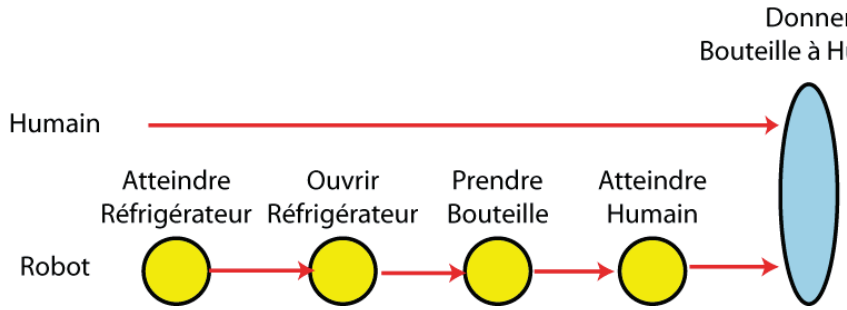
```
effortBalancing
{
   priority = -2;

   penalty{computeEffortBalancing()};
}
```

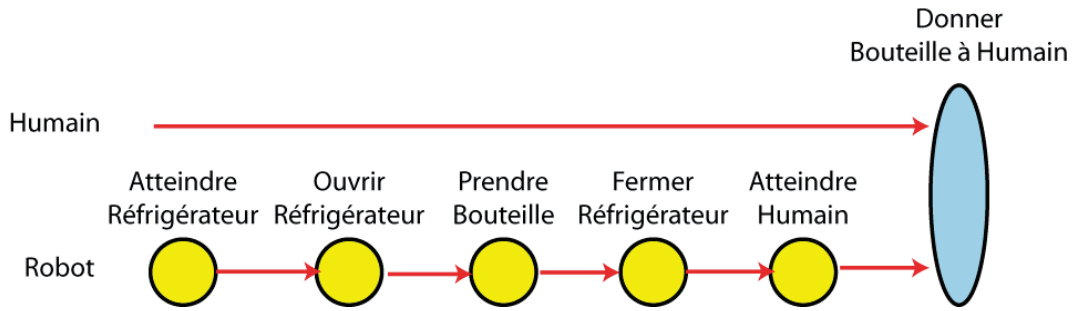
FIGURE 3.15 – Exemple de description de la règle gestion d'efforts

La syntaxe utilisée pour la description dans le domaine de planification est illustrée par la figure 3.15. La description d'une règle sociale dans le planificateur commence toujours par un mot (clef dans ce cas : "**effortBalancing**") qui permet d'identifier la règle. Le deuxième élément est la priorité. Cette priorité permet d'indiquer le degré d'importance de la règle par

rapport aux autres règles (l'utilité de cet élément va être précisée dans la section 3.4.2). Le troisième élément est la fonction de pénalité si la règle est violée. Cette fonction est externe au planificateur. C'est à l'utilisateur de la déterminer en fonction de ses besoins et des besoins du domaine de planification.



(a) Exemple de plan contenant un état indésirable



(b) Exemple de plan ne contenant pas un état indésirable

FIGURE 3.16 – Exemple pour les états indésirables

Les états indésirables

Il arrive parfois qu'un plan dit optimal par rapport aux coûts des actions produit ce qu'on appelle des états indésirables. Ces états peuvent être gênants, désagréables, ou carrément dangereux pour l'humain partenaire du robot. Prenons l'exemple où le robot doit aller chercher une bouteille d'eau pour l'humain, la bouteille d'eau est au réfrigérateur. Examinons les deux plans illustrés par les figures 3.16(a) et 3.16(b). Si on n'évalue ces plans que du point de vue de l'optimalité, il est clair que le plan 3.16(a) est plus optimal que celui de la figure 3.16(b), mais du point de vue qualité sociale ce plan présente un désavantage certain, puisque dans ce plan le robot laisse la porte du réfrigérateur ouvert. Cela peut être une situation dérangeante pour l'humain.

Cet exemple démontre que, dans certaines situations, il est préférable de sacrifier l'optimalité du plan au profit de la sociabilité de celui-ci. Il est donc nécessaire de prendre en compte le critère d'états indésirables pour pouvoir produire un plan convenable pour l'interaction homme robot. Le plus gênant dans notre exemple n'est pas que l'état indésirable apparaisse dans le plan, mais c'est le fait qu'il perdure tout le long du plan. Il y a aussi des états indésirables qui deviennent dérangeants dès leur apparition, par exemple le fait qu'un verre d'eau soit posé sur un téléviseur.

Il est clair que dans l'interaction homme robot il y a une multitude d'états indésirables. Alors, il devient nécessaire d'avoir un ordre de priorité entre tous ces états indésirables.

Nous ne sommes pas les premiers à nous être intéressés aux états indésirables dans un plan. Il y a également les travaux de [Kvarnstrom 01] sur le planificateur TALPlanner. Ce planificateur utilise des règles écrites en logique temporelle, celles-ci sont utilisées durant la phase de recherche pour élaguer des branches qui ne les respectent pas. Ce qui lui permet de réduire l'espace de recherche. Dans notre cas, les états indésirables ne sont pas considérés comme interdits, mais leur présence est indésirable.

```
undesirableState ObjectOnChair
{
  priority = -2;

  Furniture Fur;
  Object OBJ;

  conditions
  {
    Fur.type == "chair";
    OBJ >> Fur.objectsOn;
  }

  penalty{state1Pen};
}
```

FIGURE 3.17 – Exemple de description de la règle états indésirables

Les états indésirables sont décrits par un ensemble de faits particuliers. On considère que la règle est violée si et seulement si tous les faits sont vrais au même moment. Pour incorporer cette règle au domaine de planification de HATP, nous utilisons la syntaxe illustrée par l'exemple de la figure 3.17. On commence par donner la priorité à la règle *balance des efforts*. Ensuite on déclare les variables typées nécessaires à la description. Les variables sont utilisées pour décrire la conjonction des faits qui forment le noyau de la règle. Le test des états des faits est réalisé de la même façon que pour le test d'une précondition. Enfin, à chaque état indésirable on associe une fonction qui calcule la pénalité occasionnée par la règle si elle est violée.

Observons l'exemple de la figure 3.17. Cette règle va décrire l'état indésirable " *ObjectOnChair*" où l'on ne souhaite pas qu'un objet soit posé sur un certain type de meuble. Pour décrire cet état indésirable, on a besoin de deux variables. La première est de type *Furniture* et représente un meuble, la seconde est de type *Object*. La première condition permet de préciser le type de meuble (une chaise dans ce cas précis) et la seconde condition indique que l'objet est sur le meuble. Si ces deux conditions sont vérifiées au même moment, cela indique que la règle a été violée et que le plan sera pénalisé.

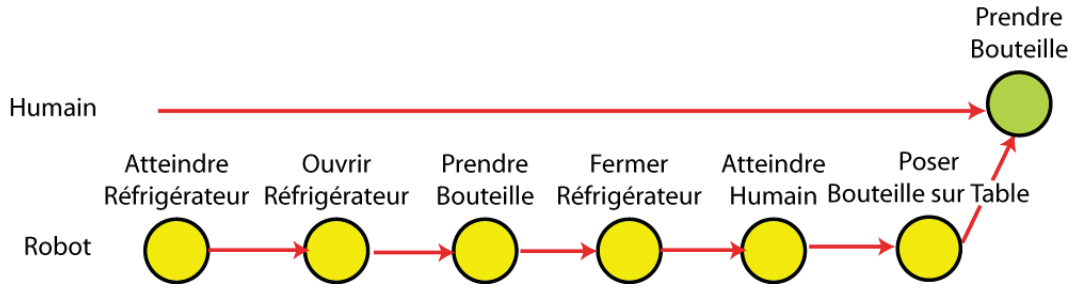


FIGURE 3.18 – Exemple de plan avec une séquence indésirable

Les séquences indésirables

La prise en compte des coûts de réalisation d'un plan peut provoquer l'apparition ou la persistance d'états indésirables, mais peut également provoquer l'apparition de combinaisons d'actions suscitant l'incompréhension chez les partenaires humains du robot. Un exemple d'une telle situation est illustré par la figure 3.18. Dans cet exemple, on peut constater que pour transmettre un objet à l'humain, il fait le choix de le déposer sur la table, forçant l'humain à développer un effort supplémentaire pour le saisir. Il paraît évident que cette situation est dérangeante pour l'humain qui s'attend à recevoir l'objet en mains propres. En effet, il vaut mieux dans cette situation que le robot donne l'objet à son partenaire humain en mains propres même si cela à un coût de réalisation plus important.

Pour produire des plans socialement acceptables, il est nécessaire de prendre en compte le critère des séquences indésirables, puisqu'il permettra d'améliorer la qualité sociale des plans, en évitant l'apparition des combinaisons d'actions particulières pouvant susciter l'incompréhension des partenaires humains du robot.

```
undesirableSequence seq12
{
  priority = -2;

  Agent ROBOT, HUMAN;
  Furniture FURN;
  Object OBJ;

  conditions
  {
    ROBOT.type == "robot";
    HUMAN.type == "human";
  }

  sequence
  {
     1:Putdown(ROBOT, OBJ, FURN);
     2:Pickup(HUMAN, OBJ, FURN) ~> 1;
  }

  penalty{seq1Pen};
}
```

FIGURE 3.19 – Exemple de description de la règle *séquences indésirables*

Les séquences indésirables sont représentées par une liste d'actions partiellement ordonnées qui se trouve au plus bas niveau d'un plan. Pour incorporer cette règle au domaine de planification de HATP, nous utilisons la syntaxe illustrée par l'exemple de la figure 3.19. Comme pour toutes les règles, on commence par donner la priorité à la règle. La description continue par un ensemble de variables qui sont nécessaires à la description de celle-ci. Nous poursuivons par un ensemble de conditions qui servent à contraindre les variables. Ensuite on décrit la séquence d'action proprement dite, la syntaxe utilisée est très similaire à celle utilisée pour la description d'une liste de tâches dans les décompositions d'une méthode. Sauf qu'on a un nouveau symbole "~>" qui permet de décrire un lien de précédence éventuellement éloigné, alors que ">" exprime un lien de précédence immédiat. Les deux liens peuvent être utilisés pour décrire la séquence d'actions. Enfin, la description de la règle se termine par la déclaration de la fonction qui permet le calcul de la pénalité à ajouter au plan, si la règle est violée.

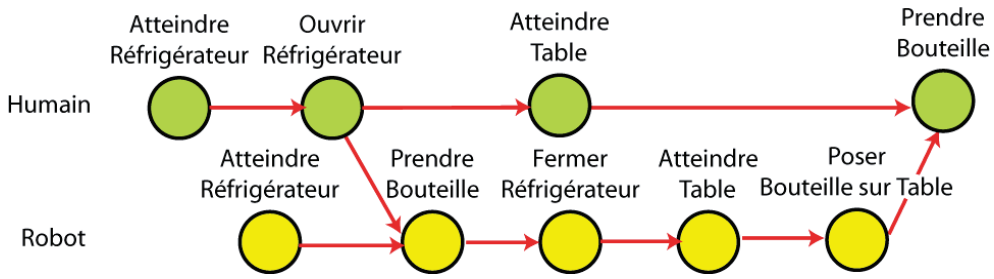


FIGURE 3.20 – Exemple de plan avec des temps morts pour l'humain

Les temps d'inactivité

Les temps d'inactivités ou temps morts sont des temps où un agent reste inactif entre deux actions successives. Ce temps d'inactivité est effectivement préjudiciable pour la qualité sociale d'un plan quel que soit le type de l'agent (humain ou robot). Dans le cas des humains il peut donner le sentiment de perte de temps, ou bien susciter l'agacement et augmenter le risque d'abandon de la tâche. Dans le cas du robot, cela risque de lui donner une image négative dans le sens où il est vu comme inutile par des observateurs humains. La figure 3.20 illustre un exemple où l'humain doit attendre que le robot ait fini ses tâches pour pouvoir réaliser les siennes.

```
westedTime
{
  priority = 0;

  Agent={Jido, Tom, John};
  penalty{wastedTimeFunction ( Agent )};
}
```

FIGURE 3.21 – Exemple de description de la règle des temps d'inactivité

L'introduction du critère des temps d'inactivités dans l'estimation de la qualité sociale d'un

temps mort évite ainsi de véhiculer une image d'inactivité du robot et/ou de susciter l'impatience des partenaires humains. L'exemple de la figure 3.21 illustre la syntaxe utilisée pour la description de cette règle. Après avoir instancié la valeur de la priorité de la règle, on décrit l'ensemble des agents concernés par celle-ci. On finit par la déclaration de la fonction qui calcule la pénalité. Lorsque le planificateur trouve un plan, il associe à chaque agent la liste de ses actions avec la chronologie de leur apparition dans le plan. À partir de là, on peut calculer le temps d'inactivité pour chaque agent et calculer la pénalité que l'on va donner au plan.

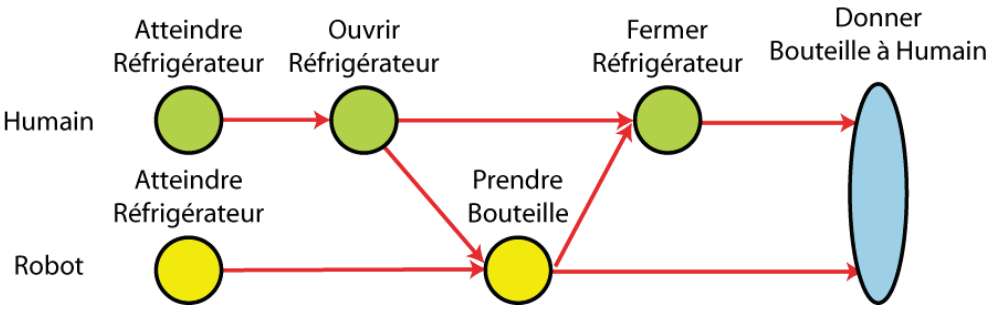


FIGURE 3.22 – Exemple d'un plan avec un lien croisé

Les liens croisés

Les liens croisés représentent le nombre de liens de synchronisation ou de précédence qu'il peut y avoir entre les activités des différents agents. Un plan où il a beaucoup de liens croisés est un plan où il y a beaucoup de dépendances inter-agents comme illustré dans la figure 3.22. En effet ce type de plan peut engendrer chez les humains un sentiment d'emprisonnement dans la tâche, puisqu'ils sont obligés de surveiller les activités du robot pour déterminer quand ils peuvent commencer leurs propres activités. De plus, les plans qui ont un grand nombre de liens croisés sont considérés comme des plans fragiles, puisqu'ils présentent un grand risque d'échec. Puisque, si une action de l'humain ne peut être réalisée, qu'après l'action du robot, cela augmente les risques des temps d'attentes pour l'humain. Dans le cas contraire, c'est-à-dire si le robot doit attendre l'humain, cela oblige le robot à focaliser toute son attention vers l'humain, ce qui augmente les risques d'erreur sur la perception.

```
controlOfIntricacy
{
  priority = 4;

  Agent={Jido, Tom};

  penalty{computeControlOfIntricacy};
}
```

FIGURE 3.23 – Exemple de description de la règle des *liens croisés*

Un nouveau critère doit donc être introduit afin de favoriser les plans avec un nombre limité de liens croisés, c'est-à-dire le nombre de liens causaux ayant pour origine une action d'un agent et pour extrémité une action d'un autre agent. Ce critère permettra de limiter les plans engendrant des sentiments de dépendance chez les partenaires humains du robot, et de limiter les plans dont l'exécution peut s'avérer fragile.

La description utilisée pour représenter les *liens croisés* est semblable à celle utilisée pour décrire la règle *temps d'inactivité* comme le montre la figure 3.23. Comme pour toutes les règles on commence par définir la priorité de la règle. On poursuit en établissant la liste de tous les agents concernés. Une fois qu'un plan a été trouvé, la planification comptabilise le nombre de liens existant entre tous ces agents, et utilise la fonction déclarée à la fin pour calculer la pénalité qui sera ajoutée au plan.

Les mauvaises décompositions

Un des critères les plus importants est l'intelligibilité des actes d'un robot, c'est-à-dire la cohérence entre ses actions et son intention. Des études ont montré qu'on pouvait modéliser les intentions dans une structure hiérarchisée [Grosz 98], [Lochbaum 98] et en déduire l'intention de chaque agent à partir de la tâche dans laquelle il est impliqué. En examinant la hiérarchie de tâches illustrée dans la figure 3.24, on s'aperçoit que l'agent qui réalise l'action *PutdownIn* le fait dans l'intention de réaliser la tâche *PutdownObjectForSomeone*, qui est elle-même réalisée dans l'intention d'effectuer la tâche *TransmitObject*. Notons que ces mêmes structures sont utilisées lors de l'exécution pour superviser la réalisation des plans. Le système PRS (Procedural Reasoning System) [Ingrand 96] utilise une structure hiérarchique pour contrôler la cohérence et la pertinence des tâches qu'il est en train de réaliser à diffèrents niveaux d'abstraction.

Toutefois, dans certains cas, ces structures hiérarchiques présentent une incohérence entre les actions engendrées et l'intention qu'elle modélise. Considérons l'exemple illustré par la figure 3.24. Cet exemple décrit une des façons qu'a un agent de transmettre un objet à un autre agent. On constate que, pour réaliser la tâche, l'agent dépose l'objet sur un meuble pour que l'autre agent le récupère. Il paraît clair que déposer un objet sur un meuble pour un autre agent ne correspond pas du tout à l'intention de transmettre un objet, ce qui peut susciter l'incompréhension chez l'humain. Il s'attend à ce que l'objet lui soit transmis en main propre ou bien, au pire des cas être déposé à coté de lui.

Il est donc nécessaire d'introduire ce critère pour détecter les intentions floues dans la structure arborescente du plan, afin de favoriser des plans assurant une meilleure intelligibilité des actes et garantissant un contrôle d'exécution plus adapté. La description d'une mauvaise décomposition passe obligatoirement par la description d'une structure arborescente composée de listes de tâches partiellement ordonnées. La figure 3.25 illustre un exemple d'une mauvaise

décomposition. La description débute par la spécification de la priorité de la règle, on enchaîne par la déclaration des variables nécessaires à la description de la règle. On définit ensuite les contraintes sur ces variables. L'étape suivante consiste à décrire la structure arborescente. On commence par la description de la liste de tâches pour chaque nœud. La syntaxe utilisée est semblable à celle utilisée pour décrire les séquences indésirables. On ajoute la description des liens hiérarchiques qui permettent de lier une tâche d'un nœud à un autre se trouvant plus bas dans la hiérarchie. Comme pour la règle des *séquences indésirables* et la notion de lien immédiat et lien éloigné, on utilise dans la hiérarchie les notions de lien de décomposition immédiat et de lien de décomposition profond pour représenter les descendants directs et indirects d'une décomposition. La figure 3.24 illustre la représentation graphique de la hiérarchie qu'on a définie dans l'exemple de la règle (figure 3.25).

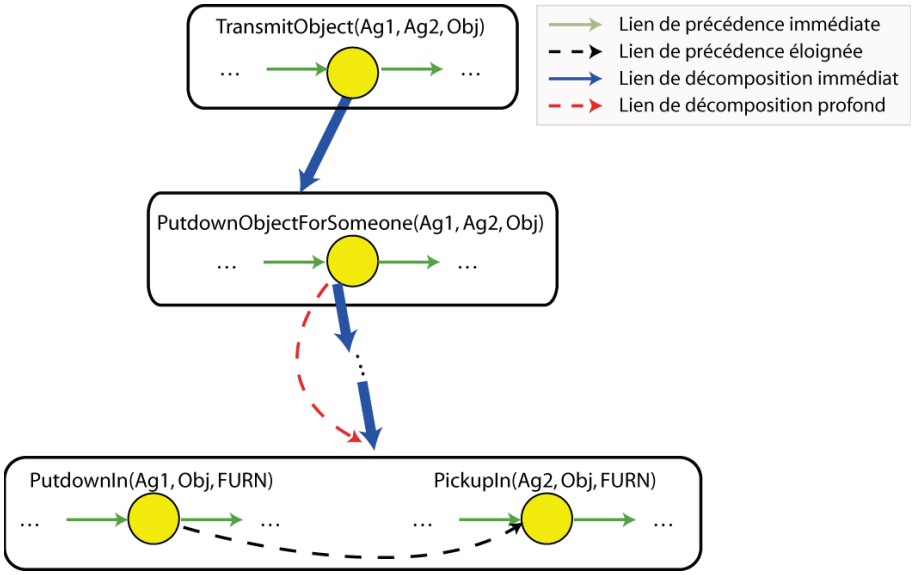


FIGURE 3.24 – Représentation graphique de la mauvaise décomposition décrite

3.4.2 Métrique et Analyse multicritères

À présent que nous avons déterminé les critères qui influencent la qualité sociale d'un plan, il reste à trouver une métrique qui permette la gestion de plusieurs critères et qui reflète le plus fidèlement possible la qualité sociale d'un plan.

La comparaison de deux plans sur un critère donné est relativement simple. Cependant, dans le cadre d'une planification avec prise en compte de la qualité sociale du plan, cette comparaison va porter sur l'ensemble des critères définis dans la description du domaine. Il va donc falloir choisir entre plusieurs plans solutions en se basant sur divers critères de décision simultanément. Ce type de décision relève donc de l'analyse multicritère.

Le principe de l'analyse multicritère est d'agréger plusieurs critères afin de prendre une

```
badDecomposition badDec1
{
  priority = -3;

  Agent Ag1, Ag2;
  Furniture FURN;
  Object Obj;

  conditions
  {
    Ag1 != Ag2;
  }

  decomposition
  {
    node:1
    {
      1:TransmitObject(Ag1, Ag2, Obj) ;
    }

    node:2
    {
      1:PutdownObjectForSomeone(Ag1, Ag2, Obj)
    }

    node:3
    {
       1:PutdownIn(Ag1, Obj, FURN);
       2:PickupIn(Ag2, Obj, FURN) ~> 1;
    }

    1:1:immediateDecomposition = node 2;
    2:1:deepDecomposition = node 3;
  }
```

FIGURE 3.25 – Exemple de description de la règle *les mauvaises décompositions*

décision basée sur des objectifs parfois contradictoires. D'après [Ben-Mena 00] l'analyse multicritère s'opère en quatre étapes :

1. Dresser la liste des alternatives (solutions) possibles.
2. Dresser les critères à prendre en compte dans la décision.
3. Établir le tableau des performances.
4. Agréger les performances pour déterminer la meilleure solution.

L'étape 3 va permettre de construire le tableau des performances ou la matrice de décision. Cette matrice est illustrée par le tableau 3.2. Elle est constituée, en ligne des alternatives, et en colonne les critères. Chaque élément a_{ij} représente les coûts ou les profits associés à l'alternative i selon le critère j. On considère alors chaque alternative comme un vecteur de N dans l'espace des critères. Les trois premières étapes sont communes à toutes les méthodes d'analyse multicritère et ne présentent que de faibles variations. La quatrième présente, elle, une diversité nettement supérieure et dépend de la méthode utilisée.

Il existe plusieurs approches pour l'analyse multicritère, les plus employées sont : WSM (*Weight Sum Method* ou somme pondérée), WPM (*Weight Product Method* ou Produit de ratios)

		Critères				
		C_1	C_2	C_3	...	C_N
Alternatives	A_1	a_{11}	a_{12}	a_{13}	...	a_{1N}
	A_2	a_{21}	a_{22}	a_{23}	...	a_{2N}
	...	...	...	...	...	...
	A_M	a_{M1}	a_{M2}	a_{M3}	...	a_{MN}
Poids relatifs		W_1	W_2	W_3	...	W_N

TABLE 3.2 – Matrice de décision pour une analyse multicritère

et AHP (Analytic Hierarchy Process ou Processus d'Analyse Hiérarchique). La somme pondérée est basée sur le principe d'une utilité additive pour chaque critère. Pour déterminer la solution à retenir, on cherche l'alternative i telle que :

$$A(a_i)_{WSM} = \min_i \left(\frac{\sum_{j=1}^{N} a_{ij} \times W_j}{\sum_{j=1}^{N} W_j}\right) \text{ pour i = 1 à M}$$

Le Produit de ratios et très similaire à la méthode de la somme pondérée, dans le sens ou l'on remplace la somme par un produit suivant cette formule :

$$A(a_i)_{WPM} = \min_i \prod_{j=1}^{N} \left(\frac{a_i}{\sum_{i=1}^{M} a_{ij}}\right)^{W_j} \text{ pour i = 1 à M}$$

Le principal défaut de la somme pondérée est sa grande sensibilité aux changements d'échelle et aussi à la possibilité de compensation entre critères. Ces défauts sont corrigés par le produit de ratio, mais nécessite que chaque échelle de critère aille dans le même sens. Pour ces raisons, nous nous sommes tournés vers l'AHP (Processus d'Analyse Hiérarchique) qui ne présente aucun de ces défauts et qui se distingue par sa façon de déterminer les poids de critères.

3.4.2.1 Processus d'Analyse Hiérarchique (AHP)

Dans cette section nous allons décrire brièvement le principe de l'AHP. Ce processus a été développé dans les années 70 par Thomas Saaty. Son principe d'application est détaillé dans [Saaty 00][Forman 01]. Cette méthode générique a été conçue pour faciliter la prise de décision basée sur plusieurs critères (parfois intangibles) dans des environnements complexes. Cette méthode a été utilisée dans de nombreuses applications. Les Raisons de sa popularité sont :

– Unités de mesures : Qualitatives et quantitatives, valeurs relatives ou absolues pour établir les priorités.

– Structure hiérarchique : Tri des éléments d'un système dans différents niveaux et dans des groupes à caractéristiques similaires.

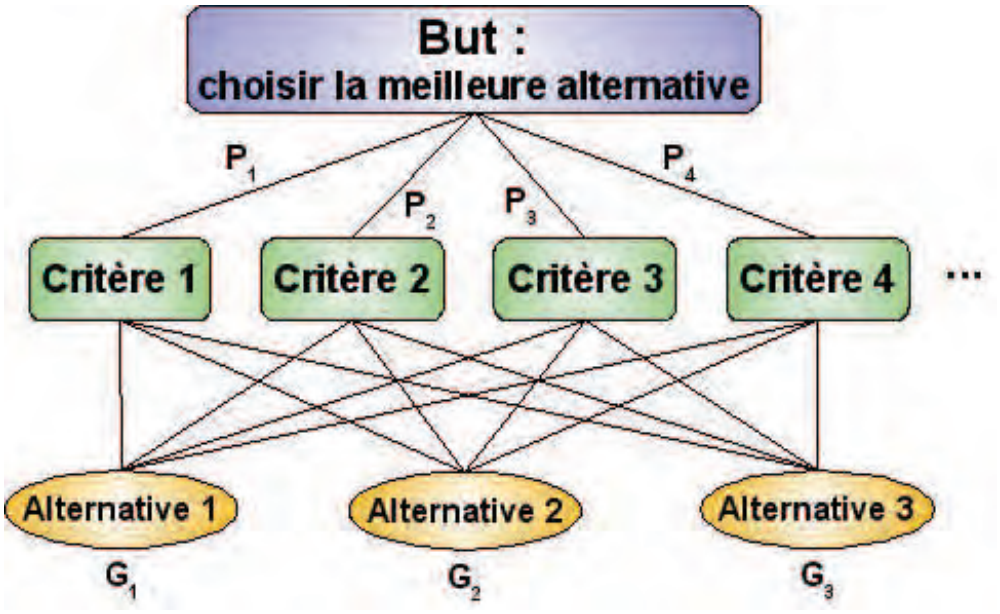


FIGURE 3.26 – Modélisation d'un problème avec le processus AHP

– Interdépendance : Permet de considérer l'interdépendance des éléments d'un système.

– Consistance : Permet de garder une consistance logique des règles utilisées pour déterminer les priorités.

– Synthèse : Permet d'obtenir une appréciation générale de la priorité de chaque alternative.

– Identification des priorités : Permet de considérer la priorité relative de chaque critère pour obtenir ainsi la meilleure alternative selon les objectifs identifiés.

Le processus de décision AHP permet donc de modéliser un problème d'optimisation multicritères qualitatifs sous la forme d'une hiérarchie comme l'illustre la figure 3.26. Cette structure hiérarchique peut éventuellement contenir des niveaux supplémentaires si un critère peut se décomposer en une combinaison de sous-critères. Le principe de la méthode AHP est d'obtenir pour chaque alternative une valeur numérique représentant sa priorité, puis de retenir l'alternative ayant la plus grande priorité. Pour appliquer la méthode AHP, il faut suivre les étapes suivantes :

1. Déterminer toutes les combinaisons binaires entre critères,
2. Déterminer les priorités locales entre critères,
3. Synthétiser les priorités globales ρ_i entre critères,
4. Pour chaque critère considéré indépendamment des autres, déterminer la priorité locale de chaque alternative,
5. Synthétiser la priorité globale G_j de chaque alternative.

Les étapes 1 et 2 se concrétisent par la conception de la matrice de comparaison des paires de critères. Il s'agit d'une matrice carrée $N \times N$ (N étant le nombre de critères) de la forme :

Importance qualitative entre les critères i et j	**Valeur numérique de C_{ij}**
Le critère i est extrêmement moins important que le critère j	1/9 1/8
Le critère i est énormément moins important que le critère j	1/7 1/6
Le critère i est beaucoup moins important que le critère j	1/5 1/4
Le critère i est légèrement moins important que le critère j	1/3 1/2
Le critère i est d'importance égale avec le critère j	1
Le critère i est légèrement plus important que le critère j	2 3
Le critère i est beaucoup plus important que le critère j	4 5
Le critère i est énormément plus important que le critère j	6 7
Le critère i est extrêmement plus important que le critère j	8 9

TABLE 3.3 – Tableau de priorité qualitative entre critères

	Critère 1	**Critère 2**	**Critère 3**	...	**Critère N**
Critère 1	1	C_{12}	C_{13}	...	C_{1N}
Critère 2	C_{21}	1	C_{23}	...	C_{2N}
Critère 3	C_{31}	C_{32}	1	...	C_{3N}
...	...	...	...	...	...
Critère N	C_{N1}	C_{N2}	C_{N3}	...	1

Les coefficients C_{ij} représentent le degré d'importance numérique du critère i par rapport au critère j. Pour déterminer ces coefficients, on utilise une échelle comparative qualitative des critères en les comparant deux à deux. Le tableau 3.3 permet ensuite de convertir ces comparaisons en une évaluation numérique. Pour que la matrice globale soit cohérente, certaines contraintes doivent être respectées :

$$\begin{cases} \forall i,\ C_{ii} = 1 \\ \forall i,\ \forall j \neq i,\ C_{ij} \times C_{ji} = 1 \\ \forall i,\ \forall j \neq i,\ \forall k \neq i \text{ et } k \neq j,\ C_{ik} = C_{ij} \times C_{jk} \end{cases} \tag{3.1}$$

Ainsi, la matrice de comparaison des paires de critères contient uniquement des 1 sur sa diagonale puisque le degré d'importance d'un critère ne saurait être différent de lui-même. La deuxième contrainte permet de s'assurer que le degré d'importance C_{ji} est l'inverse du degré d'importance C_{ij}. Ainsi, si le critère i est beaucoup plus important que le critère j, alors, pour rester cohérent, il faut que le critère j soit beaucoup moins important que le critère i. La dernière contrainte permet de maintenir la cohérence globale de la matrice, à savoir si le critère i a un degré d'importance C_{ij} par rapport au critère j et que ce dernier a un degré d'importance C_{jk}

par rapport au critère k alors le critère i doit avoir un degré d'importance $C_{ik} = C_{ij} \times C_{jk}$ par rapport au critère k.

L'étape 3 de la méthode AHP permet de déterminer les priorités globales entre les critères. Pour cela, il faut simplement calculer la valeur du premier vecteur propre. Puisque Saaty explique dans son livre [Saaty 00] que du fait de la particularité de la matrice, il n'est pas nécessaire de calculer tous les vecteurs propres de celle-ci, puisque le premier vecteur propre est le vecteur dominant et que chaque élément correspond au degré de dominance du critère avec lequel il est accordé. Saaty propose aussi une technique qui permet d'estimer la valeur de ce premier vecteur propre et cela sans passer par des calculs matriciels coûteux. La procédure est décrite comme suit :

3.1 Faire la somme des valeurs de chaque colonne.

3.2 Diviser chaque élément de la matrice par le total de sa colonne.

3.3 Calculer la moyenne des éléments de ligne de la matrice. On obtient un vecteur $[P_0; P_1; ...; P_N]$ où P_i est le coefficient de priorité du critère i.

On a donc :

$$\rho_i = \frac{1}{N} \times \sum_{j=1}^{N} \left(\frac{C_{ij}}{\sum_{k=1}^{N} C_{kj}} \right) \tag{3.2}$$

Les coefficients de priorité vont constituer le vecteur P de dimension N qui correspond à la dimension des critères. Ce vecteur doit vérifier la propriété : $\sum_{i=1}^{N} \rho_i = 1$. Plus le degré d'importance d'un critère i par rapport à l'ensemble des autres critères sera grand, plus le coefficient de priorité ρ_i sera important.

Les étapes 4 et 5 du processus AHP consistent à déterminer les priorités locales et globales de chaque alternative. Pour cela, on utilise la même échelle comparative que précédemment (voir tableau 3.3 qui permet d'exprimer un degré de préférence numérique entre les alternatives selon un critère donné). Pour chaque critère i, on écrit la matrice de second degré correspondante M_i :

	Alternative 1	**Alternative 2**	...	**Alternative M**
Alternative 1	1	α^i_{12}	...	α^i_{1M}
Alternative 2	α^i_{21}	1	...	α^i_{2M}
...	...	...	...	...
Alternative M	α^i_{M1}	α^i_{M2}	...	1

Les coefficients α^i_{tu} des matrices de second degré doivent vérifier les mêmes propriétés de cohérence que les coefficients de la matrice de comparaison des paires de critères (formule 3.1). Pour chaque critère i on détermine la priorité locale γ^i_t d'une alternative t en utilisant le même principe que pour le calcul des priorités locales entre critères. On obtient donc une équation

similaire à l'équation 3.2 :

$$\gamma_t^i = \frac{1}{M} \times \sum_{u=1}^{M} \left(\frac{\alpha_{tu}^i}{\sum_{v=1}^{M} \alpha_{vu}^i} \right) \tag{3.3}$$

De même que pour les coefficients de priorité, les priorités locales vérifient la propriété $\sum_{t=1}^{M} \gamma_t^i = 1$. Pour déterminer la priorité globale G_t associée à chaque alternative t, on combine les priorités locales en utilisant les priorités globales déterminées précédemment. La préférence globale G_t reflète la priorité de l'alternative t, et s'obtient en combinant les équations 3.2 et 3.3.

$$G_t = \sum_{i=1}^{N} \rho_i \times \gamma_t^i \tag{3.4}$$

Où :
- N est le nombre de critères,
- ρ_i est la priorité globale du critère i,
- γ_t^i est la priorité locale de l'alternative t selon le critère i.

L'alternative finalement retenue est celle qui possède la plus grande priorité.

3.4.2.2 Intégration de l'AHP au sein du planificateur HATP

HATP utilise l'approche AHP pour évaluer la qualité sociale des plans produits dans un premier temps puis la sélection du meilleur plan parmi N. Un plan sera évalué sur les critères sociaux énumérés précédemment. L'évaluation des plans produit par HATP passe par trois étapes :
- Déterminer la priorité globale de chaque critère.
- Comparer les plans pour chaque critère.
- Sélectionner le meilleur plan.

Pour calculer la priorité globale pour chaque critère, nous utilisons les priorités descriptives déclarées dans toutes les règles sociales. Ces priorités descriptives sont déterminées par l'utilisateur et permettent de déterminer le degré d'importance du critère par rapport à un critère de référence. Les critères de références sont le coût et la durée du plan, étant donné que ces deux critères sont des critères invariants. En effet, même s'il n'y a pas de règles sociales définies, les plans produits seront tout de même évalués sur ces deux critères. Les priorités descriptives sont des entiers relatifs bornés dans l'intervalle $[-8, 8]$. Une valeur de -8 indique que le critère courant est extrêmement moins important que nos critères de références, une valeur de 0 indique que les deux critères ont un degré d'importance équivalent, et c'est le cas pour nos deux critères coût et durée d'un plan. À partir de là, on peut calculer les priorités locales de chaque critère, et construire la matrice de comparaison entre critères en utilisant les formules 3.5 et 3.1. La formule 3.5 décrit de manière générale comment se fait le calcul de la priorité locale C_{i1}, qui compare un certain critère i qui a une priorité descriptive pr_i au critère de référence repéré par l'indice 1. La formule 3.1 qui a été définie dans la section précédente

décrit les propriétés de la matrice de comparaison, et c'est ce qui nous permet de construire toutes les paires C_{ij}. Une fois la matrice de comparaison construite, il ne reste plus qu'à calculer le premier vecteur propre de la matrice. Chaque élément ρ_i du vecteur va représenter la priorité globale pour le critère correspondant.

$$C_{i1} = \begin{cases} 1 + pr_i \text{ , si } pr_i \geq 0 \\ \frac{1}{1-pr_i} \text{ , sinon} \end{cases} \tag{3.5}$$

La seconde phase, qui est la phase de comparaison et d'évaluation, va se faire en ligne, c'est-à-dire durant le processus de planification. Pour évaluer deux plans et pouvoir affirmer que l'un des deux et meilleur, nous allons calculer le degré d'amélioration du plan de l'un par rapport à l'autre. Ce taux d'amélioration est calculé entre un plan complet qu'on appelle plan de référence [1] et un autre plan qui peut être partiel ou complet. Le taux d'amélioration est calculé sur la base d'une moyenne pondérée (les coefficients sont spécifiés par les priorités globales des critères), sur l'écart moyen qu'il y a entre les deux plans par rapport à chaque critère, comme le montre la formule 3.6. Si le résultat est négatif on va dire qu'il y a détérioration de la qualité du plan, sinon il y a amélioration ou maintient de la qualité sociale du plan.

$$S(P_{new}) = \frac{\Sigma_{i=0}^{N} w_i * \frac{Val_i^{ref} - Val_i^{new}}{Val_i^{ref}}}{\Sigma_{i=0}^{N} w_i = 1} \tag{3.6}$$

En fin de planification, on va se retrouver avec une multitude de plans qui ont maintenu la qualité du plan au même niveau que celle du plan de référence. Comment choisir le meilleur parmi tous ces plans, qui ont la même qualité sociale ? Pour y répondre nous allons continuer la procédure AHP. Pour un ensemble de plans, nous allons calculer leurs priorités locales et globales. Pour la priorité locale, on va comparer les plans entre eux (comparaisons par paire) par rapport à chaque critère, en privilégiant les plans de moindre coût. Cela va produire les matrices de comparaison et leurs vecteurs propres. Supposons que nous avons m plan à comparer et que nous avons n critères, alors cela produira n matrice et vecteur propre qui auront les dimensions respectives de $m \times m$ et m. Chaque élément γ_{ij} va représenter la priorité locale du plan où i et j désignent le critère et le plan concernés. Concernant la priorité globale, on va appliquer la formule $Q(P_j) = \sum_{i=0}^{n} w_i \times \gamma_{ij}$, qui pour chaque plan j va faire la somme pondérée des priorités globales de chaque critère wi avec la priorité locale de chaque plan γ_{ij}. Le meilleur plan est celui qui maximise la priorité globale.

Algorithme 3.1 : Algorithme HATP

Données : s, Arb, D
T ← *leaves*(Arb)
ChoiceReminder = ∅
Prj = ∅
répéter
  T_0 ← {t ∈ T | *predecesseurs*(t) = ∅, *s* satisfait *goal*(t)}
  pour ∀ t ∈ T_0 **faire**
    Retirer t de Arb et de T
  si T = ∅ **alors**
    P ← produire_plan(Arb, Prj)
    Ajouter P dans la pile des plans valides
    Backtracker vers une nouvelle branche
  si $P_{REF} \neq \emptyset$ **alors**
    c ← comparaison_partielle(Arb, Prj, P_{REF})
    si *c < 0* **alors**
      Couper la branche courante de l'arbre d'exploration
      Backtracker vers une nouvelle branche
  si ∀ t ∈ T, *lock(t) actif* **alors** Déverrouiller t
  T_0 ← {t ∈ T | *predecesseurs*(t) = ∅, *s* satisfait *precond*(t), *lock*(t) inactif}
  ChoiceReminder ← AjouterNouveauPoint(T_i,Arb_i,*s*)
  Choisire_branche_Mini_cout(T,Arb,*s*)
  si t *est une action* **alors**
    A ← { a est une action instanciée dans D, *s* satisfait *precond*(a)}
    Modifier *s* selon $effects$(a)
    Appliquer a à Prj
    Mettre à jour le coût
    pour ∀ t ∈ T **faire** Déverrouiller t
  sinon
    M ← { m est une instance d'une méthode dans D, *s* ne satisfait pas *goal*(m)}
    Dec ← {d_i ∈ decomposition(m), *s* satisfait *preconds*(m)}
    si *Dec* =∅ **alors**
      Couper la branche et Backtracker vers une nouvelle branche
    pour chaque d_i ∈ *Dec* **faire**
      T _copy ← T
      Arb _copy ← Arb
      Remplacer t par d_i dans T _copy et dans Arb _copy
      ChoiceReminder ← AjouterNouveauPoint(T _copy,Arb _copy,*s*)
    ChoiceReminder ← Choisire_branche_Mini_cout(T,Arb,*s*)
jusqu'à *limite de temps atteinte* ou *arbre d'exploration entièrement parcouru*

3.5 Algorithme de planification

L'algorithme principal de HATP prend en entrée `s` l'état du monde, une structure hiérarchisée `Arb` qui est le problème et `D` le domaine de planification. Cet algorithme commence par l'initialisation d'une liste de tâches `T` (ligne 1), cette liste contient les feuilles de l'arbre `Arb`. On initialise également l'ébauche de projection temporelle `Prj` et l'ensemble des points de backtrack à un ensemble vide (ligne 3 et ligne 2).

On entre ensuite dans la boucle principale (lignes 4 à 38) jusqu'à ce que la limite de temps éventuelle soit atteinte, ce qui permet de satisfaire la contrainte de la planification en temps réel, ou bien jusqu'à ce que l'ensemble de l'arbre d'exploration ait été entièrement parcouru. La première étape de la boucle principale est d'éradiquer les tâches associées à des buts qui sont déjà satisfaits dans l'état actuel de la base de faits (lignes 5 à 7), afin de limiter l'explosion de l'espace de recherche.

L'étape suivante consiste à vérifier si la liste de tâches `T` est vide, ce qui indiquerait qu'un plan valide a été trouvé (lignes 8 à 11). La troisième étape consiste à appliquer une optimisation de type *branch-and-bound* (lignes 12 à 16), en comparant le plan actuel au plan de référence afin de limiter l'espace de recherche à explorer. L'évaluation partielle ne porte que sur le sous-ensemble des règles sociales qui peuvent être évaluées pendant la phase de raffinement : la *gestion des efforts*, les *séquences indésirables*, les *liens croisés* et les *mauvaises décompositions*. Les règles sociales des types *états indésirables* et *temps d'inactivité* sont évaluées lorsque le plan complet a été trouvé et que son treillis temporel numérique a été construit.

L'algorithme a une procédure de verrouillage/déverrouillage de tâches qui lui permet de ne pas explorer plusieurs fois des branches jumelles. Initialement, toutes les tâches sont verrouillées. Quand une tâche t est sélectionnée, toutes les autres tâches sont verrouillées, c'est-à-dire qu'elles ne peuvent être sélectionnées tant que t ou une de ses sous-tâches ne soit réduite à un ensemble vide ou ait mené à une modification de l'état du monde (i.e. à l'application d'une action).

Dans le cas où la liste de tâches `T` contient des tâches qui sont toutes verrouillées, on déverrouille la totalité de `T` (ligne 17). On détermine ensuite l'ensemble T_0 contenant l'ensemble des tâches de `T` qui ne sont pas verrouillées, qui n'ont pas de prédécesseurs et dont les préconditions sont satisfaites (ligne 18). On crée autant de branches que nécessaire dans l'arbre de recherche selon la cardinalité de T_0. On choisit ensuite une branche qui minimise le coût du plan en choisissant une tâche `t` à traiter (ligne 20).

Dans le cas où la tâche sélectionnée est une action, on applique les effets de cette action sur la base de faits (ligne 23), on met à jour la projection `Prj` et le coût du plan (ligne 24 à 25). Enfin on déverrouille toutes les tâches dans `T`.

1. Un plan de référence est un plan qui a la meilleure évaluation, durant le processus de planification. Le premier plan trouvé par HATP est considéré comme le plan de référence. Si le planificateur trouve un meilleur plan il va le substituer au plan de référence.

Si la tâche `t` est une méthode, on détermine l'ensemble des décompositions valides (ligne 29), si l'ensemble résultant `Dec` est vide, on change de branche d'exploration (lignes 30). Sinon, pour chaque d_i présent dans *Dec* on crée une copie de la liste de tâches `T` et de l'arbre `Arb` et on remplace la tâche `t` par d_i (de la ligne 33 jusqu'à ligne 35). On rajoute ensuite une nouvelle branche à l'arbre de recherche (ligne 36).

L'algorithme de HATP est fondé sur celui de SHOP2. Cependant, il présente quelques différences : (1) la définition du plan. (2) la forme des problèmes de planification. (3) l'optimisation. Les deux premiers points ont été abordés précédemment. Concernant l'optimisation, comme nous avons pu le constater dans la présentation de l'algorithme, HATP utilise une heuristique de recherche basée sur un A* [Hart 68] et une optimisation du type *branch-and-bound* qui reposent toutes les deux sur l'évaluation partielle des plans. L'évaluation est fondée sur une méthode d'analyse multicritères (AHP) qui est beaucoup plus évoluée que l'approche utilisée dans SHOP2 qui est une approche additive.

3.6 Conclusions et critiques du modèle

Dans ce chapitre, nous avons décrit le formalisme du planificateur HATP qui a été spécialement conçu pour générer des plans qui prennent explicitement en compte l'humain, en considérant ses désirs, ses compétences, ses handicaps, etc.. En intégrant dans son modèle des règles sociales et une métrique qui évalue la qualité sociale des plans, il est capable de produire des plans qui sont compréhensibles, agréables, et qui respectent les conventions de comportements des humains.

Nous avons donné la description du domaine de planification et des règles sociales. Nous avons également détaillé la métrique utilisée pour la production de plans socialement acceptables. Nous avons décrit l'algorithme de planification de HATP qui intègre une heuristique de guidage et une procédure de *branch-and-bound* qui lui permet de réduire l'espace de recherche pour améliorer les performances de celui-ci.

Tout au long du chapitre, nous avons fait une comparaison entre HATP et le planificateur SHOP2. Nous avons pu constater que HATP intègre une description particulière des états du monde, qui permet de construire un modèle de chacune des entités présentes dans le monde, et de regrouper toutes les informations les concernant dans une même variable pour les rendre plus facilement accessibles. Le modèle des entités est complété avec la description des tâches présentes dans le domaine. En effet, le modèle de tâches dans HATP prend comme paramètres des variables typées, ce qui permet de poser des contraintes sur les entités. Nous avons également constaté que HATP permet une planification à partir de plan partiel, ce qui améliore l'interaction homme-robot en offrant aux partenaires humains la possibilité d'exprimer leurs désirs sur la façon de réaliser un plan. Pour la description de la structure de données d'un plan, HATP se distingue par le fait qu'il décrit le plan avec deux structures : Une projection temporelle comme celle qui est présente dans SHOP2, et également une structure hiérarchisée qui décrit la décomposition

du problème et qui permet d'améliorer la qualité de l'interaction homme-robot. Au niveau de l'algorithme de planification, nous avons pu noter que l'algorithme de planification de HATP est fondé sur celui de SHOP2, mais qu'il se différencie au niveau de l'évaluation de la qualité d'un plan. En effet, la procédure d'évaluation de la qualité d'un plan dans HATP et beaucoup plus évoluée que celle utilisée dans SHOP2, et de ce fait, elle implique des calculs plus complexes lors des évaluations partielles des plans pour *branch-and-bound* ou bien lors de l'évaluation des branches pour l'heuristique.

Le modèle de HATP tel-quel est adapté à l'interaction avec les humains, mais présente les insuffisances suivantes :

1. Lors de la planification HATP : ne prend en compte que les croyances du robot sans les confronter aux croyances des autres agents. Cela a pour effet de produire des plans qui peuvent être incompréhensibles pour l'humain dans le cas où ils ont des croyances divergentes. Les plans produits mobilisant l'humain et le robot sur une longue durée peuvent susciter un sentiment de dépendance et d'emprisonnement dans la tâche. La règle *liens croisés* permet de limiter le nombre des liens croisés entre agents, mais ne permet pas de tous les supprimer. Il est donc nécessaire d'introduire des actions de communication qui vont permettre de relâcher les contraintes de précédence entre agents. Nous avons la conviction que la prise en compte des croyances des autres agents ainsi que l'introduction des actions de communication, vont enrichir l'interaction et donner a celle-ci plus de souplesse et la render plus agréable pour les humains partenaires du robot.

2. Dans notre approche, l'interaction homme-robot n'est considérée que du point de vue du raisonnement symbolique, mais il existe un autre type d'interaction et qui est très important, c'est l'interaction au niveau physique. Cet autre type d'interaction permet de contrôler le robot au niveau fonctionnel, en contrôlant ses déplacements et ses choix de positions en présence de l'homme. Il existe des planificateurs qui traitent cette problématique, qu'on appelle planificateur de mouvement ou planificateur de manipulation, ou plus généralement planificateur géométrique. L'approche produisant des plans qui gèrent les deux niveaux d'interaction n'est pas nouvelle. On l'a déjà testée et validée dans le cadre du projet *COGNIRON*, mais cette approche est une approche dite en série, où l'on produit un plan symbolique à la suite duquel on produit et réalise un plan géométrique. Nous pensons qu'une approche dite parallèle, c'est-à-dire avec production des deux plans symboliques et géométriques en parallèle avec une forte interaction entre eux, va nous permettre de traiter le problème de l'interaction homme robot beaucoup plus finement, du fait que les deux planificateurs vont enrichir leur représentation du monde. Chaque planificateur pourra profiter des primitives de l'autre, mais cela permettra surtout au planificateur symbolique d'élargir son champs de compétence en faisant du raisonnement plus élaboré sur des faits géométriques. Le planificateur géométrique pourra élargir et adapter le panel de comportement qu'il pourra donner au robot grâce aux recommandations du planificateur symbolique.

Dans les chapitres suivants nous allons présenter des extensions du modèle HATP qui traite de ces insuffisances

4

Belief-HATP : planification avec croyances mutuelles

Dans le chapitre précédent, nous avons décrit le planificateur HATP. Nous avons pu observer ses capacités à produire des plans socialement acceptables pour le robot et pour les humains partenaires. Cependant, nous avons également noté que ce modèle présente quelques insuffisances. L'une de ces insuffisances est que le modèle prend en compte uniquement les croyances du robot pour planifier et nous avons indiqué que cela peut susciter des sentiments d'incompréhension chez son partenaire humain.

Dans ce chapitre, nous présentons une extension du modèle HATP qui prend en charge les croyances des différents agents et utilise la structure HTN du planificateur pour la gestion de ces croyances. Elle permet également au robot de produire des comportements pro-actifs pour l'acquisition ou le transfert d'information. Nous allons également présenter quelques exemples montrant l'intérêt de la prise en compte des croyances des différents agents durant la phase de planification.

4.1 Travaux voisins

La planification avec croyances mutuelles a été utilisée dans plusieurs approches. Dans le chapitre 2 nous avons abordé l'approche par planification réactive, et plus précisément le modèle *STEAM* [Tambe 97] qui utilise la théorie de l'intention jointe afin de construire des plans pour la résolution de problème multi-agents. Nous avons pu constater que ce modèle utilise une structure hiérarchisée appelée TOP (Team Oriented Plan) pour l'élaboration de ses plans. Nous avons également établi que le modèle fait une distinction entre les tâches jointes appelées "opérateurs d'équipe" et les tâches individuelles appelées "opérateurs individuels". Cette distinction va servir au moment d'instancier le plan. En effet, pour instancier un opérateur et vérifier ses préconditions, il est nécessaire de connaître son type. Si c'est un opérateur d'équipe, cela implique qu'il doit être décomposé en sous-tâches ; si c'est un opérateur individuel, cela implique que l'agent concerné va utiliser ses croyances pour réaliser la tâche. Quand tous les opérateurs individuels d'un opérateur de groupe sont réalisés, les agents impliqués reconstruisent une croyance mutuelle consistante pour conclure que l'opérateur de groupe est réalisé.

Dans cette approche, trois éléments importants sont à noter :
– Chaque agent a une croyance individuelle. La croyance mutuelle est l'union des croyances des différents agents.
– Utilisation des croyances individuelles pour la réalisation des tâches individuelles.
– Utilisation d'une structure hiérarchique pour la construction du plan.

Une autre approche est l'approche dite planification continue [Brenner 09], [Clement 03]. Elle repose sur l'idée du (*active knowledge-gathering* [Knoblock 95]), où l'agent ne planifie pas seulement pour atteindre un but, mais également pour acquérir les informations nécessaires à la réalisation de ce but. Ce type de planification utilise l'entrelacement entre planification et exécution [Ambros-Ingerson 90], ce qui lui permet de palier au manque d'information en passant d'une phase à une autre jusqu'à l'obtention d'un plan complet et correct.

Ce type de planification utilise un langage appelé ***MAPL*** pour (Multi-Agent Planning Language) [Brenner 03], qui est dérivé du langage PDDL. Ce langage décrit le domaine de planification qui inclut des actions physiques, des actions d'observations et des actions de communications. Sa représentation du monde inclut les croyances propres et les croyances mutuelles des agents.

La représentation du monde s'appuie sur les variables d'états multi-valeurs [Bäckström 93]. Dans ce modèle, chaque agent a sa propre base de croyances, et pour avoir une croyance mutuelle, il faut qu'il y ait une correspondance entre toutes les croyances individuelles de tous les agents. Le modèle a été également étendu à la notion de *unknown* qui permet de spécifier si un agent connaît ou non l'instance d'une certaine variable.

Dans cette approche, on a enrichi le modèle d'action par la notion (*Know if*), qui est une condition de re-planification qui permet de détecter la connaissance ou la non-connaissance d'une certaine information, permettant ainsi à un agent de raisonner sur la réalisation d'une action. La figure 4.1 illustre un exemple de déclaration d'une action dans le langage ***MAPL***. Dans cette action, l'agent a besoin de connaître si la porte est ouverte ou non, information qui est représentée par " *:replan (KIF ?a (doorstate ?d))*". Il faut savoir que cette condition n'intervient pas directement lors de la planification. Elle est utilisée lors de l'exécution pour permettre à l'acteur de l'action de re-planifier si la condition n'est pas respectée. Dans notre exemple si la porte n'est pas ouverte ou si l'agent ne connaît pas l'état de la porte, il peut alors re-planifier pour ouvrir la porte ou bien observer l'état de cette dernière.

```
1 (    :action move_A
2      :agent (?a - agent)
3      :parameters (?to - location)
4      :variables (?from - location ?d - door)
5      :replan (KIF ?a (doorstate ?d))
6      :precondition (and (pos ?a : ?from)
7          (entrance ?d ?from) (entrance ?d ?to))
8      :effect (pos ?a : ?to)
9 )
```

FIGURE 4.1 – Exemple de déclaration d'une action avec l'approche planification continue

L'algorithme pour la planification continue se décompose en trois phases :

1. (Re-)planification pour atteindre un état but à partir d'un état initial. Cette phase utilise un algorithme classique de recherche en avant.
2. Exécution des actions produites par la phase précédente.
3. Á l'exécution, l'algorithme compare l'état du monde courant avec l'état du monde estimé en se servant des observations que l'agent peut faire.

Cet algorithme a été étendu au cas du multi-agent en utilisant un algorithme distribué qui permet la communication entre agents. L'idée est la suivante : durant la planification d'un agent, s'il rencontre un état bloquant, il envoie aux autres agents un message sous forme d'un but pour débloquer la situation et cela jusqu'à construire un plan complet.

Cette approche a été testée sur un scénario dans le cadre d'un dialogue entre agents [Brenner 08] [Kruijff 07]. Dans le scénario on suppose que nous avons deux agents *MrChips* et *MrData*. *MrChips* peut se déplacer et prendre des objets, *MrData* peut aussi se déplacer et ouvrir les portes. Le but de *MrData* est d'avoir un café et le but de *MrChips* est de satisfaire *MrData*. La figure 4.2 illustre le plan produit et exécuté par les deux agents.

A l'exécution, *MrChips* ne connaît pas la position du café alors il fait intervenir *MrData* pour obtenir l'information (ligne 3). Puis, *MrChips* se dirige vers la porte et observe qu'elle est fermée. De nouveau il fait intervenir *MrData* pour lui ouvrir la porte (lignes 5 à 7). *MrChips* prépare le café et prévient *MrData* que le café est prêt et qu'il peut le prendre (ligne 8 à 13).

MAPSIM run starts. There are 2 agents: MrChips and MrData.
(1) MrChips: "What can I do for you, MrData?"
(2) MrData: "Please bring me the coffee, MrChips!"
(3) MrChips: "Where is the coffee, MrData?"
(4) MrData: "The coffee is in the kitchen, MrChips!"
(5) MrChips: "Please open the kitchen door, MrData!"
(6) MrData opens the kitchen door.
(7) MrChips moves to the kitchen.
(8) MrChips takes the coffee.
(9) MrChips moves to the livingroom.
(10) MrChips: "I have the coffee, MrData!"
(11) MrChips: "Please take the coffee, MrData!"
(12) MrData takes the coffee.
(13) MrData: "Thanks for the coffee, MrChips!"
MAPSIM terminates successfully.

FIGURE 4.2 – Exemple de dialogue entre agents en utilisant la planification continue

Dans l'approche de la planification continue, nous devons noter quatre points importants :

– Chacun des agents a sa croyance individuelle. La croyance mutuelle est une fusion entre les croyances individuelles de chaque agent.

– La notion de variable d'état à valeur multiple, qui permet de créer les croyances individuelles des agents.

– La notion *unknown* qui permet de modéliser le fait qu'un agent ignore une information.

– La condition de re-planification (*Know if*) qui permet de générer un but de recherche d'information.

– Le modèle des actions de communication et la notion de coprésence qui permettent le passage des informations entre agents.

Pour la conception d'un planificateur avec croyance mutuelle et d'après ce que nous avons pu noter des différentes approches, le planificateur doit intégrer certaines propriétés qui peuvent améliorer l'interaction homme-robot.

– Il doit intégrer le fait que chaque agent doit avoir ses croyances individuelles. Cela permettra au planificateur de ne plus raisonner que sur les croyances du robot, mais également de prendre en compte les croyances des humains partenaires pour pouvoir produire des plans compréhensibles par ces derniers, même en cas de croyances individuelles divergentes.

– Il doit compléter son domaine de planification par des actions de communication, qui vont permettre la transmission d'information en cas d'absence d'information ou divergence de croyances entre le robot et les humains. Cela permettra également de rendre plus clair les plans produits par le robot.

– Il doit intégrer le fait qu'un agent connaisse ou non une information, ce qui permettra au robot de générer des buts pour la recherche d'information, et de connaître les raisons de son échec, mais également de pouvoir renseigner l'humain quand ce dernier en a besoin.

Dans la suite du document nous présentons le modèle de planification avec gestion des croyances multiples et règles sociales proposé et mis en œuvre pour l'interaction homme-robot.

4.2 Description domaine de planification

Dans cette section nous présentons un modèle de planification avec information incomplète qui s'inspire de la planification continue présentée dans la section précédente.

```
factdatabase
{
    // Étape 1 : Définition des types des entités
    define entityType Place;
    define entityType Object;

    // Étape 2 : Définition des attributs des agents
    define AgentAttributes
    {
        static  atom    symbol  type;
        dynamic atom    Place   posTopo;
        dynamic set     Object  objects;
    }
    // Étape 2' : définition des entités
    define entityAttributes Objet
    {
        static  atom    number  poids;
        dynamic atom    Agent   owner;
    }



    // Étape 3 : Création des entités

    livingRoom = new Place;
    Jido = new Agent;
    Tom = new Agent;
    Mathieu = new Agent;
    Glass = new Objet;
    Bottle = new Objet;

    // Étape 4 : Initialisation des attributs
    Jido.type = "Robot";
    Jido.posTopo = livingRoom;
    Jido.objects <<= Glass;
    Jido.objects <<= Bottle;
    Jido.Myself = true;

    Glass(Jido).poids = 100;
    Glass(Jido).owner = Tom;

    Glass(Tom).poids = 50;
    Glass(Tom).owner = Mathieu;


}
```

FIGURE 4.3 – Exemple de déclaration de la base de faits

4.2.1 Représentation du monde

Dans le chapitre précédent, nous avons fait la description de la base de faits pour le planificateur HATP. Nous avons constaté qu'elle reposait sur une représentation en variables d'état sous forme d'entités et d'attributs. Cette description suppose que tous les états du monde sont connus et ne modélise que la connaissance du robot sur ces états. Elle suppose également que tous les agents partagent la même vision. Cette description unifiée pour tous les agents n'est pas adaptée à l'interaction homme-robot. En effet, un robot qui produit des plans pour lui et pour des humains partenaires et qui ne prend pas en compte la différence entre leurs connaissances, va produire des plans qui sont incompréhensibles, suscitant le sentiment de perte de temps et d'agacement chez l'humain. De plus, le robot peut être vu comme inutile. Prenons l'exemple où le robot et l'humain doivent réaliser la tâche de "nettoyer la chambre". La réalisation de la tâche va se décomposer en "aller chercher les outils pour l'humain et pour le robot puis nettoyer le sol". Nous supposons que la tâche d'aller chercher les outils est à réaliser par l'humain et que celui-ci ne connaît pas la position des outils. Il va commencer par les chercher. Le robot, en observant l'humain, peut en déduire que celui-ci a abandonné la tâche, ce qui peut impliquer que lui aussi doit l'abandonner et donc, susciter un sentiment d'incompréhension chez l'humain.

Pour remédier à cette insuffisance, nous allons étendre cette représentation pour prendre en compte une information incomplète dans le sens ou un agent peut connaître ou non une information, mais également les connaissances propres à chaque agent dans le processus de planification. Dans le reste du chapitre, nous allons utiliser le terme "*croyance*". Dans le cas du robot, cela va désigner ses connaissances sur l'état du monde. Pour tous les autres agents, cela désigne les croyances qu'a le robot sur les connaissances des autres agents.

Précédemment, nous avons constaté que toutes les variables ont le même type, qui est le type "*entité*". Dans cette extension du modèle, nous avons fait la distinction entre *Agent* et *entité* afin de pouvoir créer pour chaque agent sa propre base de croyance, indépendamment de celles des autres.

Un agent est composé d'un ensemble d'attributs qui décrivent son état interne. La figure 4.3 illustre un exemple de déclaration d'une base de faits. L'étape deux représente la phase où les attributs qui vont être associés aux agents sont déclarés. Comme on peut le voir, chaque agent va avoir une position topologique représentée par la variable ***posTopo***, et un type représenté par la variable ***type*** qui va servir à différencier les agents *humain* des agents *robot* ou des agents d'autre type.

On a rajouté dans le modèle des agents un attribut booléen particulier **Myself**, qui est créé automatiquement par le système [1]. **Myself** représente l'agent sur lequel tourne le planificateur. C'est à l'utilisateur de définir lequel des agents est **Myself**.

1. Quand le système crée l'attribut booléen **Myself**, il le met par défaut à la valeur "false"

Les Croyances

Il est important de préciser ici que nous ne faisons pas de gestion des croyances imbriquées. Par exemple " l'agent **A** croit que l'agent **B** croit que l'agent **C** crois ...". Mais on ne gère que les croyances propres de l'agent **Myself**, et les croyances qu'il peut avoir sur les connaissances des autres agents, qu'on va appeler croyances des autres agents.

Pour la représentation des croyances de chaque agent, nous avons repris le même modèle que celui défini dans le chapitre précédent pour la représentation de la base de faits et nous l'avons étendu avec la notion de variables d'état à valeurs multiples (MVSV). Une variable d'état v est instanciée à partir d'un domaine fini Dom (l'ensemble des valeurs qu'elle peut prendre). Une variable d'état à valeurs multiples V est également instanciée à partir d'un domaine Dom, et pour chaque agent $a \in A$ la variable V prend une certaine instance $V(a) = v \in Dom$. Le modèle tel quel est incomplet. En effet, on ne peut pas exprimer le fait qu'un agent ignore une information. De plus, dans le cas ou un agent différent de **MySelf** connaît l'instance d'un certain attribut, le modèle ne permet pas d'exprimer le fait que l'agent **Myself** ignore cette instance, mais il sait que l'autre agent la connaît. Pour combler ce manque, nous avons introduit deux notions qui sont *unknown* et *known*. La première modélise le fait que les agents ne connaissent pas une information. La seconde modélise le fait que l'agent **MySelf** ne connaît pas une information et qu'un autre agent la connaît. Ainsi, le domaine des instances possibles d'une variable peut être décrit par :

$$V(a) \in \begin{cases} Dom_v \sqcup \{unknown\} \sqcup \{known\} \text{si} & a \neq \text{MySelf} \\ Dom_v \sqcup \{unknown\} & \text{sinon} \end{cases} \tag{4.1}$$

Cette nouvelle représentation n'enlève rien à la capacité de HATP à construire un modèle explicite des agents. Au contraire, elle vient en complément. En effet, dans le chapitre précédent, nous avons constaté que la modélisation des agents sous forme d'entités permet de regrouper toutes les données les concernant dans la même variable. Nous avons également expliqué que ce modèle est complété par la définition des actions avec leurs paramètres typés et leur modèle de coût. Cette extension vient renforcer cette description en ajoutant au modèle des agents la représentation de leurs croyances. Dans le cadre de l'interaction homme-robot, cette extension vient servir le robot dans le sens où il peut désormais prendre en considération les croyances de l'humain lors de la construction du plan. Cela lui permet d'analyser et de réagir au manque d'information que l'humain peut avoir et, de ce fait, générer des comportements proactifs.

L'exemple de la figure 4.3 illustre la déclaration et l'initialisation d'une base de croyances pour les différents agents. L'entité "*GLASS*" est de type *object*. Ses attributs sont : son "*poids*" et son possesseur "*owner*". Si nous considérons l'attribut "*poids*", on peut constater qu'il a deux instances. La première par rapport à l'agent "*Jido*" et l'autre par rapport à l'agent "*Tom*". Nous pouvons constater que ces deux instances sont différentes, ce qui nous permet de dire que les agents "*Jido*" et "*Tom*" ont des croyances différentes sur le poids de l'objet "*GLASS*". Pour

l'attribut "*owner*", nous pouvons constater que dans les croyances de l'agent "*jido*", le possesseur de l'objet "*GLASS*" est l'agent "*Tom*", et que dans les croyances de l'agent *Tom* le possesseur est un autre agent appelé "*Mathieu*".

Mis à jour des croyances

Un changement sur les croyances d'un agent ce fait grâce aux actions qu'il réalise, soit sur des observations qu'il peut faire, soit en communiquant avec les autres agents. Cependant, Les croyances de **MySelf** évoluent également grâce à la communication avec les autres agents, mais aussi grâce aux actions qui sont réalisées. Ce qui signifie que toutes les actions réalisées par des agents affectent obligatoirement leurs croyances, mais également les croyances de **MySelf**.

Définition d'une croyance consistante

Nous définissons une croyance consistante par $\| \bigcup_{\forall a \in Ag} V(a) \| = 1$, cela signifie que pour avoir une croyance consistante sur une variable d'état V, il faut que l'union des croyances des agents forme un ensemble de dimension 1. Dans tout autre cas il y a inconsistance. Dans notre cas et du fait que nous planifions pour **Myself**, le test de la consistance des croyances se fait par paire dans le sens où nous comparons les croyances des autres agents à celle de **Myself**.

Extension de la syntaxe de HATP

La syntaxe de HATP a été décrite dans le chapitre précédent. Nous avions adopté la notation *Entity.Attribut* pour accéder à la valeur de l'attribut. Dans le cas des croyances multiples, cette syntaxe n'est plus adaptée. En effet, nous sommes dans un cas où chaque agent a sa propre base de croyances. Il est alors nécessaire d'indiquer le nom de l'agent concerné par la valeur d'un attribut.

Nous avons étendu la syntaxe pour pouvoir prendre en charge le cas des croyances multiples. Nous avons gardé les mêmes opérateurs et les mêmes fonctions que précédemment, le seul changement concerne l'accès aux données d'un attribut. Nous avons adopté la notation $Entity(Agent).Attribut$ qui renvoie la valeur de l'attribut pour un agent donné.

Nous avons également adopté les simplifications suivantes :

– $Entity(MySelf).Attibut$ qui permet d'accéder à la base de croyances de **MySelf**, ce qui permet de décrire un domaine de planification prenant en compte l'agent pour lequel on va planifier.

– $Entity.Attibut$ cette notation est équivalente à $Entity(MySelf).Attibut$. Nous avons adopté cette réduction pour ne pas créer de rupture avec la définition classique de HATP.

C'est-à-dire que lorsque nous utilisons uniquement cette notation pour la définition du domaine de planification cela revient à planifier uniquement avec les croyances de **MySelf**, ce qui correspond à utiliser HATP sans croyances multiples.

4.2.2 Extension de la définition des actions

Pour la description des actions, nous avons gardé sensiblement la même structure, c'est-à-dire la description par le tuple $A =< Name, Ag, Par, Precond, Eff, Co, Du >$ comme dans le chapitre précédent. Les différences majeures sont les suivantes :

1. Ag représente un ensemble d'agents qui sont impliqués dans la réalisation de l'action. Du fait de la séparation entre les définitions des agents et des entités, la liste des paramètres d'une action est divisée en deux : La liste Ag des agents et la liste Par des entités impliquées.
2. $Precond$ représente les préconditions de l'action comme elles ont été définies dans la section 3.3.1. Les préconditions vont être testées par rapport aux croyances des agents impliqués dans la réalisation de l'action.
3. Eff représente les effets de l'action sur les croyances des agents qui sont impliqués dans la réalisation de celle-ci. Cette mise à jour ne peut être complète que si nous prenons en considération la notion de coprésence [Lewis 02]. Nous identifions que des agents sont en coprésence s'ils peuvent s'observer mutuellement, mais également s'ils peuvent observer les mêmes objets. La notion de coprésence a pour effet de modifier l'état de croyance des agents qui ne sont pas impliqués par l'action.

   Pour mettre à jour les états de croyance de ces agents, nous utilisons la fonction Forall (définie précédemment section 3.3.1). Cette fonction permet de sélectionner un ensemble d'agents qui respectent certaines conditions et de leur appliquer un certain nombre d'effets.

La figure 4.4 décrit la déclaration de l'action **PutdownOn(Agent Ag, Object Obj, Furniture F)**. En examinant cette déclaration, nous pouvons identifier que l'acteur de l'action est l'agent **Ag** et que le but est qu'il pose l'objet **Obj** sur le meuble **F**. La précondition de l'action va être inspectée par rapport aux croyances de l'agent **Ag**. Dans la base de croyance de **Ag**, il faut qu'il soit à la même position que le meuble ($Ag.posTopo == F(Ag).posTopo$). Il faut que le meuble soit capable de supporter des objets ($F(Ag).onTop == true$) et que l'agent soit en possession de l'objet ($Obj(Ag) >> Ag.objects \ \wedge \ Obj(Ag).owner == Ag$).

Les effets de cette action vont affecter les croyances de l'agent **Ag**, avec les changements suivants : l'objet **Obj** est sur la table ($F(Ag).objectsOn <<= Obj$), il n'est plus dans la liste des possessions de l'agent **Ag**, et le possesseur de l'objet n'est plus l'agent **Ag** ($Obj(Ag).owner == NULL$).

La réalisation de cette action va également avoir des effets sur tous les agents qui respectent une de ces deux conditions :

```
action PutdownOn(Agent Ag, Object Obj, Furniture F)
{
// Déclaration des préconditions
  preconditions
  {
    Ag.posTopo == F(Ag).posTopo;
    F(Ag).onTop == true;
    Obj(Ag) >> Ag.objects;
    Obj(Ag).owner == Ag;
  };
  // Définition des effets
  effects
  {
    Ag.objects =>> Obj;
    F(Ag).objectsOn <<= Obj;
    Ag.full = "false";
    Obj(Ag).owner = NULL;
    Obj(Ag).furniture = F;

    Forall(A)=(Agent; {OR
                        {Ag>>A.visibleAgent;}
                        {F>>A.visibleFurnitur;}
                      };)
    {
    F(A).objectsOn <<= Obj;
    Obj(A).owner = NULL;
    Obj(A).furniture = F;}
  };

  cost{PutdownOnFonctionCost(Ag,Obj,F)};
  duration { PutdownOnFonctionDuration(Ag,Obj,F); };
}
```

FIGURE 4.4 – Exemple de description d'une action dans HATP avec croyances multiples

1. $Ag >> A.visibleAgent$ qui signifie que l'agent **A** peut observer l'action de l'agent **Ag**.
2. $F >> A.visibleFurnitur$ qui signifie que l'agent **A** peut observer les actions réalisées sur le meuble **F**.

Tout agent respectant une de ces deux conditions verra ses croyances modifiées comme suit :
– La nouvelle position de l'objet sera sur le meuble ($Obj(Ag).furniture = F$).
– L'objet va rentrer dans la liste des objets qui sont sur le meuble ($F(A).objectsOn <<= Obj$).
– Le possesseur de l'objet sera mis à *NULL*.

4.2.3 Action de communication

Nous avons ajouté au domaine de planification des actions de communication qui vont permettre le passage des informations entre deux agents ou plus. Nous ne faisons pas de distinction entre communication verbale ou non-verbale. La seule distinction que nous faisons est entre communications privées et publiques. Les actions de communications sont décrites par le tuple

$$A =< Name, Ag, Par, Precond, Eff, Co, Du >$$

Où :
– $Name$ représente le nom de l'action.
– Ag représente un couple d'agents composé de l'orateur et de l'auditeur.
– Par est un ensemble composé de deux éléments $< En, Att >$, où En représente une entité et Att représente un des attributs de l'entité En. Ce couple entité et attribut précise le sujet de la conversation.
– $Precond$ tel que défini précédemment, représente les préconditions qui doivent être vérifiées avant l'exécution de l'action. Généralement elles vont porter sur la proximité physique entre les agents : il faut que les deux agents soient à une distance permettant la communication.
– Eff représente les effets de l'action sur les états de croyances de l'agent auditeur.
– Co et Du, comme précédemment, représentent respectivement les fonctions coût et durée.

Les figures 4.5 et 4.6 décrivent la déclaration des actions de communications :
– **GiveInformationAbout(Agent Ag1, Agent Ag2, Var, Att)**
– **GetInformationAbout(Agent Ag1, Agent Ag2, Var, Att)**

Le but des deux actions est d'échanger une information entre les agents **Ag1** et **Ag2** concernant l'attribut **Att** de l'entité **Var**.

Pour la première, **GiveInformationAbout**, l'information passe de **Ag1** à **Ag2** comme représenté dans les effets de l'action. Pour la deuxième, c'est le cas contraire. L'information passe de l'agent **Ag2** à l'agent **Ag1**. Ces actions n'ont d'effet que sur les croyances des agents impliqués, c'est-à-dire qu'elles sont des conversations privées, mais l'ajout de la fonction Forall

peut rendre la conversation publique. Dans l'exemple de l'action **GiveInformationAbout**, tout agent proche des deux agents **Ag1** et **Ag2** reçoit la même information que les agents concernés par l'action.

```
action GiveInformationAbout(Agent Ag1, Agent Ag2, Var, Att)
{
// Déclaration des préconditions
  preconditions
  {
    Ag1.posTopo == Ag2.posTopo;
    Var(Ag2).Att != Var(Ag1).Att;
  };
  // Définition des effets
  effects
  {
    Var(Ag2).Att=Var(Ag1).Att;
    Forall(A)=(Agent; {A != Ag1; A != Ag2; Ag1.posTopo>>A.nextPosition;};)
    {
      Var(A).Att=Var(Ag1).Att;
    }
  };

  cost{GiveInformationAboutFonctionCost(Ag1,Ag2)};
  duration { GiveInformationAboutFonctionCost(Ag1, Ag2,Var,Att); };
}
```

FIGURE 4.5 – Exemple : Déclaration d'une action de communication pour donner une information

```
action GetInformationAbout(Agent Ag1, Agent Ag2, Var, Att)
{
// Déclaration des préconditions
  preconditions
  {
    Ag1.posTopo == Ag2.posTopo;
    Var(Ag1).Att != Var(Ag2).Att;
  };
  // Définition des effets
  effects
  {
    Var(Ag1).Att=Var(Ag2).Att;
  };

  cost{GetInformationAboutFonctionCost(Ag1,Ag2)};
  duration { GetInformationAboutFonctionCost(Ag1, Ag2,Var,Att); };
}
```

FIGURE 4.6 – Exemple 2 : Déclaration d'une action de communication pour recevoir une information

4.2.4 Extension de la définition des tâches de haut niveau

Pour la description des méthodes dans ce modèle, nous avons gardé la même description que précédemment. Une méthode est définie par le tuple $M =< Name, Par, goal, Replan, D >$, où :

– *Name* représente un symbole qui définit le nom de la tâche. Dans l'exemple de la figure 4.7 le nom de la méthode est **GetObject**.

– *Par* représente les paramètres de la tâche. C'est une liste non ordonnée d'entités et

```
method GetObject(Agent Ag, Object Obj)
  {

  condition
    {
        { Obj.furniture== unknown;
          Obj.owner== unknown;
        };
     subtasks{AG = SELECT(Agent, { AG.MySelf == true; });
              1:SearchFor(AG, Obj);
              2:PlanAgain(AG)>1;}
    };

    goal
    {
      Obj >> Ag.objects;
      Obj.owner == Ag;
    };


    //decomposition 1
    {
      preconditions
      { Obj.owned == "false"; };

      subtasks
      { 1:GetObjectFromFurniture(Ag, Obj, Obj.furniture); };
    }

    //decomposition 2
    {
      preconditions
      { Obj.owned == "false"; };

      subtasks
      {
        AG = SELECT(Agent, { AG != Ag; });
        1:GetObjectFromFurniture(AG, Obj, Obj.furniture);
        2:TransmitObject(AG, Ag, Obj) > 1;
      };
    }

    //decomposition 3
    {
      preconditions
      {
        Obj.owned == "true";
        Obj.owner != Ag;
      };

      subtasks
      { 1:TransmitObject(Obj.owner, Ag, Obj); };
    }
  }
```

FIGURE 4.7 – Exemple de description d'une méthode

d'agents. Dans l'exemple de la figure 4.7 les paramètres sont : un agent **Ag** et une entité de type objet **Obj**.

– *goal* représente une liste des conditions qui définissent le but ou les méta-effets associés à la méthode. Si ces conditions sont respectées alors le but de la méthode est déjà réalisé. Dans ce cas, la méthode est réduite à un ensemble vide. Le but de la méthode décrit dans la figure 4.7 est que l'agent **Ag** soit en possession de l'objet **Obj**. Cela se traduit par $Obj >> Ag.objects$ et $Obj.owner == Ag$.

– *Replan* représente une liste des conditions qui permettent de tester s'il y a un manque d'information ou s'il y a une inconsistance sur les croyances pour la décomposition et la réalisation de la méthode. Si les conditions sont respectées, la méthode est remplacée par une liste de tâches. Pour l'exemple de la figure 4.7, la condition pour que la méthode **GetObject** soit non réalisable est que la position de l'objet soit inconnue. Cela se traduit par $Obj.furniture == unknown \ \wedge \ Obj.owner == unknown$. Si ces conditions sont valides, la méthode sera remplacée par la liste de tâches correspondante.

– *D* représente l'ensemble des décompositions possibles de la méthode. Ces décompositions ont la même description que celles données dans la section 3.3.2 du chapitre précédent.

Comme dans notre modélisation, chaque agent a sa propre base de croyance, nous avons créé un type particulier de méthode, appelé méthode terminale. Une méthode terminale est une macro-action, c'est-à-dire que la méthode ne fait intervenir qu'un seul agent qui est l'acteur de la méthode. La figure 4.8 illustre la description d'une méthode terminale.

Cette distinction va nous servir pour le test des préconditions. En effet, comme pour les actions, les tests sur les préconditions, la condition goal et la condition Replan d'une méthode terminale vont se faire avec les croyances de l'agent qui réalise la méthode. La figure 4.8 illustre la description d'une méthode terminale. Pour les méthodes non-terminales, les mêmes tests vont se faire avec les croyances de **MySelf**, comme on peut le constater sur l'exemple de la figure 4.7.

Pour l'interaction homme-robot, cette distinction a trois avantages :

– Elle permet la production de plan sûr dans le sens où il n'y a pas d'échec dû à un manque d'information ou à une information erronée.

– Elle permet de produire des plans qui sont plus compréhensibles pour l'humain.

– Elle permet de donner au robot un comportement proactif.

Lors de la planification, nous pouvons détecter les cas où, soit l'humain ne dispose pas d'information pour réaliser sa partie du plan, soit ses informations sont erronées. Cette détection se fait au niveau des conditions *Replan*, puisque c'est à ce niveau que le test pour l'inconsistance des croyances est fait. S'il y a inconsistance, la tâche est remplacée par une nouvelle séquence de tâches qui inclut une action de communication entre **MySelf** et l'agent qui doit réaliser la

```
terminal method GetObjectFromFurniture(Agent Ag, Object Obj, Furniture F)
{
    condition
   {
       { Obj(Ag).furniture== unknown;
         Ag.MySelf == false;
       };
    subtasks{AG = SELECT(Agent, { AG.MySelf == true; });
             1:ReachAgent(AG, Ag);
             2:GiveIformationAbout(AG, Ag, Obj, furniture)>1
             3:GetObjectFromFurniture(Ag, Obj, F)>2;}

          { Obj(Ag).furniture== unknown;
         Ag.MySelf == true;
       };
    subtasks{
              1:SearchFor(Ag, Obj);
              2:PlanAgain(Ag)>1}
   };

  goal
  {
    Obj(Ag) >> Ag.objects;
    Obj(Ag).owner == Ag;
  };

  //decomposition 1
  {
    preconditions
    {
      Obj(Ag) >> F(Ag).objects;

    };

    subtasks
    {
      1:ReachFurniture(Ag, F);
      2:FreeOneHand(Ag);
      3:PickupOn(Ag, Obj, F) > 1, > 2;
    };
  }
}
```

FIGURE 4.8 – Exemple de description d'une méthode terminale

tâche, comme il est illustré sur la figure 4.8. L'inclusion de cette nouvelle séquence de tâches va permettre de corriger les informations de l'agent en charge de la réalisation de la tâche. Si c'est un humain, il pourra mieux comprendre le plan produit par le robot, mais cela va donner également au robot un comportement proactif.

Dans le cas où c'est **MySelf** qui ne possède pas l'information, la tâche est également remplacée par une séquence de tâches. Sur l'exemple de la figure 4.7, la tâche **GetObject** est remplacée par la séquence $(SearchFor(AG, Obj) \rightarrow PlanAgain(AG))$. Dans cette séquence le robot va réaliser la tâche **SearchFor** dont le but est de localiser la position de l'objet **Obj**. L'action **PlanAgain(AG)** est une action de communication entre HATP et l'exécuteur du plan. Elle lui indique que HATP demande une re-planification après l'exécution du plan courant. Cette action n'a pas d'effets et ses préconditions sont toujours valides.

Il est important d'indiquer que le choix de la séquence de tâches associée à la condition *Replan* n'est pas prédéterminé, et sa définition reste à l'appréciation du concepteur du domaine et au comportement qu'il veut donner au robot.

4.2.5 Extension sur la définition du problème et des règles sociales

La définition du problème ou but de planification ne change pas par rapport à la première version du planificateur. Il est toujours décrit soit par une liste de tâches soit par un plan partiel.

La gestion des croyances multiples ne va pas affecter grandement les règles sociales sauf la règle *états indésirables*. Pour rappel, cette règle évite l'apparition dans le plan de certains faits qu'on a défini auparavant comme des états gênants ou dangereux. Dans le cas de la planification avec croyances multiples, cette règle ne va concerner que les croyances de **MySelf**, puisque c'est le seul qui a une vision globale du plan, du fait que c'est pour lui que le plan est construit. Cela implique que ni la structure ni la syntaxe de la règle ne vont changer.

4.3 Algorithme de planification

L'algorithme que nous présentons ici est construit sur la base de l'algorithme HATP présenté dans le chapitre 3. Cet algorithme prend en entrée, le problème *Arb*, la base des croyances B et le domaine de planification D. La première phase est une phase d'initialisation, qui consiste dune part à traiter le problème pour le transformer en une liste de tâches partiellement ordonnée que l'algorithme peut traiter (ligne 1)), dautre part à initialiser la projection temporelle (ligne 2). On entre alors dans la boucle de traitement principal (ligne 3 à 23) qui ne s'arrêtera que si on a exploré tout l'arbre de recherche, ou bien si la limite de temps a été atteinte. La première étape est de filtrer la liste de tâches de toutes les tâches qui sont déjà réalisées (ligne 4). Pour cela, la tâche doit vérifier deux conditions :(1)la tâche n'a pas de prédécesseur ou ses prédécesseurs sont déjà réalisés. (2) la condition *goal* de la méthode doit être vérifiée. Si après le filtrage la liste de tâches est vide, cela indique qu'on a un plan valide (ligne 5 à ligne 8). L'étape suivante est une

Algorithme 4.1 : Algorithme HATP pour la gestion des croyances multiples

Données : B, Arb
T ← *leaves*(Arb)
Prj = ∅
répéter
  Pré-traitement sur les tâches
  si T = ∅ **alors**
    P ← produire_plan(Arb, Prj)
    Ajouter P dans la pile des plans valides
    Explorer nouvelle branche
  si $P_{REF} \neq \emptyset$ **alors**
    comparer le plan P au plan de référence P_{REF} et couper la branche si nécessaire
  $T_0 \leftarrow \{t \in T \mid predecesseurs(t) = \emptyset$, B satisfait $precond(t)$, $lock(t)$ inactif$\}$
  Sélectionner une tâche pour traitement
  si $t \in T_0$ *est une action* **alors**
    Modifier B selon les effets
    Ajouter t a la projection Prj
    Mettre à jour les coûts
  sinon
    si *conditions de réalisation sont vérifié*
    alors
      remplacer t avec TL
    sinon
      traitement habituel sur une méthode
jusqu'à *limite de temps atteinte* ou *arbre d'exploration entièrement parcouru*

optimisation (branch-and-bound) qui permet de limiter l'espace de recherche (ligne 9 à ligne 10). La prochaine étape est de sélectionner une tâche à traiter (ligne 11 et ligne 12). Si la tâche sélectionnée est une action, ses effets sont appliqués sur la base des croyances, l'action est ajoutée dans la liste de projection et les coûts sont mis à jour (ligne 13 à ligne 16). Dans le cas où la tâche est une méthode, les conditions de réalisation *Replan* de la méthode sont testées, ce qui permet de savoir si toutes les informations nécessaires à la réalisation de la tâche sont disponibles (ligne 18). Si ces conditions sont vérifiées, alors on est dans le cas où l'on doit planifier pour acquérir ou donner de l'information. Dans tous les cas, la méthode est remplacée par la liste de tâches correspondante (ligne 20). Si les conditions de réalisation ne sont pas vérifiées, on considère que l'on est dans le cas normal et on effectue le traitement habituel (décomposition, remplacement et création de nouvelles branches d'exploration) (ligne 22).
Il est nécessaire de noter que pour les tests des conditions *goal*, des préconditions et l'application des effets, le planificateur suit les contraintes imposées durant l'écriture du domaine, c'est-à-dire qu'une précondition pour une action va être testée sur les croyances de l'agent qui la réalise. Les effets vont affecter les croyances d'un ensemble d'agents déterminés par l'action. Pour les méthodes, c'est la même chose : si on traite une méthode terminale on va effectuer tous les tests par rapport aux croyances de l'acteur de la méthode. Pour les méthodes qui ne sont pas terminales, tous les tests vont être effectués par rapport aux croyances propres de l'agent **MySelf**.

4.4 Exemples et fonctionnement

4.4.1 Exemple : Gestion des croyances

Dans cet exemple, nous avons trois agents : un robot (*Jido*) et deux humains (*John* et *Tom*). Les agents évoluent dans un appartement composé de deux pièces *room1* et *room2*. Nous supposons que le robot ne peut pas se déplacer. Dans *room1* il y a une table, et une bibliothèque qui contient des livres. Sur la table il y a une paire de lunettes. *Jido* et *Tom* se situent dans *room1*, *John* est dans *room2*.

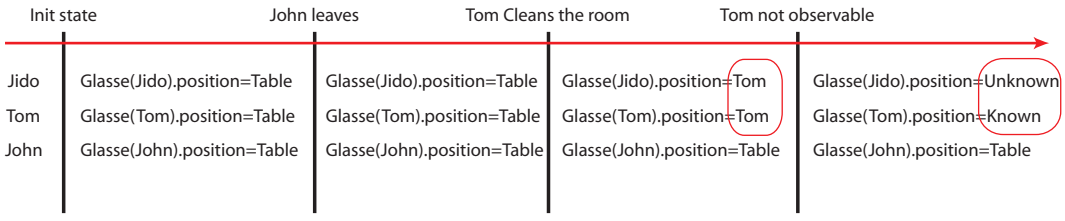
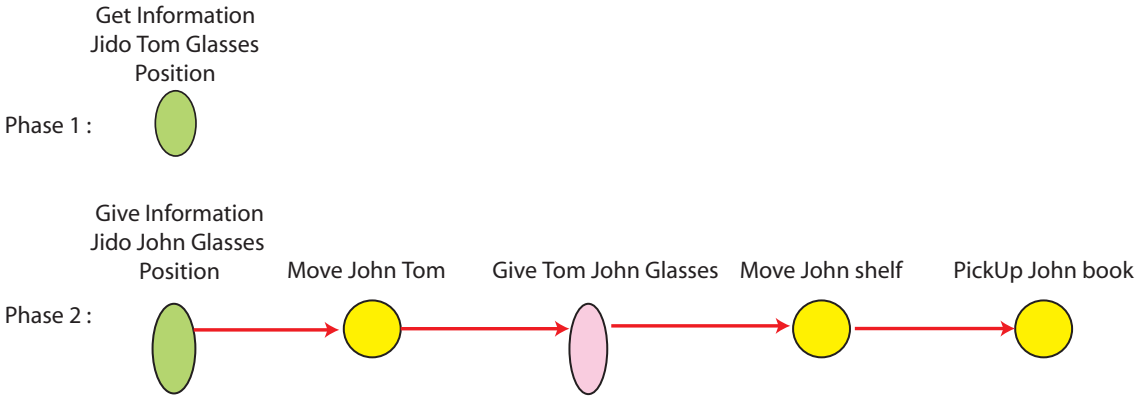


FIGURE 4.9 – Évolution de la base des croyances

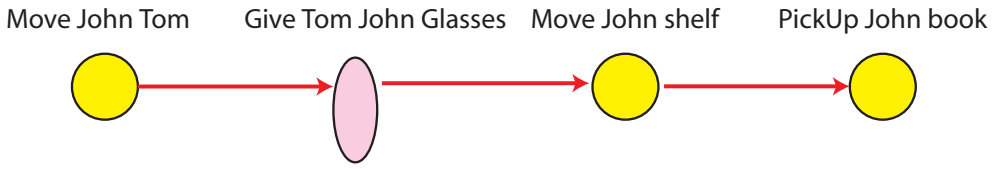
Nous supposons qu'au départ tous les agents ont les mêmes croyances. Nous supposons également que *John* sort de l'appartement et y revient après un moment. Entre temps, *Tom* fait du ménage et il range les lunettes dans *room2*. Sur la figure 4.9 nous pouvons voir l'évolution

des croyances des agents d'après les observations de *Jido*.

On peut voir que les croyances de *John* restent les mêmes puisqu'il n'a pas pu observer toutes les actions effectuées durant son absence. Pour *Jido* et *Tom*, nous remarquons que leurs bases de croyances ont évoluées. *Tom* va opérer des changements, ce qui va modifier ses états de croyance. *Jido* peut observer partiellement ces changements, ce qui va également modifier ses états de croyance. *Tom* va prendre les lunettes qui se trouvaient sur la table pour les ranger et *Jido* peut l'observer. Leurs bases de croyances vont évoluer de $Glasses.position == Table$ à $Glasses.position == Tom$. Dès que *Tom* passe de room1 à room2 pour aller ranger les lunettes, *Jido* ne peut plus observer les actions de *Tom*, ce qui a pour conséquence d'engendrer de nouveaux états de croyances. Pour *Jido* ces croyances passent de $Glasses(Jido).position == Tom$ à $Glasses(Jido).position == Unknown$ et pour *Tom*, elles passent de $Glasses(Tom).position == Tom$ à $Glasses(Tom).position == Known$. En effet, le robot *Jido* ne peut pas savoir ce que *Tom* a fait des lunettes dans *room2*, car il ne peut pas l'observer, mais, étant donné que *Tom* est le dernier à avoir manipulé les lunettes, il est le seul à connaître leur position.



(a) Plan produit par HATP avec gestion des croyances



(b) Plan produit par HATP classique

FIGURE 4.10 – Comparaison de plan produit par HATP avec et sans gestion des croyances

Si nous supposons que *John* revient et qu'il a pour but de lire un livre, ce but va se traduire par $(GetObjet(John, Livre) \wedge GetObjet(John, Lunette))$. En supposant que *Jido* connaisse le but de *John* alors il peut planifier pour l'assister. La figure 4.10(a) représente le plan produit par *Jido*. Ce plan est calculé en deux phases :

– Phase 1 : *Jido* récupère l'information de la position des lunettes chez *Tom*.
– Phase 2 : cette phase dépend de la première. Si *John* peut entendre la communication

entre *Jido* et *Tom*, cela affecte sa base de croyances et le planificateur ne produit que la séquence qui permet à *John* de réaliser son but. Dans le cas contraire, *John* n'a pas pu entendre la communication. Le planificateur introduit alors une action de communication en plus de la séquence d'actions nécessaire pour informer *John* de la position des lunettes.

Si le problème est résolu avec HATP classique (sans croyances multiples), la base de faits de départ peut être : soit celle dans laquelle *Jido* a la croyance que les lunettes sont chez *Tom*, soit celle dans laquelle il ne connaît pas leur position. La figure 4.10(a) illustre le plan produit avec la première base de faits (les lunettes sont chez *Tom*). Ce plan peut susciter chez *John* de l'incompréhension, car dans ses croyances les lunettes sont sur la table. Avec la seconde base de faits, le planificateur ne trouve pas de plan, ce qui va donner au regard des humains une image d'inutilité du robot.

4.4.2 Exemple : Amélioration de la synchronisation entre agents

Dans cet exemple, nous supposons que nous avons deux agents, un humain *Tom* et un robot *Jido*. Les deux agents ont pour but de construire une table. Pour faciliter l'illustration des plans qui vont être produits, nous supposons que la table est composée d'un pied et d'une planche. La construction de celle-ci suit la procédure : (1) assembler le pied et la planche. (2) fixer le pied à la planche. On suppose que la fixation du pied à la planche ne peut être faite que par un humain. Les agents évoluent dans un espace clos, un atelier par exemple. Dans cet espace il y a deux armoires *cupboard1* et *cupboard2* qui contiennent respectivement les planches et les pieds. L'assemblage doit se faire sur une table se trouvant dans l'atelier.

La figure 4.11 représente un plan produit par HATP classique. Ce plan est correct, mais il présente un inconvénient au niveau de la synchronisation entre les deux agents. Dans ce plan, on suppose que *Tom* sait implicitement quand *Jido* a fini d'assembler la table et cela, sans tenir compte des observations que *Tom* peut faire. Dans la réalité, un tel plan obligera le partenaire humain à suivre toutes les actions du robot pour savoir à quel moment il doit intervenir.

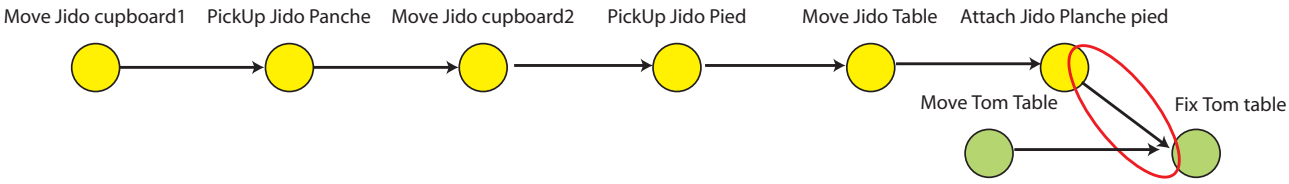


FIGURE 4.11 – Plan produit par HATP classique

En soumettant le même problème à HATP avec gestion des croyances, celui-ci produit le plan en fonction des observations que *Tom* peut faire. Il va produire le même plan que celui de la figure 4.11 si *Tom* peut observer les actions et, de ce fait, sa base de croyances est affectée par les effets des actions de *Jido*. Dans le cas où *Tom* ne peut pas faire d'observation, le

planificateur produit le plan de la figure 4.12 dans lequel il inclut une action de communication pour prévenir *Tom* que la table est assemblée. En effet, le planificateur rend plus souple les contraintes de précédence entre les agents, en prenant en compte leurs connaissances des états du monde et en les confrontant à celles du robot. Dans l'exemple, comme le robot avertit l'humain quand il doit agir, Tom est libéré de la contrainte de surveiller les agissements du robot pour savoir quand intervenir, ce qui rend l'interaction plus appréciable du point de vue de l'humain.

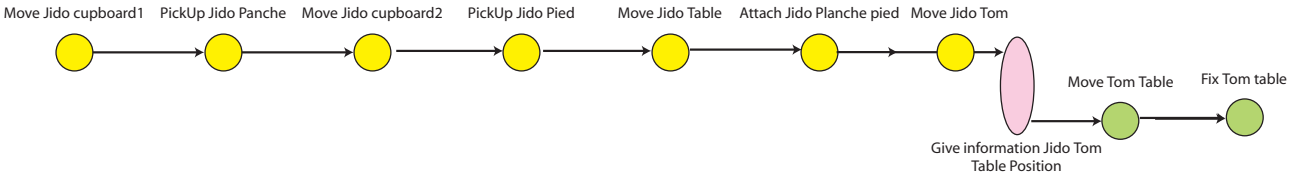


FIGURE 4.12 – Plan produit par HATP avec gestion des croyances

Nous considérons le même exemple avec un autre problème pour lequel le but du robot est d'aller chercher un pied de table et de le donner à l'humain. Cela se traduit par la méthode $GetObject(Tom, Pied)$. On suppose que le robot ignore la position du pied. Dans ce cas, deux alternatives se présentent au robot :

1. Le robot exécute la séquence $Move(Jido, Tom) \rightarrow GetInfo(Jido, Tom, Pied, Position) \rightarrow Replan()$. Il commence par se déplacer vers l'humain pour acquérir l'information sur la position, puis re-planifie. Cette alternative est possible si et seulement si la croyance de l'humain est différente de $Unknown$.
2. Le robot exécute la séquence $Move(Jido, cupboard1) \rightarrow Look(Jido, cupboard1) \rightarrow Replan()$. Il commence par atteindre cupboard1, puis il effectue une observation pour voir si le pied est là ou non, puis il re-planifie. Si l'objet est trouvé, on entre en cycle normal de décomposition. Sinon le robot va visiter une autre position et cela récursivement jusqu'à trouver l'objet ou bien jusqu'à ce qu'il n'y ait plus de position à observer. Dans ce cas, il y a échec.

Dans cet exemple, nous avons vu que le robot entrelace planification et exécution. Il planifie pour résoudre un problème donné, mais également pour acquérir des informations qui lui sont utiles pour résoudre le problème, ce qui lui procure une autonomie supplémentaire.

4.5 Conclusion

Dans ce chapitre, après avoir étudié les approches existantes pour la planification avec croyances mutuelles, nous avons présenté une extension du modèle HATP permettant la gestion des croyances des différents agents intervenant dans le plan. Nous avons étendu le formalisme

de définition des tâches et proposé l'ajout d'actions de communication dans le modèle. Nous avons montré comment utiliser la structure HTN de HATP pour produire des plans qui tiennent compte des différentes croyances des agents, et nous avons noté que l'utilisation de ces structures hiérarchisées n'impliquent pas la modification de l'algorithme de planification, ce qui est un avantage puisqu'à tout moment on peut réutiliser le modèle de HATP classique.

Nous avons pu constater l'intérêt de l'inclusion des croyances multiples dans le modèle pour l'interaction homme-robot. Cette inclusion permet une gestion plus efficace et plus souple des liens croisés, en remplacant les liens fragiles par des actions de communication.

Néanmoins, ce modèle utilise des notions géométriques dans sa représentation symbolique comme, par exemple, la notion de coprésence qui permet une gestion plus réaliste du transfert d'information entre agent. Ce qui rejoint la critique du modèle HATP faite dans le chapitre 3.

5

Hybride-HATP : planification avec contraintes géométriques

Nous avons démontré dans le chapitre 3 qu'il est nécessaire de contrôler le comportement d'un robot qui cohabite avec des humains. Cela passe par le contrôle de la qualité sociale du plan que le planificateur produit, mais cela n'est pas suffisant. Car le contrôle du comportement d'un robot se fait sur deux niveaux. Au niveau décisionnel, ce que nous avons étudié dans les chapitres 3 et 4, et également au niveau fonctionnel, c'est-à-dire au niveau de la réalisation du plan. En effet, les décisions prises à ce niveau peuvent affecter considérablement la qualité sociale d'un plan. Pour éviter cela, il est nécessaire de conditionner les décisions prises au niveau fonctionnel afin de s'assurer de la qualité sociale du plan produit et réalisé.

Dans ce chapitre, nous allons présenter notre approche d'une planification hybride pour l'interaction homme-robot qui consiste en un couplage entre notre planificateur symbolique et un planificateur géométrique. Ce travail a été effectué en collaboration avec M. Amit Kumar Pandey, actuellement en thèse au LAAS, pour la partie planification géométrique.

5.1 Travaux voisins

Dans la littérature il existe plusieurs approches pour faire fonctionner un planificateur symbolique avec un planificateur géométrique. [Guitton 10] a fait une classification des différentes approches existantes :

– L'approche classique : Planification de tâche suivie par la planification géométrique.

– L'approche non conventionnelle : Planification géométrique suivie par la planification de tâche.

– L'approche hybride : Les deux planificateurs fonctionnent en parallèle.

5.1.1 Approche classique

On l'appelle approche classique, car c'est l'approche la plus répandue dans les architectures des robots [Sisbot 06]. Dans cette approche, La planification de tâche et la planification géométrique sont considérées de façon séquentielle. Le planificateur de tâche reçoit un but et produit une séquence d'actions symboliques discrètes pour satisfaire le but. Dans une deuxième étape, le plan discret est transmis au planificateur géométrique, qui doit suivre la séquence et respecter des contraintes géométriques de l'environnent et du robot pour produire un plan continu [Fainekos 07], [Belta 07], [Caldiran 10].

5.1.2 Approche inverse

Par apposition à l'approche classique, cette approche se base sur un pré-calcul géométrique qui produit un graphe. Les nœuds sont calculés à partir des buts locaux et les contraintes géométriques sur les actions. Les arcs sont calculés en se basant sur la planification de déplacement. Cette approche est une approche spécifique à un problème particulier qui est la planification de mission. Nous pouvons citer le travail de [Chanthery 05], qui a utilisé cette approche pour des missions d'explorations de zones de combat pour des drones.

5.1.3 Approche hybride

Dans cette approche, les deux planificateurs travaillent en parallèle, c'est-à-dire que le planificateur de tâche sélectionne une action, et le planificateur géométrique la valide, en calculant sa faisabilité en tenant compte des contraintes géométriques de l'environnement et du robot. En cas d'échec, le planificateur de tâche reprend la planification en considérant les causes de l'échec. Dès les années 80 des travaux ont été menés dans ce sens, notamment dans le cadre de la planification d'assemblage [Gottschlich 93]. Nous pouvons citer les systèmes HANDY [Lozano-Pérez 87], SHARP [Laugier 85] et SPARA [Mazon 90].

Des travaux orientés vers la robotique autonome ont commencé à émerger à partir de 2001 avec le planificateur SHAPER [Guéré 01]. Ce planificateur est basé sur le langage STRIPS augmenté par des faits numériques, pour représenter les aspects géométriques du problème. L'idée de ce planificateur est d'accélérer le processus de planification en apprenant la topologie

d'un graphe d'accessibilité entre les états.

En 2004 un nouveau planificateur a vu le jour, aSyMov [Gravot 04], [Cambon 05]. Il est le premier qui combine réellement planificateur symbolique et planificateur géométrique. Il est dédié aux problèmes de manipulation en robotique autonome. Le planificateur utilise le langage PDDL2.1 de niveau 2 [Fox 03] (avec typage des symboles et attributs numériques). aSyMov fait intervenir un ensemble de prédicats symboliques particuliers qui représentent des états géométriques. Pour la description des préconditions et des effets de ses actions, il utilise des prédicats symboliques classiques et ses prédicats particuliers. Ces derniers vont être transformés en contraintes pour le planificateur de déplacement. aSyMov a été mis au point sur la base du planificateur métrique-FF [Hoffmann 02], qui est un planificateur à recherche heuristique en avant dans un espace d'état. Pendant la recherche de plan et à chaque fois que le planificateur rencontre une action nécessitant un mouvement, il valide l'action par le planificateur de déplacement. Le but de ce planificateur est de générer des plans faisables dans le monde réel. En effet, un plan produit par un planificateur symbolique n'est pas forcement réalisable dans la réalité, du fait que le planificateur symbolique ne modélise qu'une portion de l'environnement. C'est pourquoi, ce modèle suggère le couplage de deux approches (symbolique et géométrique) pour produire des plans qui soient faisables [Gravot 05].

D'autres travaux ont suivi la même démarche qu'aSyMov. Nous pouvons citer les travaux de [Guitton 09] qui combine un planificateur de tâche HTN (Hierarchical Task Network) et un planificateur de déplacement basé sur l'approche RRT (Rapidly-exploring Random Tree) [LaValle 98]. Ce planificateur ne vise pas à résoudre le problème de la manipulation en robotique, mais traite la problématique du déplacement d'un robot pour la réalisation d'une mission. Le planificateur augmente la définition des actions par des préconditions et des effets géométriques. Les préconditions définissent géométriquement la manière de réaliser les actions. Ces préconditions sont ensuite envoyées au module de raisonnement géométrique qui se charge de trouver une solution entre la configuration actuelle et la configuration désirée. Les effets géométriques permettent de mettre à jour les configurations du robot en fin de réalisation de l'action.

Nous pouvons également citer les travaux de [Wolfe 10], dans lesquels un planificateur dédié à la manipulation en robotique a été développé. Ce planificateur utilise la même association que dans [Guitton 09], c'est-à-dire un planificateur de tâche HTN et un planificateur de déplacement basé sur RRT. Le modèle prend en charge la description des opérateurs de très bas niveau, par exemple des opérateurs qui décrivent les mouvements de la base, du torse, des bras ou des pinces. Ces opérateurs de bas niveau vont décrire la réalisation d'opérateur de plus haut niveau appelé HLA (High-Level Actions) tel que **Pickup** ou **Putdown**. Le planificateur utilise la structure hiérarchisée du planificateur symbolique HTN, mais n'exploite pas son système de transition qui va être assuré par la partie géométrique.

Il existe d'autres approches développées pour des problèmes spécifiques, qui sont radicalement différentes. Elles ne combinent pas planification symbolique avec planification géométrique, mais

modifie plutôt l'algorithme de planification géométrique afin qu'il puisse traiter des tâches de haut niveau. Nous pouvons citer les travaux de [Conner 07] [Choi 09] qui localisent l'ensemble des actions du robot et l'espace d'états dans l'espace de configuration du robot et des objets. Il utilise ensuite cette nouvelle représentation comme entrée du planificateur géométrique et trouve un chemin solution. Comme ces travaux sont basés sur des approches très différentes de la notre, nous ne les détaillerons donc pas davantage dans ce manuscrit.

5.2 Interface entre planificateur symbolique et planificateur géométrique

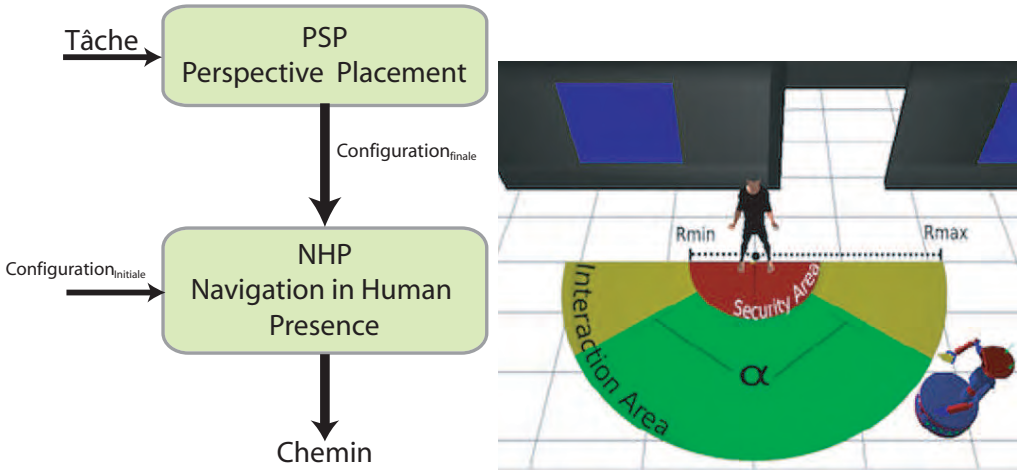
5.2.1 Description du planificateur géométrique

Le planificateur géométrique utilisé s'appelle MHP pour (Motion in Human Presence)[Sisbot 07b]. C'est un planificateur qui peut produire des plans qui prennent en compte la présence des humains et les contraintes que cela engendre. Le planificateur MHP se décompose en deux modules, comme représenté dans la figure 5.1.a :

PSP(PerSpective Placement) : c'est un générateur de configurations qui permet au robot de décider où se placer et quelle configuration choisie pour réaliser l'action [Luis F. Marin-Urias 08]. Des études psychologiques menées sur le "perspective placement" et le "perspective taking" [Moll 06],[Flavell 92], ont montré que la capacité de l'être humain à choisir comment se placer pour réaliser une tâche dépend de son intention, de la tâche à réaliser et de la personne qu'il a en face de lui. Le module essaye de reprendre cet esprit en rajoutant des contraintes sur les configurations qu'il va générer, ce qui va donner au robot un comportement socialement acceptable vis-à-vis de l'humain. Ces contraintes sont une continuité des études effectuées sur l'interaction homme-robot [Huettenrauch 06], [Koay 07]. Elles sont décrites brièvement ci-dessous :

- **Zone d'interaction** : pour qu'un robot interagisse avec une personne, il faut qu'il respecte certaines règles :
  1. Zone de sécurité : elle est définie comme la distance minimale entre le robot et l'humain (figure 5.1.b).
  2. Zone d'interaction : pour qu'un robot puisse interagir avec l'humain il faut qu'il soit dans la zone d'interaction (figure 5.1.b).
- **Collision** : PSP doit s'assurer que la configuration choisie pour le robot n'est pas en collision
- **Orientation des capteurs** : les capteurs doivent être orientés vers la cible si on veut l'apercevoir.
- **Sécurité et confort pour l'humain** : les configurations sélectionnées doivent être sûres pour l'humain, elles doivent aussi respecter le confort de celui-ci si la tâche est une tâche d'interaction physique.
- **visibilité mutuelle** : dans le cas d'une tâche d'interaction, il faut s'assurer que la configuration choisie permette aux deux agents de se voir mutuellement.

NHP (Navigation in Human Presence) : c'est un planificateur de déplacement qui, en plus du calcul d'un chemin entre deux configurations, permet de calculer des configurations finales, en s'appuyant sur la structure cinématique et les préférences des autres agents [Sisbot 08]. Il a également la capacité de traiter les problèmes de déplacement et la manipulation d'un objet en présence d'humains. À partir de la configuration initiale du robot et de la configuration finale calculée par PSP, le planificateur génère un mouvement sûr, confortable et intelligible.



(a) Architecture du planificateur géométrique (b) La représentation de la zone d'interaction avec un humain

FIGURE 5.1 – Architecture du planificateur géométrique la représentation de la zone d'interaction

Dans ce chapitre, nous nous intéressons au contrôle du comportement du robot, Pour cela, nous nous intéressons au contrôle des configurations de celui-ci, c'est-à-dire le contrôle des configurations générées par PSP(PerSpective Placement) en y ajoutant des contraintes en plus de celles qu'il gère habituellement.

Pour générer une configuration, PSP a besoin de connaître le contexte dans lequel l'action doit être réalisée, c'est-à-dire qu'il ne va pas générer la même configuration pour un déplacement si c'est un déplacement pour voir un homme ou pour voir un meuble. Une fois le contexte connu, PSP crée une grille autour de sa cible, chaque cellule représente une position possible, et pour chaque cellule, PSP calcule toutes les configurations possibles. Il choisit la meilleure configuration qui respecte les contraintes citées auparavant.

5.2.2 Fonctionnement du système

La figure 5.2 illustre le fonctionnement entre le planificateur symbolique et le planificateur géométrique. Nous pouvons voir que le planificateur symbolique prend en entrée le domaine de planification, les descriptions symboliques et géométriques du monde et le problème de planification.

Le planificateur géométrique est initialisé par le planificateur symbolique. Il lui transmet toutes les données géométriques nécessaires. Une fois que celui-ci a son état du monde, il peut

mettre à jour certains faits qu'on appelle *faits géométriques*, qui sont utilisés par le planificateur symbolique.

Quand le planificateur reçoit un but, il commence son processus de planification. Durant la décomposition du but, à chaque action trouvée le planificateur symbolique demande une validation de l'action au niveau géométrique, jusqu'à trouver un plan valide pour les deux planificateurs.

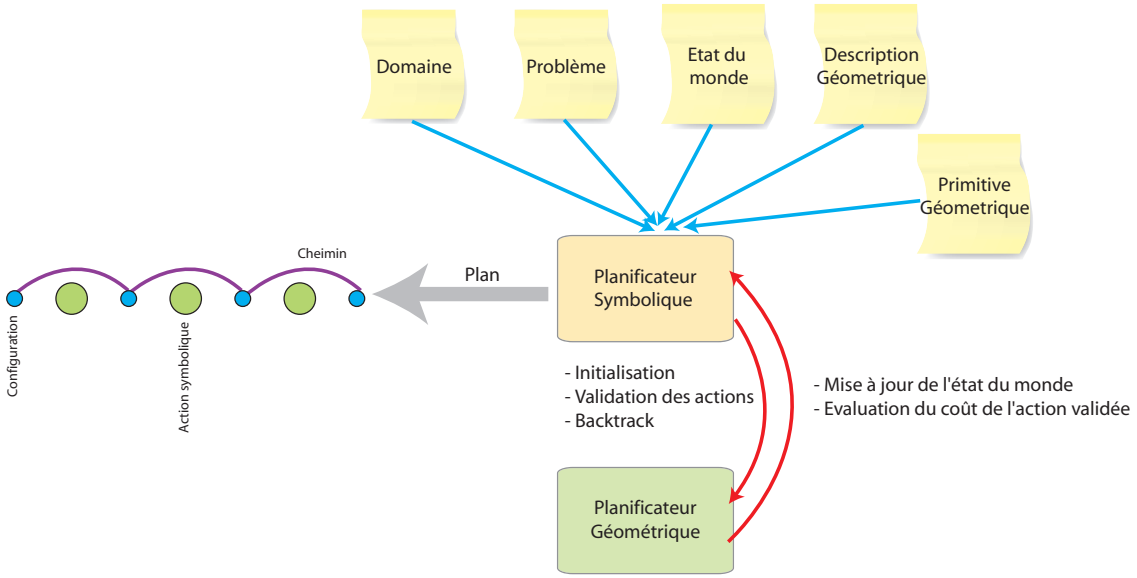


FIGURE 5.2 – Architecture pour un planificateur hybride

5.3 Extension du domaine de planification

Dans cette section, nous proposons un ensemble de modifications du planificateur HATP afin de lui permettre de collaborer avec un planificateur géométrique. Ces modifications touchent toute la structure du modèle : la base de faits, le modèle de tâche et l'algorithme de planification. Ces modifications ne vont pas toucher à la généricité du système, ce qui signifie qu'à tout moment nous pouvons utiliser le planificateur classique. Le but de cette collaboration est l'amélioration de la qualité de l'interaction homme robot, la production de plans réalisables et optimaux dans le monde réel.

5.3.1 Extension de la représentation des états du monde

Comme nous l'avons vu précédemment, le formalisme s'appuie sur la notion d'entités et d'attributs qui décrivent tous les objets et concepts existants dans le monde. Il paraît évident que cette description est incomplète, puisqu'elle ne décrit qu'une partie symbolique du monde et omet la représentation géométrique de celui-ci. Pour pallier à l'incomplétude du modèle, nous associons au modèle symbolique de chaque entité son modèle géométrique correspondant. *En* =<

$En_{symb}/En_{geom}>$. La partie symbolique En_{symb} est la même que celle définie précédemment dans le chapitre 3, chaque entité est définie par un ensemble d'attributs qui décrivent son état.

La partie géométrique En_{geom} décrit l'entité en termes de degrés de liberté de cinématique, de position, etc... Cette représentation est intégrée aux entités en ajoutant un nouveau type d'attributs. La figure 5.3 illustre un exemple de déclaration d'une base de faits. Si nous considérons la déclaration des attributs d'un agent, nous retrouvons les déclarations classiques des attributs symboliques et en plus de ces déclarations, nous trouvons la définition de l'attribut géométrique "*AgentModel*" qui fait la liaison entre la partie symbolique et la partie géométrique.

Nous avons donné un type aux attributs géométriques. Ils peuvent être soit du type **P3d_Rob** soit du type **P3d_Obj**. Cette distinction va nous permettre de créer deux classes d'entités, les entités dynamiques qui auront le type **P3d_Rob** et les entités statiques qui auront le type **P3d_Obj**. Nous définissons une entité statique comme toute entité qui a une position et une configuration fixe, par opposition à une entité dynamique qui a une position ou une configuration variable. Cette distinction est nécessaire au planificateur géométrique pour ses traitements internes.

Dans l'exemple de la figure 5.3, en observant la déclaration de l'entité type "*Place*", nous constatons que son attribut géométrique est du type statique **P3d_Obj**, ce qui est logique puisqu'un lieu n'est pas déplaçable. Si nous considérons les déclarations des entités de type "*Agent*" ou "*Object*", nous remarquons que leurs attributs géométriques sont du type **P3d_Rob**, puisque les deux ("*Agent*" et "*Object*") peuvent changer de position. Il faut bien garder à l'esprit que les attributs géométriques ne sont pas utilisés explicitement par la planification symbolique, mais sont nécessaires à l'initialisation du planificateur géométrique.

Pour la description des faits symboliques, nous avons gardé la même syntaxe que celle décrite auparavant dans le chapitre 3. Elle se base sur les entités et leurs attributs comme illustré dans la figure 5.3.

Pour compléter la description des états du monde, nous avons fait le choix d'inclure certains états géométriques, qui vont nous permettre d'accroître les capacités de raisonnement et d'améliorer la qualité sociale des plans de notre planificateur. En effet, en contraignant certains de ces états, cela nous permet d'exercer un contrôle sur les solutions que le planificateur géométrique va produire et donc avoir un contrôle sur la qualité des plans produits. Ce point sera abordé dans les sections suivantes.

Ces faits géométriques appelés aussi *Relations géométriques* ou *Relations spatiales* lient deux ou plusieurs entités par une certaine relation géométrique. Cette relation est définie par des primitives géométriques telles que distance, angle, visibilité, etc... Pour ces faits géométriques, nous avons adopté la syntaxe $name(En_j, En_k)$ qui se traduit par "Une relation spatiale qui a le nom de *name* et qui lie une paire d'entités distinctes En_j et En_k". Cette représentation est

```
factdatabase
{
    // Étape 1 : Définition des types des entités
    define entityType Place;
    define entityType Object;

    // Étape 2 : Définition des attributs
    define entityAttributes Agent
    {
        static  atom    symbol  type;
        dynamic atom    Place    posTopo;
        dynamic set     Object   objects;
        p3d_Rob                  AgentModel;
    }

    define entityAttributes Objet
    {
        static  atom    number  poids;
        dynamic atom    Agent   owner;
        p3d_Obj                 ObjetModel;
    }
    define entityAttributes Place
    {
        p3d_Obj                 ObjetModel;
    }

   // Étape 3 : Définition des geomRelations

   define GeomRelation
    {
    visible(Agent, {Agent OR Object});
    reachable(Agent, Object);
    Relational_Distance(Agent, Agent);
    nextforgive(Agent, Agent);
    nextfortalk(Agent, Agent);
    nextforlook(Agent, {Agent OR Object});
    }

    // Étape 4 : Création des entités

    livingRoom = new Place;
    Jido = new Agent;
    Glass = new Objet;
    Bottle = new Objet;

    // Étape 5 : Initialisation des attributs
    Jido.type = "Robot";
    Jido.posTopo = livingRoom;
    Jido.objects <<= Glass;
    Jido.objects <<= Bottle;
    Jido.AgentModel ="\home\salili\P3D\JidoModel.p3d";

    Glass.poids = 100;
    Glass.owner = Jido;
    Glass.ObjetModel ="\home\salili\P3D\GlassModel.p3d";

    Bottle.poids = 1000;
    Bottle.ObjetModel ="\home\salili\P3D\BottleModel.p3d";

    livingRoom.ObjetModel ="\home\salili\P3D\livingModel.p3d";
loadScene("\home\salili\P3D\SceneModel.p3d").
}
```

FIGURE 5.3 – Exemple de déclaration d'une base de faits

asymétrique : $R_1(En_j, En_k) \neq R_1(En_k, En_j)$, car une relation qui est vérifiée dans un sens ne l'est pas forcément dans l'autre. Par exemple, la relation *visible* qui lie deux agents **A1** et **A2**. L'agent **A2** peut être visible par l'agent **A1**, mais cela ne veut pas dire que l'agent **A1** est visible par l'agent **A2**. Ces relations sont également booléennes, car elles ne servent qu'à affirmer ou infirmer la validité d'une relation entre deux entités.

Pour illustrer le sens d'une relation spatiale et répondre à la question comment se fait la liaison entre le niveau symbolique et le niveau géométrique pour une relation, prenons l'exemple d'une relation que nous allons appeler "Relational_Distance". Cette relation va lier deux entités de type agent, comme elle a été définie dans l'exemple de la figure 5.3. Au niveau symbolique, cette relation va répondre à la question "est-ce qu'un des deux agents a violé l'espace vital [1] de l'autre ?".

Au niveau géométrique, cette relation représente une primitive ou une fonction qui teste la distance entre deux agents et la compare à une certaine valeur ($\sqrt{(x1-x2)^2+(y1-y2)^2} > valeur$). Si la distance calculée est inférieure à la valeur de référence, la relation spatiale prendra la valeur "*vraie*", c'est-à-dire que l'espace vital entre les deux agents est violé. Sinon elle prendra la valeur "*faux*". Le nom de la relation spatiale fait la liaison entre sa représentation symbolique et sa représentation géométrique.

Comme nous l'avons noté précédemment, les relations géométriques sont représentées de façon symbolique dans le planificateur de tâche. Ce dernier n'a aucune influence sur la qualité de ces relations en termes de diversité et richesse. La diversité et la richesse de celles-ci dépendent exclusivement des capacités du planificateur géométrique à fournir des primitives de haut niveau, et également de la possibilité de faire des compositions entre primitives de base.

Nous avons fait le choix de différencier la syntaxe des faits symboliques et des faits géométriques pour deux raisons : (1) pouvoir garantir la généricité de la représentation des états du monde, puisque la déclaration et l'utilisation des faits géométriques restent optionnelles dans tout le système. (2) pouvoir faciliter la gestion des backtracks que nous détaillerons à la section 5.4.3.

En examinant l'exemple de la figure 5.3, nous pouvons constater que la création et l'initialisation de la base de faits se fait en cinq étapes :

– La première étape est la définition des types d'entités. Seule l'entité de type "*Agent*" est prédéfinie par le système.

– La deuxième est la définition des attributs avec leurs types. C'est à ce niveau que la liaison entre une entité et sa représentation géométrique est faite, mais cela reste optionnel car nous pouvons avoir des entités qui n'ont aucune existence physique. Par exemple, nous pouvons

1. Un espace vital est une distance entre deux individus qui est nommée distance relationnelle. Cette dernière varie en fonction du degré d'affectivité liant les individus.

créer l'entité *association* qui regroupe des individus d'un certain type.

– La troisième est la création des relations spatiales avec leur type. On peut voir sur l'exemple qu'une relation peut regrouper plusieurs types. Par exemple, la relation "*visible(Agent, Agent OR Object)*" qui définit la notion de visibilité, cette relation peut lier un agent à un autre agent ou bien un agent à un objet.

– La quatrième étape est la création des entités. Le système va automatiquement créer l'instance de l'entité déclarée et lui associer les attributs correspondants.

– La cinquième étape est l'initialisation des attributs. Dans cette étape, les attributs symboliques sont initialisés comme nous l'avons décrit précédemment au chapitre 3. Pour les attributs géométriques, l'initialisation se fait par la lecture d'un fichier. Au niveau symbolique, nous ne gardons que l'adresse du fichier pour l'utiliser lors de l'initialisation du planificateur géométrique. La représentation du monde n'est pas complète, il manque certains détails qui sont cruciaux pour le niveau géométrique. Par exemple, pour la définition d'une chambre au niveau symbolique, nous n'allons pas nous intéresser aux murs, mais au niveau géométrique ceux-ci sont très importants, puisqu'ils permettent de générer des déplacements en évitant les collisions. Pour cela, nous avons rajouté la commande ***loadScene*** qui charge dans le système l'adresse d'un fichier qui contient la description de l'environnement 3D. Pour les relations spatiales, le système calcul dynamiquement toutes les combinaisons possibles des relations en se basant sur les relations déclarées, les types qu'elles font intervenir et les entités déclarées ; l'instanciation de ces relations va être faite par le niveau géométrique.

5.3.2 Extension du modèle des actions

Le modèle des actions est également affecté par la collaboration entre la planification de tâche et la planification géométrique. En effet, une action sur le plan symbolique doit avoir sa correspondante sur le plan géométrique. De plus, au niveau symbolique, nous devons introduire la notion de contrainte géométrique, qui permet au planificateur symbolique de garder le contrôle sur les solutions produites par la planification géométrique en incluant des contraintes supplémentaires en entrée de celle-ci.

Au niveau symbolique les actions sont décrites par un tuple

$$A =< Name, Par, Precond, Eff, GF, Co, Du, EC >$$

Où :

– $Name$ définit le nom de l'action.

– Par représente les paramètres de l'action. C'est une liste typée et non ordonnée d'entités.

– $Precond$ représente les préconditions de l'action. Ces préconditions doivent être vérifiées avant de pouvoir réaliser l'action. Une précondition est une conjonction de faits (symbolique

et géométrique), ces faits portent soit sur le test des valeurs des attributs associés aux entités présentes dans Par, soit sur le test des valeurs des relations spatiales qui impliquent ces mêmes entités.

– $Eff =< Eff\_expected, Eff\_side >$ représente les effets de l'action. Ils définissent les changements que le monde va subir si l'action est réalisée. Nous définissons un effet comme changement de valeur sur les attributs des entités présentes dans la liste des paramètres et sur les relations spatiales qui les lient. Nous avons fait le choix de faire la distinction entre deux types d'effets "$Eff\_expected$" et "Eff_Side". Nous définissons les "$Eff\_expected$" comme les effets attendus de l'action sur le monde et qui n'agissent que sur les états des entités concernées par l'action. Les "Eff_Side" sont des effets de bord de l'action. Nous faisons cette distinction car nous considérons que les "Eff_Side" sont des effets non-contrôlables par le planificateur symbolique. Ces effets non-contrôlables concernent exclusivement les faits géométriques. Le planificateur géométrique est le seul à connaître les états de ces faits, mais il peut également calculer l'évolution de leur état suite à la réalisation d'une action. La déclaration de ces effets reste optionnelle au niveau du planificateur symbolique.
Concernant la mise à jour des faits géométriques, nous avons faits les choix suivant :

1. les faits géométriques vont être mis à jour par le planificateur géométrique (i.e. le planificateur de tâche ne peut pas modifier les instances des relations spatiales).
2. Si les effets de bord sont déclarés avec un ensemble vide, le planificateur géométrique mettra à jour toutes les relations spatiales existantes dans le système. Dans le cas contraire, il se contentera de mettre à jour les relations spatiales déclarées dans les effets de bord. Cette restriction va permettre dans certains cas de réduire le temps de calcul pour la mise à jour des faits géométriques.

Exemple : Prenons l'exemple d'un agent qui réalise une action de déplacement. Considérons aussi la relation spatiale visibilité (*visible(Agent, objet)*) qui indique si un objet est visible ou non par un agent. L'effet attendu de l'action de déplacement est que l'agent change de position, mais il se peut que l'action ait aussi des effets secondaires ou effets de bord, qui affectent l'état des relations spatiales dans le monde. Le déplacement peut affecter la visibilité de l'agent qui la réalise, mais aussi la visibilité des autres agents, en se positionnant dans leurs champs de vision.

– GF représente un flag qui indique si l'action est une action physique ou non (c'est-à-dire nécessitant une planification géométrique [2]). C'est à ce niveau que le lien entre l'action symbolique et sa correspondante géométrique se fait, en indiquant le nom de la fonction géométrique correspondante. La déclaration des flags est optionnelle.

– Co représente une fonction coût associée à l'action et permet d'évaluer le coût lié à l'application de l'action. Si l'action en question est une action géométrique, l'estimation du

2. Une action ne nécessitant pas une planification géométrique est une action qui n'a pas d'effet géométrique. Exemple l'action *parler* n'a d'effet que sur les croyances d'un autre agent donc elle n'est pas une action géométrique

coût est donnée par le planificateur géométrique. Dans les autres cas, nous utilisons la fonction coût comme elle a été définie dans la section 3.3.1.

– *Du* représente une fonction de calcul du temps associée à une action. Elle permet d'évaluer la durée moyenne du temps de réalisation de l'action. Nous utilisons la même fonction durée que celle qui a été définie dans la section 3.3.1.

– *EC* représente un ensemble de contraintes que la réalisation de l'action doit satisfaire. Ces contraintes sont décrites par une conjonction de faits géométriques et sur les valeurs qu'elles doivent prendre. Ces contraintes sont prises en compte par le planificateur géométrique lors de la réalisation d'une action. Ces contraintes sont des contraintes dures, c'est-à-dire que si le planificateur ne peut pas les satisfaire, nous considérons que l'action n'est pas réalisable. L'association d'une contrainte à la déclaration d'une action est considérée comme une association rigide. En effet, cela exige du planificateur géométrique de satisfaire la contrainte à chaque fois qu'on lui demande de valider l'action. Cette association rigide correspond parfaitement au modèle d'action des planificateurs hybrides [Gravot 04] et [Guitton 09] que nous avons étudié précédemment dans la section 5.1.
Dans notre approche, nous considérons que l'association d'une contrainte à une action dépend du but pour lequel l'action est réalisée. Le tableau 5.1 illustre cela. Nous pouvons voir que pour satisfaire quatre buts différents, nous associons quatre contraintes différentes à une même action de déplacement. Comme nous sommes dans une structure HTN, les buts vont être déterminés au niveau des méthodes et cela va être décrit dans la section suivante.

```
action Move(Agent A1, Place P)
{

    // Déclaration des préconditions

    preconditions {A1.posTopo != P;};

    // Déclaration des expected_effects

    expected_effects {A1.posTopo == P;};

    // Déclaration des side_effects

    side_effects {visible(A1, ?);
                  reachable(A1, ?);
                 };

    // Flag géométrique et fonction de calcul de la durée de l'action
    GeomeFlag(Gotofunction);
    duration { MoveDurationFonction(A1,P); };

}
```

FIGURE 5.4 – Exemple de description d'une action dans HATP hybride

La figure 5.4 illustre la déclaration de l'action *Move(Agent A1, Place P)*. Cette action a deux paramètres : un agent **A1** et un lieu **P**. Le but de l'action est de déplacer **A1** de sa position

actuelle à une position finale **P**. Les préconditions sont que l'agent **A1** ne soit pas à la position cible **P**. L'effet direct de l'action est que la nouvelle position de **A1** est à **P**. Les effets de bord de l'action portent sur la visibilité de **A1** avec tous les objets ou agents. Le Symbole "?" définit une variable libre. Le type de la variable est déterminé par la déclaration de la relation spatiale. *GeomeFlag* définit que l'action est une action géométrique, ce qui implique qu'il n'y a pas de déclaration pour la fonction de calcul de coût, car l'estimation du coût est calculée par le planificateur géométrique. Le symbole se trouvant entre parenthèses dans *GeomeFlag* représente un pointeur qui désigne la fonction géométrique correspondante au "*Move*" symbolique.

Nous complétons l'exemple avec le tableau 5.1. Nous pouvons voir sur cet exemple qu'à partir d'une action "Move" qui est décrite dans la figure 5.4 de façon générique, nous pouvons modéliser quatre actions de déplacement qui sont complètement différentes du point de vue du comportement de l'agent qui va exécuter l'action.

But	**Action**	**Contrainte**
Déplacement pour donner un objet	Move(Agent A1, Place P)	nextforgive(A1, A2)==true ;
Déplacement pour récupère un objet	Move(Agent A1, Place P)	nextforget(A1, A2)==true ;
Déplacement pour interaction verbale	Move(Agent A1, Place P)	nextfortalk(A1, A2)==true ;
Déplacement pour faire une observation	Move(Agent A1, Place P)	nextforlook(A1, Obj)==true ;

TABLE 5.1 – Exemple d'association de différentes contraintes avec une action de déplacement

5.3.3 Extension sur le modèle des méthodes

Le modèle des méthodes est également affecté par la collaboration entre la planification de tâche et la planification géométrique. En effet, c'est au niveau des méthodes que se fait la gestion des contraintes, qui vont être associées aux actions en bas de la hiérarchie.

Une méthode est décrite par le tuple

$$M =< Name, Par, goal, D, EC >$$

Où :

– $Name$ représente le nom de la méthode.

– Par représente une liste typée des paramètres de la méthode.

– *goal* est une liste de conditions qui définit le but ou les méta-effets associés à la méthode. Si ces conditions sont respectées alors la méthode est réduite à un ensemble vide. Ces conditions peuvent porter sur les faits symboliques et géométriques.

– D comme définie dans le chapitre 3, représente une liste de couples $< P_i, TL_i >$. Cette liste représente l'ensemble des décompositions possibles pour la méthode. Ainsi pour chaque décomposition i, si la précondition P_i est validée, alors la liste partiellement ordonnée TL_i est une façon possible de réaliser la méthode. Les préconditions peuvent porter sur les faits symboliques et géométriques.

– *EC* représente un ensemble de contraintes que la réalisation de la tâche doit satisfaire. Elles sont décrites par une conjonction de faits géométriques et les valeurs qu'elles doivent prendre. Les contraintes associées à une tâche n'ont aucun effet direct sur celle-ci, c'est-à-dire quelles n'ont aucune influence sur le choix ou la validation de ses décompositions. Les contraintes associées à une méthode sont propagées sur toutes ses descendances. C'est-à-dire, qu'à chaque décomposition valide de la tâche est associée un ensemble de contraintes qui étaient initialement associées à la méthode.

Une liste de tâches a la même structure que celle du chapitre 3 (section 3.3.2) avec une petite différence au niveau des contraintes. Toute tâche T_{ij} qui est présente dans une liste de tâches TL_i et qui appartient à une méthode M peut avoir son propre ensemble des contraintes $EC_{T_{ij}}$. Cet ensemble est indépendant de l'ensemble des contraintes EC_M associé à la méthode M. Si la liste de tâches TL_i est validée, alors l'ensemble des contraintes EC_M sera propagé à la liste de tâches T_i. Le nouvel ensemble des contraintes $EC_{T_{ij}}$ associé à la tâche T_{ij} sera fusionné avec EC_M et donnera un nouvel ensemble $EC_{T_{ij}} = \{EC_{T_{ij}} \sqcup EC_M\}$.

La figure 5.5 illustre un exemple de déclarations de deux méthodes *GetObject(Agent Ag, Object Obj)* et *GetObjectFromFurniture(Agent Ag, Object Obj, Furniture F)*. *GetObject* décrit les possibilités pour un agent *Ag* d'acquérir un objet *Obj*. Il y a trois façons de faire, soit c'est l'agent lui-même qui va chercher l'objet, soit c'est un autre agent qui va chercher l'objet et le lui transmet, soit l'objet est déjà chez un agent et celui-ci va le transmettre à l'agent *Ag*.

La deuxième méthode *GetObjectFromFurniture*, décrit comment un agent *Ag* peut faire pour acquérir un objet *Obj* qui se trouve sur un meuble *F*. Dans ce cas-là, nous n'avons qu'une seule décomposition : l'agent se déplace jusqu'au meuble *F* et prend l'objet *Obj* se trouvant dessus.

En examinant la seconde décomposition de la méthode *GetObject*, nous constatons que la sélection de l'agent *AG1* est contrainte par la relation spatiale *graspable(AG1,Obj)==true* qui signifie que l'agent doit être capable de tenir l'objet, ce qui limite l'ensemble des agents qui peuvent satisfaire cette contrainte et de ce fait limite l'espace d'exploration. Nous notons également que la réalisation de tâche *GetObjectFromFurniture* est soumise à la contrainte *holdforgive(AG1, Obj)==true*, qui signifie qu'à la fin de la tâche l'agent *AG1* doit tenir l'objet *Obj* d'une manière à laisser des configurations de prise libres pour un autre agent. Cette contrainte va être propagée lors de la décomposition de la tâche *GetObjectFromFurniture* et elle va venir rajouter une contrainte supplémentaire sur sa liste de tâches. Dans cet exemple la contrainte n'aura aucune influence sur les deux premières tâches *ReachFurniture(Ag, F)* et *FreeOneHand(Ag)*, puisque la première est une tâche de déplacement qui ne peut pas influencer la position de prise et la deuxième est une tâche qui sert à débarrasser l'agent d'un objet qu'il a en main si nécessaire. C'est la troisième tâche *PickupOn(Ag, Obj, F)* qui doit assurer une configuration de prise pour l'agent *Ag* qui satisfait la contrainte *holdforgive(AG1, Obj)==true* et de ce fait elle assure que la réalisation de la tâche *GetObjectFromFurniture* satisfait la même

```
method GetObject(Agent Ag, Object Obj)
  {
    goal
    {
      Obj >> Ag.objects;
      Obj.owner == Ag;};

    //decomposition 1
    {
      preconditions
      { Obj.owner == NULL; };

      subtasks
      { 1:GetObjectFromFurniture(Ag, Obj, Obj.furniture); };
    }

    //decomposition 2
    {
      preconditions
      { Obj.owner == NULL; };

      subtasks
      {
        AG1 = SELECT(Agent, { AG1 != Ag; graspable(AG1,Obj)==true;)});
        1:GetObjectFromFurniture(AG1, Obj, Obj.furniture);
                                  constraint{holdforgive(AG1, Obj)==true;};
        2:RemitObjet(AG1, Ag, Obj) > 1;
      };
    }

    //decomposition 3
    {
      preconditions
      {
        Obj.owner != NULL;
        Obj.owner != Ag;};

      subtasks
      { 1:RemitObjet(Obj.owner, Ag, Obj); };
    }
  }
*****************************************************************************
method GetObjectFromFurniture(Agent Ag, Object Obj, Furniture F)
  {
    goal
    { Obj >> Ag.objects;
      Obj.owner == Ag;};

    //decomposition 1
    {
      preconditions
      { F.onTop == true;
        Obj >> F.objectsOn;
        Obj.owner == NULL;};

      subtasks
      { 1:ReachFurniture(Ag, F); constraint{reachable(Ag, Obj)==true;};
        2:FreeOneHand(Ag);constraint{reachable(Ag, Obj)==true;};
        3:PickupOn(Ag, Obj, F) > 1, > 2;
      };
    }
  }
```

FIGURE 5.5 – Exemple de déclaration de méthodes dans hybride-HATP

contrainte.

5.3.4 Contraintes

Comme nous l'avons vu précédemment, nous définissons une contrainte comme un état désiré pour une relation spatiale. Ces contraintes sont associées à des tâches (action et méthode). Si la tâche est une méthode, alors la contrainte est propagée sur toutes les décompositions de celle-ci. Sinon la contrainte va contraindre la réalisation de l'action, plus exactement, elle va contraindre certains effets de bord de l'action.

En analysant le modèle plus en profondeur, nous pouvons constater qu'il est sur-contraint, du fait, de la propagation des contraintes sur toutes les actions générées par une méthode. Une action va hériter d'un ensemble de contraintes qui n'ont rien à voir avec son but et sur lesquelles elle n'a aucun effet. Cela va rendre le problème insoluble pour le planificateur géométrique. Prenons un exemple où nous avons l'action prendre un objet sur un meuble *PickUp(Ag, F,Obj)*, supposons que suite à des décompositions et des propagations de certaines contraintes l'action se retrouve avec la contrainte sur la proximité entre agents *nextforGive(Ag,Ag1)==true*. Dans ce cas, le planificateur ne peut pas trouver de solution, puisque l'action n'a aucun effet sur la proximité entre agents.

Pour relaxer le problème nous avons fait le choix d'associer une liste d'action à chaque type de relation spatiale, que nous appelons *liste d'influence*. Cette liste va permettre de filtrer certaines contraintes associées à une action. Prenant comme exemple la relation *visible(Agent, (Agent OR Objet))* qui va lier deux agents ou bien un agent et un objet. À cette relation, nous allons associer la liste d'actions *(Move, PutdDown, Pickup)* sachant que dans le domaine nous avons aussi l'action (Give). Il est aisé de voir que les actions présentes dans la liste ont toutes des effets de bord qui peuvent influencer l'état de la relation *visible*.

5.3.5 Récapitulatif

La figure 5.6 récapitule les liaisons que nous avons établies entre le modèle de la planification de tâche et le modèle de la planification géométrique.

Comme nous l'avions énoncé précédemment, le but de ce couplage est de produire des plans faisables et qui assure au robot un comportement socialement acceptable en contrôlant la qualité sociale des plans produits au niveau symbolique, mais également au niveau géométrique. Pour pouvoir exercer un contrôle sur les solutions produites au niveau géométrique, nous avons introduit la notion d'action contrainte. Cela signifie que la réalisation de l'action au niveau géométrique passe inévitablement par la satisfaction des contraintes qui sont associées à celle-ci. Les contraintes nous permettent donc de conditionner les solutions produites au niveau géométrique. Nous avons également démontré que les contraintes associées aux actions dépendent

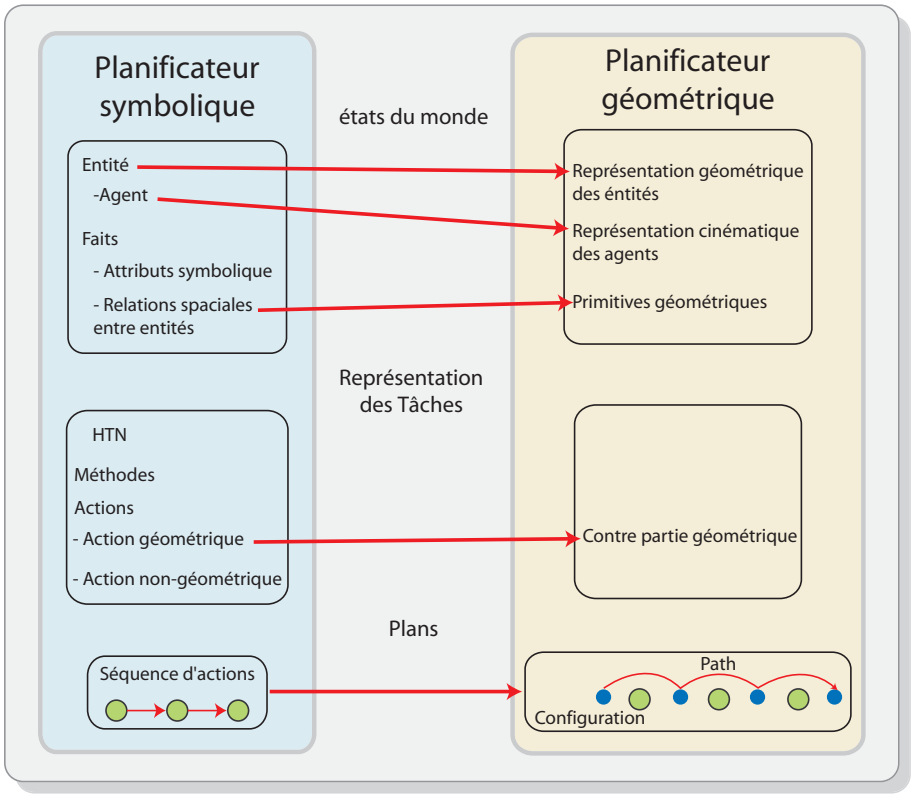


FIGURE 5.6 – liaison entre la représentation symbolique et la représentation géométrique.

du but de celle-ci. Ce qui implique que l'association rigide contraintes-action n'est pas l'approche la mieux adaptée pour une planification hybride. Nous avons proposé un procédé qui permet de sélectionner les contraintes à appliquer aux actions en fonction de la situation, et c'est cela qui fait l'originalité de notre approche.

5.4 Algorithme Hybride d'HATP

Avant de décrire le fonctionnement de l'algorithme, nous allons donner une description des structures de données que les deux planificateurs partagent.

5.4.1 Structure de données

Comme nous l'avons vu dans la section 5.3.1, la base de faits wb se divise en deux parties wb_{Symb} et wb_{Geom}. La première est la représentation symbolique du monde et des agents et la deuxième est la contrepartie géométrique. Dans ce qui suit, nous ne faisons pas de distinction entre base de faits symboliques et base de faits géométriques, car les deux planificateurs ont accès aux mêmes données à travers la même variable que nous appelons base de faits wb.

Une autre structure de données que les deux planificateurs partagent est la liste de tâches TL. Cette liste peut contenir des actions et des méthodes. Dans le cas d'une action, le modèle

présenté à la section 5.3.2 est complété par deux bases de faits géométriques wb_{Geom_b} et wb_{Geom_a}. La première représente la configuration du monde avant la réalisation de l'action et la seconde représente la configuration du monde après réalisation de l'action. Nous avons adopté cette structure pour pouvoir gérer plus aisément les backtracks au niveau géométrique.

5.4.2 Description de l'algorithme

L'algorithme 5.1 décrit le fonctionnement hybride de HATP. L'algorithme prend en entrée `wb` l'état du monde (symbolique et géométrique), `Arb` le problème et `D` le domaine de planification. Cet algorithme commence par l'initialisation d'une liste de tâches `T` (ligne 1), cette liste contient les feuilles de l'arbre `Arb`. On initialise également la projection temporelle `Prj` à un ensemble vide (ligne 2), le planificateur géométrique (ligne 3) et la structure qui héberge les points de backtracks (ligne 4).

On entre ensuite dans la boucle principale (lignes 5 à 34) jusqu'à ce que la limite de temps éventuelle soit atteinte ou jusqu'à ce que l'ensemble de l'arbre d'exploration ait été entièrement parcouru. La première étape de la boucle principale est de supprimer les tâches associées à des buts qui sont déjà satisfaits dans l'état actuel de la base de faits (lignes 6 à 9) afin de limiter l'exploration de l'espace de recherche. L'étape suivante consiste à vérifier si la liste de tâches `T` est vide ce qui indiquerait qu'un plan valide a été trouvé (lignes 10 à 13). La troisième étape consiste à appliquer une optimisation de type *branch-and-bound* (lignes 14 à 18) afin de limiter l'espace de recherche à explorer si un plan de référence a déjà été trouvé.

L'étape suivante consiste à déterminer quelle est la tâche à traiter en premier. On détermine l'ensemble T_0 contenant l'ensemble des tâches de `T` qui n'ont pas de prédécesseurs et dont les préconditions sont satisfaites (ligne 19). Si T_0 est différent de l'ensemble nul, alors il est créé dans l'arbre de recherche autant de branches que nécessaire selon la cardinalité de T_0 (ligne 21) puis une branche est choisie en sélectionnant une tâche `t` de T_0 (ligne 22). Dans le cas où T_0 est vide, on détermine un l'ensemble T_0 contenant l'ensemble des actions de `T`, qui n'ont pas de prédécesseurs et dont les préconditions ont été invalidées par des faits géométriques qui vont être stockés dans l'ensemble $SetGF$ (ligne 24). Une fois l'ensemble des faits géométriques impliqués établi, une liste des dernières actions qui ont modifié leurs états est déterminée. Pour chaque action sélectionnée le fait géométrique qui a causé l'échec est ajouté à l'ensemble des contraintes associées à l'action.(lignes 25 à 28). L'ensemble des actions mises en cause déterminé, l'ensemble est ordonné suivant l'ordre de la projection temporelle $Proj$ (ligne 29). La liste des actions ordonnées est transmise au planificateur géométrique (ligne 30) pour qu'il puisse backtracker sur ces actions et résoudre le problème avec des contraintes supplémentaires. Si la réponse du planificateur géométrique est négative, c'est-à-dire qu'il n'a pas trouvé de solution, un backtrack est opéré au niveau symbolique pour explorer de nouvelles branches (lignes 31 à 32). Dans le cas contraire, c'est-à-dire que le planificateur géométrique a réussi à trouver une solution. Le planificateur symbolique continue sa procédure normalement. Une fois la tâche sélectionnée, elle

Algorithme 5.1 : Algorithme de planification de HATP hybride

Données : wb, Arb
T ← $feuilles$(Arb)
Prj = ∅
InitGeometry(wb)
Point_BT =init_point_de_backtack(wb, T, Prj, Arb)
répéter
  T_0 ← {t ∈ T | $predecesseurs$(t) = ∅, wb satisfait $goal$(t)}
  pour ∀ t ∈ T_0 **faire**
    Retirer t de Arb
    Retirer t de T
  si T = ∅ **alors**
    P ← produire_plan(Arb, Prj)
    Ajouter P dans la pile des plans valides
    Explorer nouvelle branche dans Point_BT
  si $P_{REF} \neq \emptyset$ **alors**
    c ← comparaison_partielle(Arb, Prj, P_{REF})
    si $c > 0$ **alors**
      Couper la branche courante de l'arbre d'exploration
      Backtracker ;
  T_0 ← {t ∈ T | $predecesseurs$(t) = ∅, wb satisfait $precond$(t)}
  si $T_0 \neq \emptyset$ **alors**
    Point_BT ← N(T_0) crée branches dans l'arbre d'exploration
    Choisir une branche à explorer dans Point_BT et la tache t dans T_0 correspondante
  sinon
    T_0 ← {t ∈ T et t est une action | $predecesseurs$(t) = ∅, $precond$(t) a échouée à cause des faits géométrique SetGF }
    pour chaque *fait géométrique* $fg \in$ SetGF **faire**
      Sélectionner la dernière action a_i dans Prj qui a modifié le fait fg
      Ajouter fg dans les contraintes associées à l'action a_i
      Ajouter a_i dans la liste Bliste
    Ordonner la liste Bliste suivent l'ordre de Prj
    Backtrack géométrique sur les actions présentes dans Bliste
    si *Échec de la géométrie* **alors**
      Choisir une autre branche à explorer dans Point_BT
  Traitement_sur_tache(t, wb, Prj, T, Arb)
jusqu'à *limite de temps atteinte* ou *arbre d'exploration entièrement parcouru*

est traitée avec la procédure 5.2.

Algorithme 5.2 : Procédure de traitement des tâches

```
   Données : t, wb, Prj, T, Arb
 1 si t est une méthode alors
 2     M ← { m est une instance d'une méthode dans D, wb ne satisfait pas goal(m)}
 3     Dec ← {d ∈ decomposition(m), wb satisfait preconds(m)}
 4     si Dec =∅ alors
 5         Backtracker sur nouvelle branche
 6     sinon
 7         pour chaque élément d_i dans Dec faire
 8             Modifier T en remplaçant t par d_i
 9             Propager les contraintes de t dans d_i
10             Modifier Arb en ajoutant d_i comme décomposition de t
11             Dans Point_BT créer branche
12         Choisir une branche à explorer
13 sinon
14     A ← { a est une action instanciée dans D, wb satisfait precond(a)}
15     si t est une action géométrique alors
16         si Validation_géométrique( a, T, wb) alors
17             Modifier wb selon effects(a)
18             Appliquer a à Prj
19             Mettre à jour le coût
20         sinon
21             Couper la branche courante de l'arbre d'exploration dans Point_BT
22             Sélectionner une nouvelle branche à explorer
23     sinon
24         Modifier wb selon effects(a)
25         Appliquer a à Prj
26         Mettre à jour le coût
```

La procédure de traitement des tâches est décrite dans l'algorithme 5.2. Si la tâche sélectionnée `t` est une méthode, L'ensemble des décompositions valide est déterminé (ligne 3), si l'ensemble résultant `Dec` est vide, un backtrack est effectué pour explorer d'autres branches (lignes 4). Sinon, de nouvelles branches sont créées dans l'arbre d'exploration selon la cardinalité de `Dec` (lignes 7 à 11). Dans chaque branche, la liste de tâches `T` et l'arbre `Arb` sont mis à jour, et les contraintes sont propagées (lignes 8 et 10). La procédure se termine par la sélection d'une nouvelle branche d'exploration (ligne 12).

Si la tâche `t` est une action géométrique (ligne 15), sa validité au niveau géométrique est testée (ligne 16). Si elle est validée, les effets sont appliqués sur la base de faits, les coûts du plan sont mis à jour et l'action est ajoutée à la projection temporelle Prj (lignes 17 à 19). Si l'action n'est pas validée au niveau géométrique, un backtrack et effectué pour explorer une

autre branche d'exploration (ligne 22). Dans le cas ou l'action n'est pas géométrique, elle est traitée de la même façon que précédemment, c'est-à-dire que les effets sont appliqués sur la base de faits, les coûts sont mis à jour et l'action est ajoutée à la projection temporelle (lignes 24 à 26).

5.4.3 Gestion des backtracks

Pour la gestion des backtracks nous avons trois cas qui peuvent être envisagés :

Backtrack symbolique Le backtrack symbolique, comme son nom l'indique survient, au niveau symbolique. Il se produit dans quatre cas possibles. Dans tous ces cas le planificateur rejette la branche en cours pour explorer une nouvelle branche :

1. Échec sur la réalisation d'une action, lorsque les préconditions ne sont pas valides.
2. Échec sur la décomposition d'une méthode, c'est-à-dire qu'aucune des décompositions d'une tâche ne peut être réalisée.
3. Le coût du plan développé dépasse le coût du plan de référence.
4. Échec de la plannification géométrique, c'est-à-dire que le planificateur ne peut pas valider une action.

Backtrack géométrique Le backtrack géométrique ne se produit qu'au niveau des actions. Du fait que le planificateur géométrique fait une instanciation incrémentale sur :
– Les trajectoires, essentiellement position initiale et finale des trajectoires.
– Les emplacements, le choix d'un emplacement pour poser un objet peut contraindre le choix quand on souhaite en poser un autre.

De ce fait, nous concéderons que si une action a_i n'est pas réalisable au niveau géométrique, la première cause est la configuration initiale. Cependant, cette configuration initiale est plus ou moins affectée par les actions qui la précèdent. Ainsi, le backtrack géométrique se fait dans l'ordre inverse de la séquence d'actions qui précède l'échec. C'est-à-dire, si a_i est l'action causant l'échec, le planificateur va backtracker sur a_{i-1} puis a_{i-2} jusqu'à a_0. Le backtrack s'arrête une fois qu'il a trouvé un ensemble de configurations qui satisfont la séquence d'actions et qui permettent de réaliser l'action a_i, auquel cas il ny a pas de solution.

Backtrack géométrique guidé par le symbolique Ce dernier cas est un cas particulier, car c'est une situation où il y a un échec au niveau symbolique, mais qui va être traité au niveau géométrique. Considérons l'exemple de la figure 5.7. Nous avons la séquence d'action $Pickup- > Move- > Give$. Les actions *Pickup* et *Move* ont été validées au niveau symbolique et au niveau géométrique, l'action *Give* n'est pas encore validée au niveau symbolique. Nous concéderons aussi qu'il n'y a qu'un seul fait géométrique qui est ***HoldeForGive(Jido,Bottle)***. Ce fait est consommé par l'action *Give*, il est modifié par effet de bord par l'action *Pickup*.

Dans ce cas de figure, l'action *Give* n'est pas validée à cause d'un fait symbolique pur. Le planificateur fait un backtrack symbolique et va explorer une autre branche. Si la cause est un fait géométrique, le planificateur symbolique donne un ordre de backtrack au planificateur géométrique en indiquant sur quelle action il doit backtracker. Cette action va être modifiée par le planificateur symbolique. Plus précisément, le planificateur va rajouter des contraintes sur celle-ci. Dans l'exemple, le planificateur symbolique donne un ordre de backtrack sur l'action *Pickup* au planificateur géométrique, puisque c'est la dernière action qui a modifiée l'état ce fait HoldeForGive(Jido,Bottle). Sur cette action est rajoutée la contrainte HoldeForGive(Jido,Bottle)==true. Si le planificateur géométrique réussit à trouver une solution alors un plan est trouvé, sinon un backtrack symbolique est opéré.

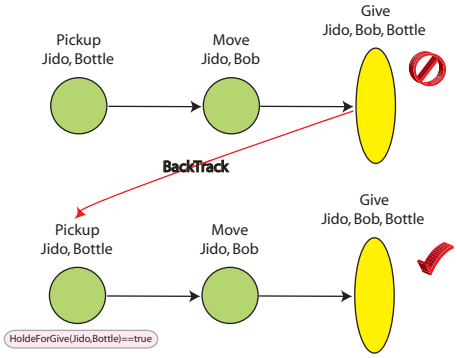


FIGURE 5.7 – Exemple d'un backtrack au niveau géométrique guidé par le planificateur symbolique

5.5 Exemple et fonctionnement

Pour démontrer les capacités et les limites du planificateur hybride, nous allons illustrer son fonctionnement par deux exemples d'interaction homme-robot.

5.5.1 Scénario Interaction homme-robot

Scénario 1

Ce scénario représenté par la figure 5.8 est un scénario d'interaction face-à-face. Deux agents sont assis autour d'une table : un humain et un robot. Plusieurs objets sont posés sur la table :une tasse à café, une bouteille et une boite contenant du café, ainsi que différents objets (des boites) considérés comme des obstacles pour la manipulation et la visibilité. Le but du robot est d'aider l'humain à préparer du café.

Sur la figure 5.8, l'espace de travail est divisé en plusieurs zones colorées. Les zones en bleu et en jaune correspondent à l'accessibilité respective de la main droite et de la main gauche d'un agent, la couleur verte représente les zones accessibles par les deux mains d'un même agent. Ces

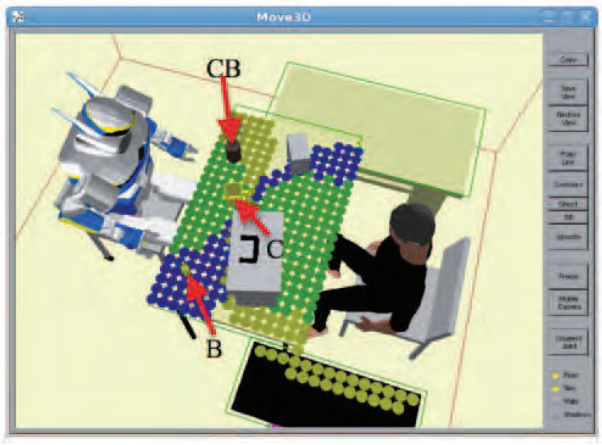


FIGURE 5.8 – Scénario 1 : Modèle 3D de l'environnement où évoluent l'humain et le robot (Image générée par le planificateur géométrique).

zones sont calculées par le planificateur géométrique.

Les actions basiques que le planificateur symbolique a dans son domaine et qui ont un équivalent géométrique sont :

– **Pickup(Agent Ag , Objet Obj, Meuble F)** : Action de prendre un objet **Obj** se trouvant sur un meuble **F** par un agent **Ag**.

– **Putdown(Agent Ag , Objet Obj, Meuble F)** : Action de déposer un objet **Obj** sur un meuble **F** par un agent **Ag**.

– **Give(Agent Ag1 , Agent Ag2, Objet Obj)** : Action de transmettre un objet **Obj** de l'agent **Ag1** à l'agent **Ag2**.

Les tâches de haut niveau utilisées sont :

– **GetObject(Agent Ag, Objet Obj)** : Méthode qui permet à l'agent **Ag** d'obtenir l'objet **Obj**

– **GetObjectFromeFurniture(Agent Ag, Objet Obj, Meuble F)** : Méthode qui permet à l'agent **Ag** d'obtenir l'objet **Obj** qui se trouve sur le meuble **F**

– **MakeCoffee(Agent Ag)** : Méthode qui permet à l'agent **Ag** de faire du café.

– **TransmiteObject(Agent Ag1, Agent Ag2, Objet Obj)** : Méthode qui permet de faire passer un objet **Obj** de l'agent **Ag1** à l'agent **Ag2**.

Le but transmis au planificateur est décrit par **MakeCoffee(H1)**. Ce but se traduit par le fait de trouver une séquence d'action qui permet à l'humain **H1** de faire du café. Le planificateur commence par tester si le but est déjà réalisé. Si c'est le cas, la planification est arrêtée. Sinon nous cherchons les décompositions valides appartenant à la méthode **MakeCoffee**. Dans ce cas il n'y en a qu'une seule, l'humain doit obtenir une tasse, une bouteille d'eau et la boite à café. Comme la bouteille et la tasse sont inaccessibles par l'humain, c'est le robot qui va s'en charger avec la séquence **GetObjectFromeFurniture** → **TransmitObject**.

Toute la faisabilité du plan va se jouer au niveau des tâches **TransmitObject** puisque la tâche va se décomposer en **Putdown** → **Pickup**, et le problème va se poser au niveau du choix de la position de pose des objets. Comme nous pouvons le voir sur la figure 5.8, la zone d'accessibilité commune pour les deux agents est très réduite et ne peut supporter qu'un nombre limité d'objets à la fois.

Pour voir comment ce problème va être solutionné par le planificateur, nous avons sélectionné une des solutions (figure 5.9) que le planificateur a produites. Les cercles en vert représentent les actions du robot, les cercles en jaune représentent les actions de l'humain, les flèches bleues représentent des backtraks géométriques guidés par le planificateur symbolique, les flèches rouges représentent des relations de précédence, et les flèches noires représentent des backtracks géométriques purs. Les rectangles bleus regroupent les contraintes imposées sur l'action qui leur est associée.

Nous pouvons observer sur la figure 5.9 qu'au niveau de l'action (4), nous avons un backtrack géométrique. Le problème est que le choix de la position de pose de l'action (2) a rendu impossible la pose de la bouteille dans l'action (4). Pour cela le planificateur backtrack sur les choix qu'il a fait auparavant pour trouver une solution. Du fait de l'instanciation incrémentale du planificateur géométrique, le backtrack ne se fait pas directement sur l'action (2), il doit passer inévitablement par l'action (3) pour arriver à l'action (2).

Un autre backtrack a lieu au niveau de l'action (6). Cette action est non valide au niveau symbolique à cause de l'état d'accessibilité de la tasse. L'action (2) devait assurer que la position de pose de la tasse soit accessible à l'humain, mais le choix effectué au niveau de l'action (4) pour poser la bouteille a changé l'état d'accessibilité de la tasse. Dans ce cas, un backtrack géométrique au niveau de l'action (4) est fait en rajoutant la contrainte que la tasse reste accessible pour l'humain.

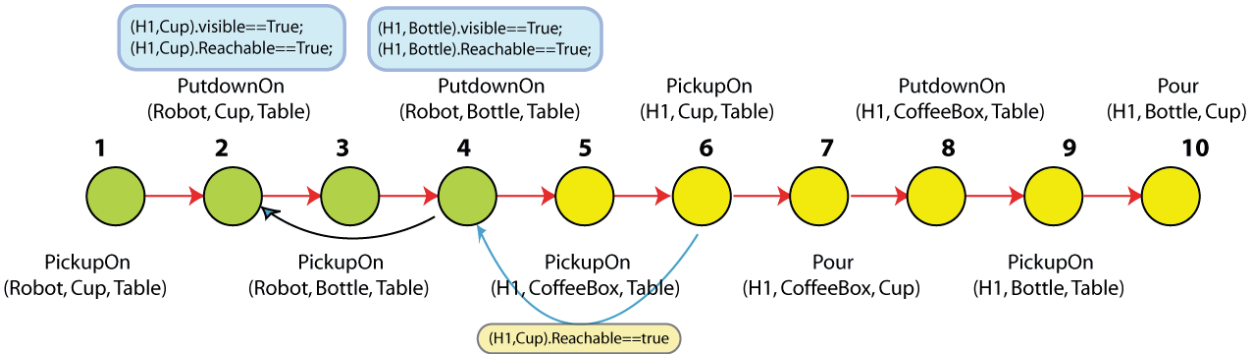


FIGURE 5.9 – Séquence d'actions produite par le planificateur hybride, pour le scénario 1.

Nous pouvons constater que le modèle présente un problème. La figure 5.10, illustre le même plan que celui de la figure 5.9 mais après parallélisation. Nous remarquons que l'action (5) n'a

pas de dépendance et que l'action (6) a des contraintes de précédence avec les actions (5) et (2). Dans le paragraphe précédant nous avons souligné qu'il a un backtrack au niveau de l'action (4) qui est causé par les effets de bord de l'action (2), mais si nous considérons le plan après parallélisation, nous pouvons constater sur le plan que l'action (5) est totalement indépendante, c'est-à-dire qu'elle peut être réalisée à tout moment. Alors que si l'action (5) était réalisée avant l'action (4), cela aurait résolu le problème du backtrack géométrique. Nous pensons qu'une des solutions qui pourrait limiter le nombre backtrack géométrique est de paralléliser le plan durant la phase planification, ce qui nous aidera aussi à mieux détecter les plans semblables, et de ce fait, limiter l'espace de recherche.

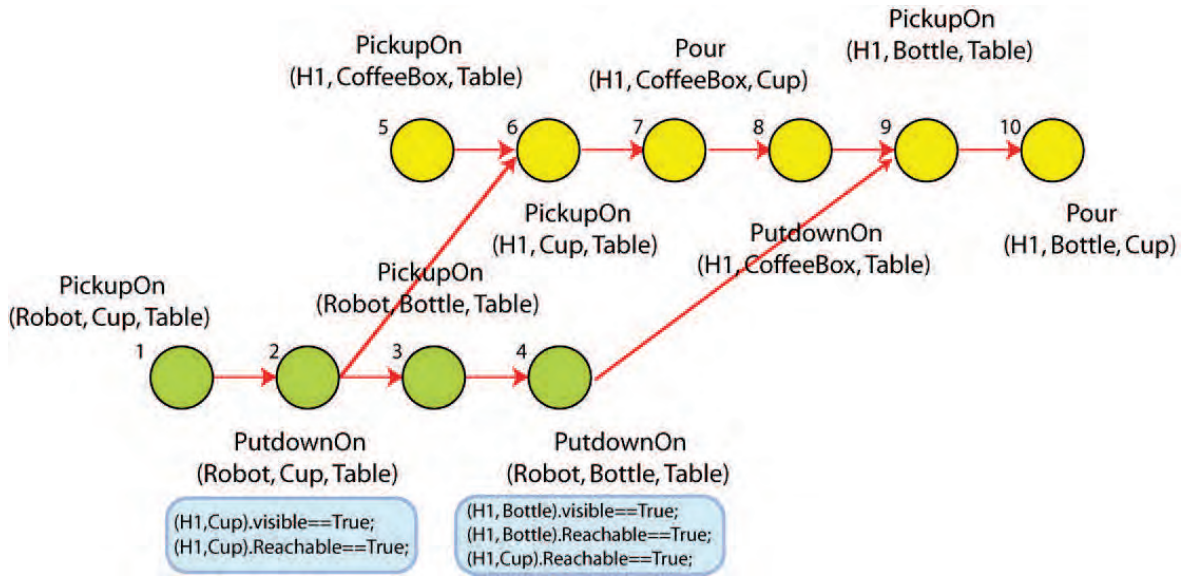


FIGURE 5.10 – Plan parallélisé produit par le planificateur hybride, pour le scénario 1

Scénario 2

Dans ce scénario, nous allons montrer l'intérêt de l'intégration du planificateur géométrique avec un planificateur symbolique pour l'interaction homme robot. Nous allons comparer deux plans identiques en termes de but et de nombre d'actions. Le premier est un plan produit par le planificateur symbolique et le deuxième est produit par le planificateur hybride.

Nous supposons qu'il y a deux agents, un robot *Jido* et un humain. Le but transmis au planificateur est simple et consiste à apporter un objet pour l'humain et le déposer sur une table. Les actions qui vont être utilisées sont les mêmes que celles décrites dans l'exemple précédent, c'est-à-dire *Move*, *Pickup* et *PutDown*. Les faits géométriques utilisés sont :

– *distance_security(Agent, Agent)* : distance minimale entre deux agents qui garantit la préservation de leur espace vital et leur sécurité au niveau géométrique.

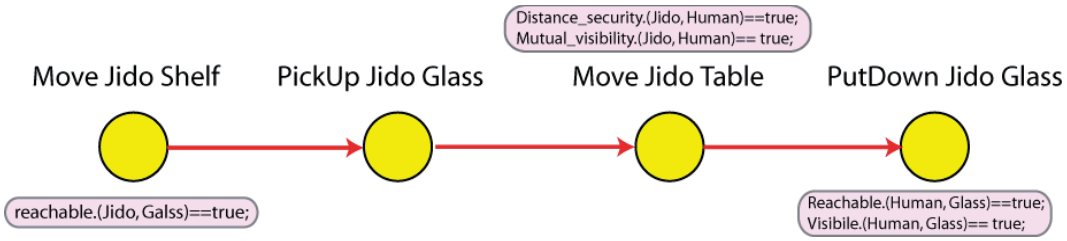
– *visibility(Agent, (Agent OR Objet))* : visibilité entre un agent et un objet ou bien entre

un agent et un autre agent.

– *Mutual_visibility(Agent, Agent)* : visibilité mutuelle entre deux agents.

La figure 5.11 illustre deux plans. Le premier est produit par le planificateur HATP classique et le second est produit par hybride-HATP. Nous pouvons déjà constater que c'est le même plan, mais complété par les contraintes géométriques qui visent à rendre l'interaction plus agréable pour l'humain.

(a) Plan produit par HATP classique



(b) Plan produit par HATP hybride

FIGURE 5.11 – Exemple 2 : comparaison entre deux plans HATP produits avec et sans planificateur géométrique

En analysant les plans action par action :

– Pour l'action *Move*, l'ajout de la contrainte **(Jido, Galsse).reachable == true** assure que le robot va choisir des configurations qui respectent la contrainte, et de ce fait assure le succès d'une prochaine action au niveau géométrique.

– L'action *Pickup* est la même dans les deux plans. À la seule différence que dans le plan (b) avec géométrie, le robot est assuré d'être dans la bonne configuration pour réaliser sa tâche.

– Pour le second *Move*, les contraintes associées à l'action vont imposer au planificateur géométrique de choisir des configurations où il ne met pas l'humain en danger et où il est toujours visible par celui-ci. Ce qui oblige le robot à adopter un comportement physique socialement acceptable.

– Pour l'action *PutDown*, la contrainte va obliger le robot à se mettre à la place de l'humain en adoptant sa perspective (Perspective Placement [Luis F. Marin-Urias 08]) pour pouvoir choisir une position où poser l'objet qui soit atteignable par celui-ci. Cela va assurer l'action future de l'humain et son confort.

L'utilisation des contraintes dans le cadre de l'interaction homme-robot influence le comportement du robot pour générer des comportements qui sont socialement acceptables. Les contraintes utilisées ont été empruntées au générateur de configuration PSP (présenté dans la section 5.2.1). Nous pouvons constater que l'utilisation de ces contraintes au niveau symbolique est beaucoup plus avantageuse, puisque le planificateur peut adapter la contrainte utilisée en fonction de la situation et en fonction de l'agent avec lequel le robot va interagir, et cela de façon complètement générique.

L'utilisation des contraintes et des règles sociales permet d'avoir des comportements acceptables pour l'interaction homme robot, mais cela reste insuffisant, car il reste un élément important qui est le mouvement du robot. Le déplacement d'un robot en présence d'un humain est fondamental dans le cadre de l'interaction, comme il a été démontré dans [Sisbot 05], [Sisbot 10]. En effet pour pouvoir partager l'espace avec un humain, il faut que les mouvements du robot soient sûrs, intelligibles et agréables pour l'humain, ce que le modèle n'assure pas actuellement. Puisqu'il ne contrôle que les configurations, c'est-à-dire les états initiaux et finaux, mais qu'il n'a aucun contrôle sur les états intermédiaires. Ce qui implique, aucun contrôle sur la trajectoire qui relie deux configurations.

Actuellement, les planificateurs de déplacement ou de mouvement dédiés à l'interaction assurent deux qualités, la sûreté et l'intelligibilité du mouvement, mais cela n'est pas suffisant, il est nécessaire que les mouvements du robot soient agréables pour l'humain. Ce résultat ne peut être obtenu qu'avec un couplage avec un planificateur symbolique.

Supposons que dans une pièce, nous avons un humain qui regarde la télévision et un robot qui fait du ménage. Dans cette situation, les mouvements du robot doivent être sûres et intelligibles, mais également agréables. Cela se traduit par des déplacements qui gênent le moins possible l'activité de l'humain ; par exemple, il doit éviter de bloquer la visibilité de l'humain qui regarde la télévision. Ce petit exemple montre bien la nécessité d'un mouvement agréable pour l'interaction homme-robot.

5.5.2 Scénario pour la manipulation d'objet

Nous concéderons l'exemple de la figure 5.12. Dans cet exemple, nous avons deux cubes C_1 et C_2. Ces cubes sont attachés avec une ficelle, ce qui crée une contrainte géométrique qui est **Max_distance(C_1,C_2)** qui correspond à $(distance(C_1, C_2) < longueur\_ficelle)$. Les deux cubes sont posés sur une table, sur deux lignes parallèles L_1 et L_2. L'écart entre les lignes est inférieur à la longueur de la ficelle. Le but du robot est de déplacer les deux cubes de leurs positions actuelles Pi_1 et Pi_2 aux positions Pf_1 et Pf_2, avec la contrainte que le robot ne peut déplacer qu'un cube à la fois, et que quand un objet est posé, il doit toujours être reposé sur la ligne où il se trouvait.

Dans ce scénario, nous allons réutiliser les mêmes actions atomiques que celles définies précédemment dans le scénario 1, en rajoutant au domaine une nouvelle méthode qui est

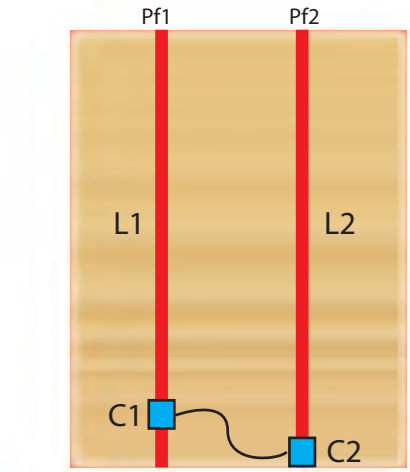


FIGURE 5.12 – Illustration du scénario 3

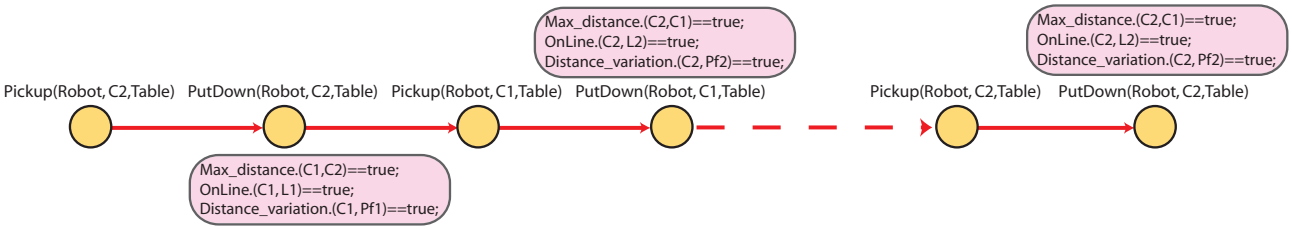


FIGURE 5.13 – Plan produit par HATP hybride pour le scénario 3

BringObjectCloseTo(Agent Ag, Object Obj1, Object Obj2, Furniture F). C'est une méthode récursive qui ne s'arrête qu'une fois que l'objet **Obj** est sur la position **Pos**. Elle n'a qu'une seule décomposition possible et qui est valide à tout moment. La séquence de tâches qui la compose est ($Pickup \rightarrow Putdowwn \rightarrow BringObjectCloseTo$).

Les contraintes utilisées dans cet exemple sont :

– **Max_distance(Objet,Objet)** : elle indique que la distance entre deux objets doit être inférieure à une certaine valeur. Dans le cas du scénario, cette valeur est égale à la longueur de la ficelle.

– **OnLine(Objet, ligne)** : elle définit le fait que l'objet se trouve ou non sur une certaine ligne.

– **Distance_variation(Objet, Position)** : elle renseigne sur la variation de la distance entre un objet et une position.

La figure 5.13 illustre le plan qui va être produit par le planificateur. Nous concéderons que le plan n'est qu'une succession de **Pickup** et **Putdown**, Le planificateur symbolique n'a aucun contrôle sur la taille de la séquence d'actions ni sur les choix de position de pose. Les seules contraintes qu'il impose sont celles associées aux actions, et les précédences entre les

actions. Si nous modélisions le même problème avec un planificateur purement symbolique, nous ne pourrions pas atteindre ce degré de généricité. Puisque nous serions obligé d'ajouter au planificateur une fonction qui calcule le pas et qui tient compte de la longueur de la ficelle, les longueurs des lignes et l'écartement entre les lignes, en plus les actions **Pickup** et **Putdown** qui seront spécifiques à ce problème.

5.6 Conclusion

Dans ce chapitre nous avons présenté notre approche pour un planificateur hybride, qui combine un planificateur symbolique basé sur un modèle HTN et un planificateur géométrique qui peut fournir des primitives de haut niveau. Cette combinaison n'est pas une nouveauté, puisqu'il existe plusieurs travaux qui traitent de la problématique comme nous avons pu le constater dans la section 5.1. L'originalité de notre approche réside au niveau de la modélisation des tâches et des contraintes, la gestion des backtracks et son utilisation pour l'interaction homme-robot. En effet, nous avons pu constater que, dans notre modèle de tâches, il n'y a pas d'association rigide entre contrainte et action, et nous avons souligné le fait que cette association dépend du but de l'action. Nous avons montré le mécanisme de propagation de contrainte qui permet de gérer ces associations entre contrainte et action. Nous avons également montré que le système a une gestion efficace des backtracks, ce qui permet de limiter leur nombre. Nous avons illustré le fonctionnement sur plusieurs exemples où nous avons pu observer l'intérêt d'un tel couplage entre planification symbolique et planification géométrique.

Nous avons également noté que le modèle présente quelques défauts. Nous avons constaté que le traitement séquentiel des actions durant la planification génère des backtracks inutiles. Cela peut être évité en parallélisant le plan durant la phase de planification ou en mettant en place une procédure qui fait un ordonnancement efficace.

De même, nous avons pu constater que le modèle présente un défaut dans le contrôle du comportement du robot pour l'interaction avec des humains. En effet, le modèle est conçu pour le contrôle des configurations (robot et objet) et la gestion de certains états géométriques, mais un des éléments importants dans l'interaction homme-robot est le mouvement et le déplacement du robot. Il est nécessaire de contrôler les mouvements du robot pour arriver à des comportements et des déplacements agréables pour l'humain.

6

Expérimentations et intégration du planificateur

Dans le chapitre 3, nous avons détaillé le modèle de planification HATP pour la production de plan socialement acceptable pour les humains coopérant avec des robots. Le chapitre 4 traite de l'enrichissement du modèle HATP par la gestion des croyances multiples. Nous avons également pu constater que pour les deux modèles, l'algorithme de planification principal est le même. Cela nous permet d'arriver à une conclusion toute simple qui est que les performances temporelles des deux modèles sont sensiblement les mêmes, puisque dans le modèle avec gestion des croyances, celle-ci se fait avec la structure HTN et non par un algorithme spécifique.

Dans ce chapitre nous allons tester l'influence des règles sociales sur le temps de calcul des plans et également évaluer les performances de l'heuristique de guidage. Ensuite, nous allons illustrer quelques exemples de l'utilisation de HATP sur un robot réel pour l'interaction homme-robot, et son intégration dans une architecture pour la gestion de mission multi-robots.

6.1 Choix technologiques pour l'utilisation en temps réel

Le planificateur HATP a été développé pour être utilisé en ligne sur un robot. Cette caractéristique impose des contraintes de fonctionnement en temps réel, notamment en termes

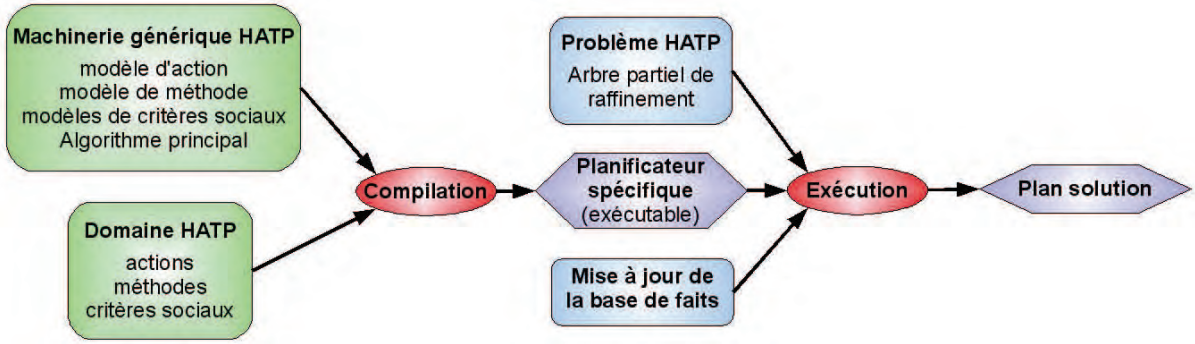


FIGURE 6.1 – Principe de compilation de HATP

de temps de calcul et d'utilisation de mémoire. Pour répondre efficacement à ces contraintes, certains choix technologiques sont nécessaires.

6.1.1 Principe de compilation

On peut classer les planificateurs délibératifs en deux familles distinctes :

Les planificateurs génériques qui utilisent le même algorithme quelque soit le domaine de planification.

Les planificateurs dédiés qui utilisent des domaines spécifiques adaptés à leur algorithmique.

Le tableau suivant résume les avantages et les inconvénients de chacune des deux approches [Ilghami 03] :

	Planificateurs dédiés	**Planificateurs génériques**
Gamme d'applications	Limitée	Large
Rapidité d'exécution	Très bonne	Moyenne
Utilisation de mémoire	Très bonne	Importante

Les planificateurs dédiés ont d'excellentes performances pour une exécution en temps réel, mais ne ciblent qu'un nombre limité d'applications. Afin de permettre l'utilisation de planificateurs génériques pour des applications concrètes, certains travaux ([Ilghami 03]) se sont focalisés sur la génération de planificateurs dédiés à partir d'un processus de planification générique. Ce principe a été repris dans le planificateur HATP. Comme l'illustre la figure 6.1, la description du domaine HATP est utilisée pour générer de manière automatique le code C++ spécifique. Ce code est ensuite compilé pour générer un exécutable intégrant directement le domaine de planification et utilisant les bibliothèques génériques du moteur principal d'HATP. Lorsque l'exécutable généré est lancé, il peut recevoir des ordres pour la mise à jour de la base de faits, et des requêtes contenant les problèmes de planification.

6.1.2 Structure multi-thread

Afin de réduire le temps de calcul processus de planification, HATP a été décomposé en trois threads afin de paralléliser un maximum de calculs. La figure 6.2 illustre la structure multi-

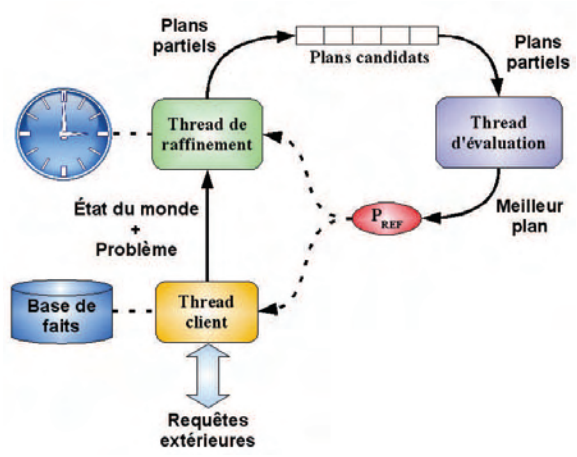


FIGURE 6.2 – Structure multi-thread de HATP

threads de HATP.

Le thread de raffinement permet d'explorer l'espace de recherche. Pour chaque plan en cours d'élaboration il construit l'arbre de raffinement et une ébauche de la projection temporelle. Il accueille l'heuristique et la procédure d'optimisation de type *branch-and-bound*. À chaque fois qu'un plan valide est trouvé par le thread de raffinement, il est stocké dans la pile des plans candidats. Dans le thread d'évaluation s'effectue la parallélisation des plans afin de pouvoir les comparer au plan de référence P_{REF}. Lorsqu'un meilleur plan est trouvé, il devient le nouveau plan de référence courant. Le thread de raffinement est connecté à un chronomètre permettant de limiter le temps de planification. Ainsi, HATP peut recevoir des requêtes lui permettant de rechercher le meilleur plan en un temps limité. Cette dernière spécificité est une caractéristique importante pour un planificateur amené à fonctionner en ligne sur un robot. Enfin, le thread client permet à HATP de communiquer avec l'extérieur. Il a en charge (1) le maintien de la base de faits, (2) la transcription du problème reçu en une structure informatique exploitable par le thread de raffinement, et (3) la transcription du meilleur plan trouvé en une structure informatique exploitable par le module logiciel en charge de contrôler son exécution.

6.2 Évaluation du modèle HATP

Dans cette section, nous allons tester l'influence de l'introduction des règles sociales sur le temps de calcul du planificateur, et également voir les améliorations qu'apporte l'heuristique pour la recherche de plan. Nous n'allons pas comparer les performances du planificateur à d'autres planificateurs existants et qui ont déjà fait leur preuve, puisque cela a déjà été fait dans les travaux de recherche de doctorat de Montreuil [Montreuil 08]. Tous les tests ont été effectués sur une machine avec un système d'exploitation linux, un processeur Intel core duo 1.87 GHz et 1Go de mémoire vive.

Pour les deux tests nous allons utiliser le même scénario. La figure 6.3 illustre la situation de départ du scénario. Il implique deux agents, un humain *Tom* qui se trouve sur le sofa et

un robot *Jido* qui est à côté de la porte. Dans ce scénario, l'humain *Tom* souhaite boire de l'eau. Pour cela, il lui faut obtenir un verre *Glass* qui se trouve dans le placard et la bouteille d'eau *Bottle*, qui se trouve sur la table. Ensuite, il retourne au sofa (La description de ce but est illustrée par la figure 6.4 en langage HATP). L'ensemble des actions de bas niveau que les deux agents peuvent réaliser sont : se déplacer, ouvrir/fermer la porte d'un meuble, prendre/déposer un objet dans/sur un meuble et donner un objet à un autre agent.

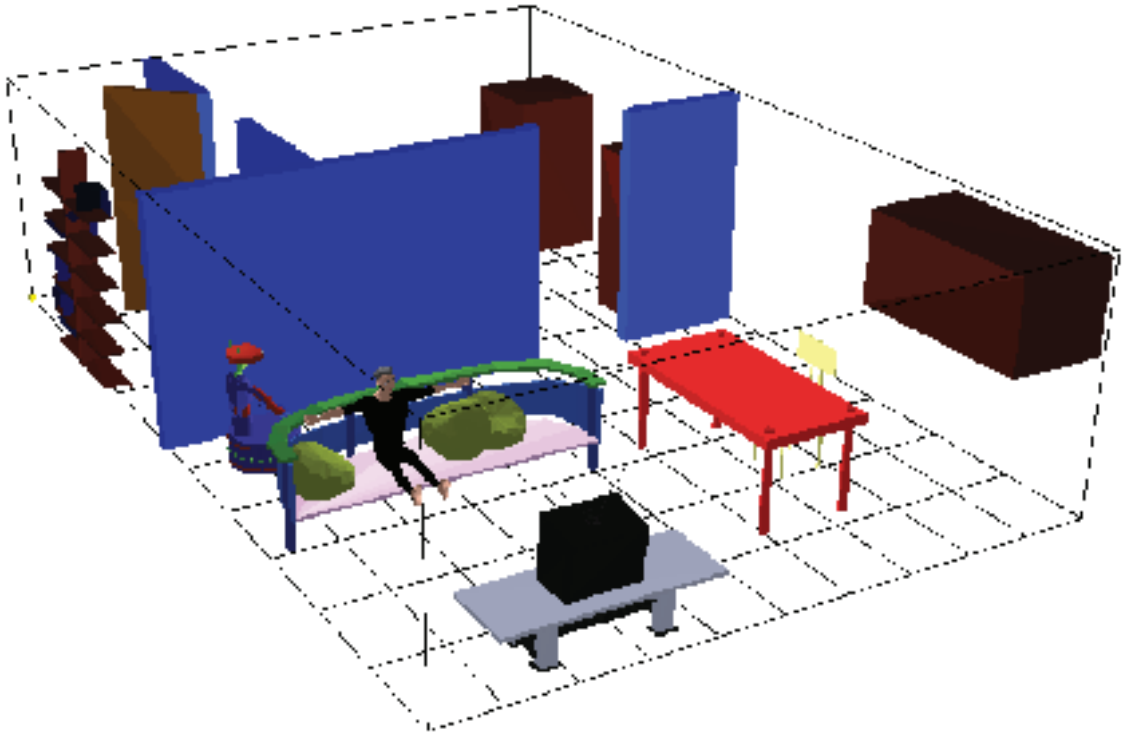


FIGURE 6.3 – Situation de départ du scénario test

6.2.1 Influence des règles sociales

La figure 6.5.(a) illustre le plan que produit HATP en l'absence de règle. On peut constater que le plan est tout à fait correct mais qu'il présente quelques inconvénients pour l'humain, puisque c'est lui qui fait le plus gros du travail. Il y a aussi quelques défauts de comportement ou d'incohérence comme : l'humain qui ouvre la porte du placard pour que le robot puisse prendre la bouteille, le robot qui va poser la bouteille sur le sofa que l'humain récupère à cet endroit. Si on introduit quelques règles comme :

– Etat indésirable 1 : on ne souhaite pas voir l'objet posé sur des meubles type sofa.

– Etat indésirable 2 : éviter de laisser la porte des meubles ouverte.

– Séquence indésirable 1 : on ne souhaite pas voir apparaître dans le plan une séquence telle que *déposer* suivie immédiatement par l'action *prendre*, qui soit faite par deux agents différents.

– Séquence indésirable 2 : on ne souhaite pas que l'ouverture et la fermeture du placard

```
// Description des noeuds
node:1
{
 1:GetObject(Thierry, Glass);
 2:GetObject(Tom, Bottle);
 3:ReachFurniture(Tom, Sofa) > 1, > 2;
}
```

FIGURE 6.4 – Description du problème pour le scénario test

soient faites par deux agents différents.

– On active aussi la règle balance d'effort qui explicite que le robot doit travailler plus que l'humain.

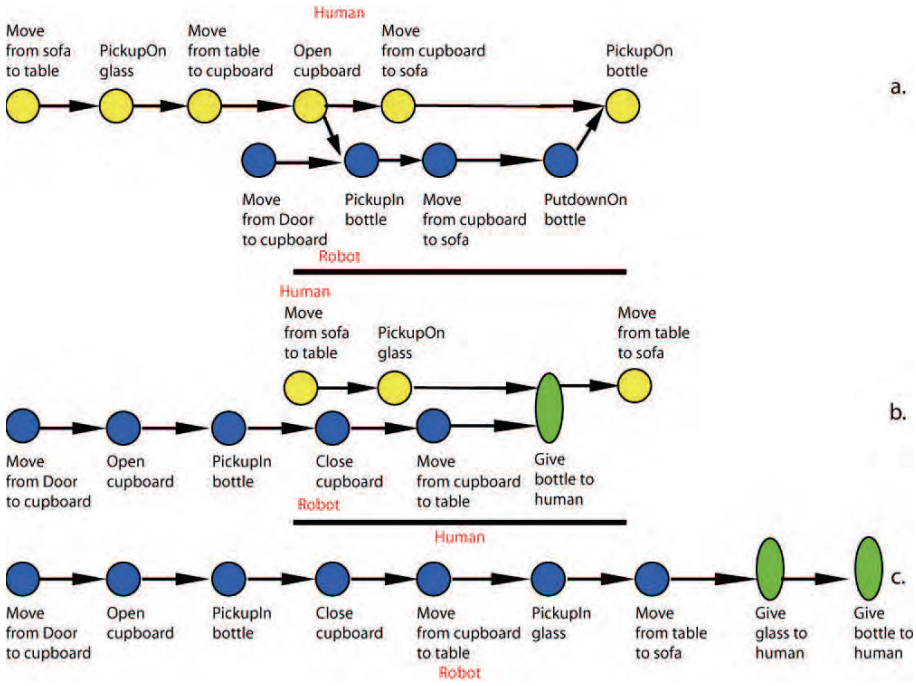


FIGURE 6.5 – Influence des règles sociales sur la qualité des plans produits

L'activation de toutes ces règles va produire pour le même problème le plan illustré par la figure 6.5.b. On peut voir que l'introduction de ces règles a complètement épuré le plan des imperfections que nous avons citées auparavant. Cependant, le plan n'est pas parfait et présente toujours quelques défauts. En effet, on peut observer qu'il y a toujours l'intervention de l'humain, qui se déplace du sofa vers la table pour aller chercher le verre et récupérer la bouteille chez Jido pour ensuite revenir vers le sofa. On constate que l'humain doit attendre que le robot ait fini ses tâches pour pouvoir récupérer la bouteille. Ce qui introduit un temps d'attente qui peut être gênant pour l'humain. Si nous introduisons les règles *liens croisés* et *temps morts*, le plan produit et illustré par la figure 6.5.c ne présente aucun défaut, respecte toutes les règles sociales et de ce fait, il est agréable pour l'humain.

```
// Description des noeuds
node:1
{
 1:GetObject(Tom, Glass);
 2:GetObject(Tom, Bottle);
 3:ReachFurniture(Tom, Sofa) > 1, > 2;
}

node:2
{
 1:GetObjectFromFurniture
        (Jido, Glass, Table);
 2:TransmitObject(Jido, Tom, Glass) > 1;
}

node:3
{
 1:GetObjectFromFurniture
        (Tom, Bottle, Table1);
}

// Description des liens hiérarchiques
1:1: decomposition = node 2;
1:2: decomposition = node 3;
```

FIGURE 6.6 – Description du plan partiel PP1

```
// Description des noeuds
node:1
{
 1:GetObject(Tom, Glass);
 2:GetObject(Tom, Bottle);
 3:ReachFurniture(Tom, Sofa) > 1, > 2;
}

node:2
{
 1:GetObjectFromFurniture
        (Jido, Glass, Cupboard);
 2:TransmitObject(Jido, Tom, Glass) > 1;
}

node:3
{
 1:GetObjectFromFurniture
        (Tom,Bottle, Table);
}
node:4
{
 1:ReachFurniture(Jido, Cupboard);
 2:SomeoneOpenFurniture(Cupboard);
 3:PickupIn(Jido, Glass, Cupboard) > 1, > 2;
 4:SomeoneCloseFurniture(Cupboard) > 3;
}
// Description des liens hiérarchiques
1:1: decomposition = node 2;
1:2: decomposition = node 3;
2:1: decomposition = node 4;
```

FIGURE 6.7 – Description du plan partiel PP2

Maintenant nous allons mesurer l'influence de l'introduction des règles sur le temps de calcul du planificateur. Nous allons prendre le même scénario et le même problème, mais également deux plans partiels de ce dernier illustré par les figures 6.6 et 6.7, qu'on appellera PP1 et PP2 et P pour le problème initial. On impose à PP1 une décomposition aux tâches (1) et (2) du nœud père (1). La tâche (1) doit se décomposer d'après la description du nœud (2), et la tâche (2) se décompose d'après le nœud (3). De la même façon, pour PP2 on impose les mêmes décompositions que celle de PP1 et on ajoute la contrainte que la tâche (1) du nœud (2) se décompose conformément au nœud (3).

Les résultats obtenus sont illustrés par le graphique de la figure 6.8. Ce graphique décrit l'évolution du temps de calcul des plans en fonction de l'introduction des règles. Nous avons fait le test sans introduction de règles, pour les règles *états indésirables* et *séquences indésirables*, et avec toutes les règles en même temps. Les règles utilisées sont celles citées dans l'exemple précédent.

Nous pouvons constater que l'introduction des règles augmente le temps de calcul en moyenne de 25% et que ce temps de calcul augmente linéairement avec la complexité du problème. Cela s'explique par le fait qu'un sous-ensemble de règles est évalué après la phase de raffinement, comme nous l'avions noté précédemment dans l'algorithme 3.1. Cela a pour conséquence l'augmentation du temps de calcul global. Nous constatons cela en comparant dans le graphique le temps de calcul pour les règles *séquences indésirables* qui sont évaluées en ligne,

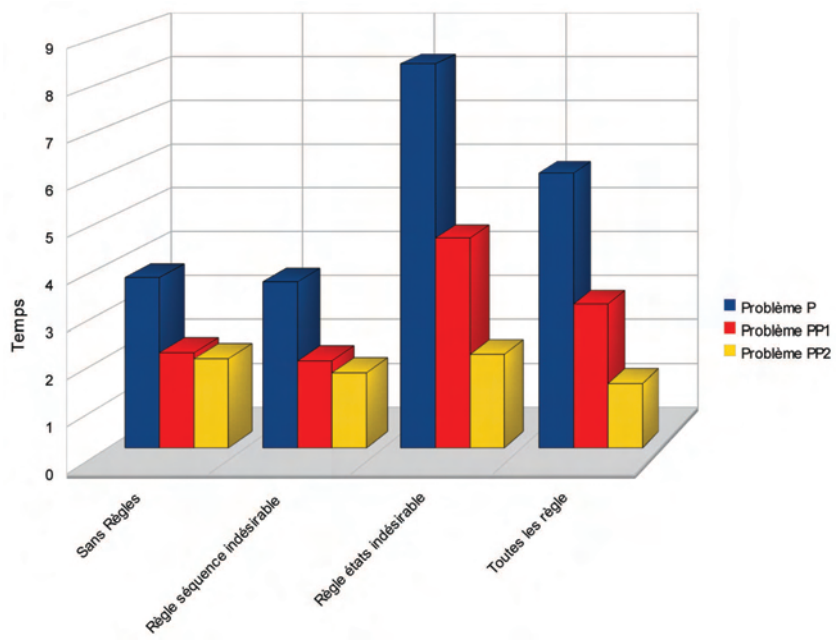


FIGURE 6.8 – Influence de l'introduction des règles sur le temps de calcul d'un plan : le test est effectué pour une planification sans règles, avec toutes les règles, pour une règle évaluée en ligne et une autre règle évaluée après le processus de raffinement.

et *états indésirables* qui sont évaluées après la phase de raffinement.

La planification pour un robot réel et qui interagit avec des humains impose deux critères que le planificateur doit satisfaire :

1. produire le meilleur plan qui respecte les règles sociales.
2. La planification en temps réel, ce qui signifie que le planificateur doit fournir une solution le plus rapidement possible, sinon on risque de compromettre l'interaction avec l'humain en suscitant chez lui un sentiment d'impatience ou d'incompréhension.

HATP présente un défaut majeur, car son algorithme est un algorithme HTN qui est donc exponentiel en temps comme l'a montré Erol et .al dans [Erol, K. 94]. De plus la recherche de la meilleure solution impose à HATP l'exploration de toutes les solutions existantes. Pour cela, il doit parcourir tout l'espace de recherche associé au problème. Le travail de Montreuil [Montreuil 08] montre que la taille de l'espace de recherche varie en fonction de la complexité du problème. Cette complexité évolue en fonction des dépendances entre les tâches, c'est-à-dire que pour un même problème, si la recherche est effectuée dans un domaine HTN totalement ordonné, la dimension de l'espace de recherche est beaucoup plus réduite que si la recherche est effectuée dans un domaine HTN partiellement ordonné.

L'algorithme HATP est particulièrement sensible à cela : du fait que c'est un planificateur multi-agent, son domaine de planification est obligatoirement partiellement ordonné. Ceci est confirmé par la mesure de la dimension de l'espace de recherche pour les trois problèmes de l'exemple précédent. Le tableau 6.1 montre que plus le problème est contraint plus la dimension

	Nombre total de branches	Branches menant à un échec	Branches menant à un succès
Problème P	> 285678	> 118816	> 166842
Problème PP1	4722	1261	3461
Problème PP2	827	80	27

TABLE 6.1 – Dimensions des espaces de recherche associés aux problèmes soumis à HATP

de l'espace de recherche est réduite. Il paraît clair que, pour pouvoir répondre au critère planification en temps réel, il est nécessaire de munir l'algorithme HATP d'une heuristique pour le diriger durant la phase de recherche.

6.2.2 Efficacité de l'heuristique

Nous allons tester l'efficacité de l'heuristique sur un scénario d'assemblage d'objets. Le but du scénario et d'assembler une table, celle-ci est constituée de quatre pieds et d'une planche : $pied1, pied2, pied3, pied4, planche$. Pour assembler deux pièces, nous devons utiliser un outil. Deux types d'outils sont disponibles : $riveteuse, tournevis$, mais ils ne peuvent être utilisés qu'avec certaines pièces. La *riveteuse* ne peut être utilisée que pour assembler le *pied*1 et le *pied4* à la *planche*, et de même, le *tournevis* ne peut être utilisé que pour assembler le *pied*2 et le *pied*3 à la *planche*.

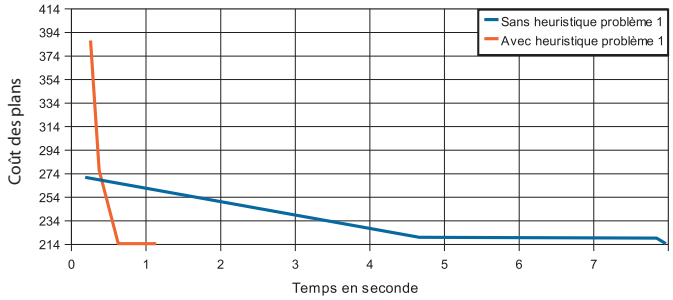
Dans ce scénario nous avons trois agents, deux robots (*Jido* et *HRP2*) et un humain *Tom*. Tous les trois sont dans un atelier où il y'a une table de travail qui sert à assembler la table. Il y'a également différentes caisses qui servent à stocker les pièces et les outils (une caisse pour les outils, une autre pour les pieds et une dernière pour les planches).

Nous supposons que l'action d'assembler deux pièces ne peut être faite que par un humain aidé par un autre agent (Dans notre cas un robot). L'effet de cette action est que les deux objets assemblés disparaissent et qu'un nouvel objet est créé. Nous supposons que le robot *HRP2* ne peut pas effectuer d'action de déplacement afin de forcer le planificateur à faire participer l'humain.

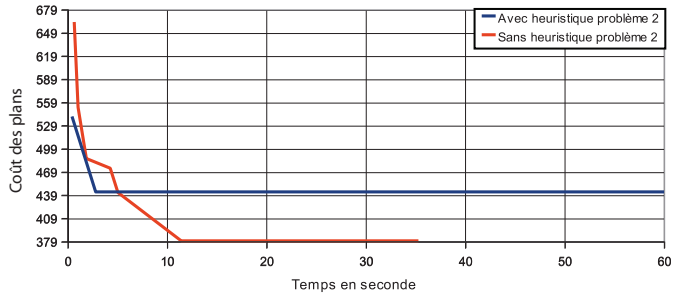
Nous allons donner au planificateur deux buts. Le premier, appelé *problème1* correspond à l'assemblage d'un pied et d'une planche et le second, *problème2* correspond à l'assemblage de deux pieds et une planche. Nous limitons le temps de calcul du planificateur à 1min.

Les figures 6.9 illustre l'évolution du coût des plans en fonction du temps pendant le processus de planification. Les figures 6.9(a) et 6.9(a) correspondent respectivement aux réponses du *problème1* et *problème2*. Nous constatons que dans les deux cas le planificateur converge plus rapidement vers la solution optimale. Pour le *problème1*, l'heuristique conduit le planificateur vers la solution optimale en moins d'une seconde contre $8s$ pour le même algorithme sans heuristique. Pour le second but *problème2*, l'heuristique conduit vers la solution optimale en $11s$, par contre, pour le planificateur sans heuristique la solution optimale n'est pas atteinte dans la limite de temps imparti.

L'utilisation de l'heuristique nous assure la convergence rapide vers la solution optimale ou une solution approchant l'optimum dans la limite du temps accordé au planificateur, c'est-à-dire



(a) Comparaison pour le problème 1



(b) Comparaison pour le problème 2

FIGURE 6.9 – Comparaison entre l'algorithme HATP avec et sans heuristique

qu'elle nous permet de satisfaire le critère de planification en temps réel.

6.3 Utilisation de HATP sur un robot réel

Dans cette section nous allons voir l'intégration de HATP dans une architecture dédiée à l'interaction homme-robot. Cette architecture tourne actuellement sur un robot interactif qui peut partager son espace et coopérer avec des humains.

6.3.1 Architecture logicielle de contrôle

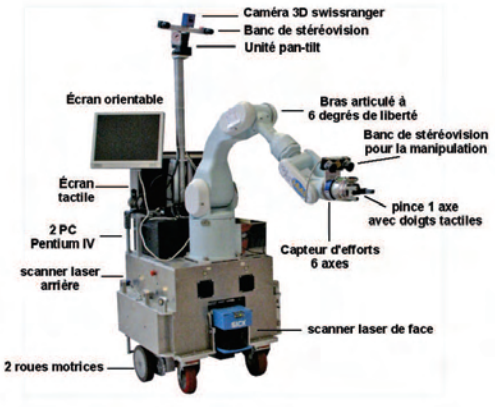


FIGURE 6.10 – Le robot Jido

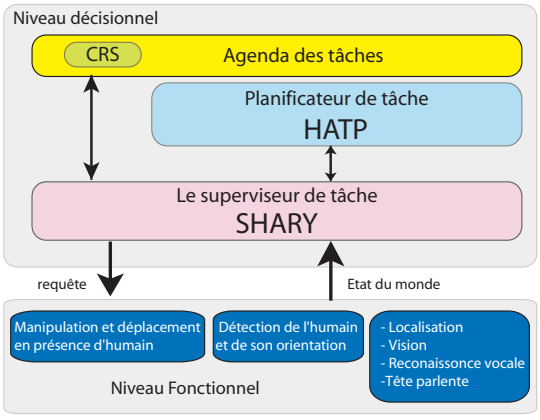


FIGURE 6.11 – Architecture logicielle

Le robot utilisé pour les expérimentations réelles s'appelle Jido et est illustré par la figure 6.10. Ce robot est composé d'une plateforme mobile contrôlée par deux roues motrices ainsi que d'un bras articulé à six degrés de liberté. La perception de l'environnement se fait par l'intermédiaire de plusieurs capteurs, dont notamment un scanner laser orientable à l'avant, un scanner laser fixe à l'arrière, des capteurs d'efforts sur la pince, un banc de stéréovision fixe pour la manipulation d'objets, un banc de stéréovision orientable et une caméra 3D. Jido est également équipé d'un écran tactile, lui permet de communiquer avec son partenaire humain via une interface.

L'utilisation d'un planificateur sur un robot exige que le robot soit doté d'une architecture logicielle adaptée permettant son implantation et son interfaçage avec les autres éléments de l'architecture logicielle du robot. Nous avons donc développé une architecture logicielle [Clodic 09] pour le niveau délibératif de contrôle du robot. Cette architecture permet une supervision adaptée à la problématique des robots assistants et un interfaçage avec un planificateur tel que HATP. L'architecture logicielle développée pour des applications robotiques interactives est illustrée par la figure 6.11. Cette architecture se compose de deux niveaux : le niveau fonctionnel et le niveau délibératif. Le niveau fonctionnel est composé d'un ensemble de modules logiciels permettant de contrôler les actionneurs du robot et de transcrire les données fournies par les capteurs en informations exploitables par le niveau délibératif, ou bien d'exécuter les commandes de ce dernier. Dans le cas de Jido, le niveau fonctionnel contient notamment un module de vision robotique [Burger 08] permettant la détection, la reconnaissance et le suivi des humains se trouvant aux alentours du robot, ainsi qu'un module de détection des positions des partenaires humains, à l'aide des données fournies par les scanners lasers [Sisbot 06]. Jido est aussi équipé d'un synthétiseur vocal [Bailly 01] qui lui permet d'interagir avec ses partenaires humains, ainsi qu'un planificateur de mouvement et de déplacement en présence d'humain [Sisbot 07a], [Sisbot 07c]. Ces modules fournissent des primitives fondamentales pour un robot interactif.

Le niveau délibératif est composé de trois modules principaux en plus de notre planificateur de tâches :

1. *La base de connaissances* est composée de connaissances de trois types :
   - Les *connaissances a priori* représentent les données invariables ciblant l'environnement dans lequel évolue le robot : la carte, les meubles présents, les zones topologiques, les objets présents, etc... Les connaissances a priori sont des connaissances *fixes* dans le temps c'est-à-dire qui ne seront pas modifiées lors du déroulement de la manipulation avec le robot.
   - Les *connaissances contextuelles* représentent les données portant sur l'environnement, qui vont changer pendant le déroulement de la manipulation : position des objets, personnes en présence, etc.
   - Les *connaissances sur les partenaires* représentent les connaissances portant sur les partenaires humains du robot. Pour chaque partenaire humain du robot, on regroupe l'ensemble des informations le concernant.

2. *L'agenda de tâches* est chargé de définir les tâches de haut-niveau qui doivent être réalisées. Il maintient une liste ordonnée de tâches de haut niveau qu'on appelle *TODO liste*, il a un système de raisonnement qui donne au robot un comportement proactif, par exemple en prenant l'initiative d'adopter un comportement d'un robot curieux, en décidant d'acquérir des informations sur l'état de l'environnement (Par exemple, l'exploration de nouveaux objets placés par une personne sur une table). Il est connecté avec un système de reconnaissance de chroniques (CRS) [Dousson 93], qui permet de reconnaître une situation à partir d'une conjonction d'événements partiellement ordonnés. Dans la mise en œuvre courante, CRS a été essentiellement utilisé pour interpréter les activités des humains qui se trouvent dans le voisinage du robot. L'agenda de tâches génère les tâches à accomplir à partir d'un ordre explicite des humains ou bien à partir des situations reconnues par CRS, il ordonne ces tâches en fonction de leur priorité et du contexte dans lequel le robot se trouve. Ce module permet donc au robot de répondre aux ordres/suggestions de ses partenaires, mais également de réagir à des situations spécifiques.
3. *Le superviseur* : Il est en charge d'émettre les requêtes vers le niveau fonctionnel afin de permettre la réalisation des actions courantes. Il est également en charge de contrôler le bon déroulement de l'exécution des tâches courantes, de gérer la communication ciblant l'interaction homme-robot, et de contrôler les activités de l'humain au cours de la réalisation d'un plan. Le superviseur qui a été développé s'appelle SHARY, son fonctionnement est détaillé dans [Clodic 07]. De plus, ce superviseur est capable de considérer une hiérarchie complète de tâches afin de pouvoir vérifier le bien fondé de l'action courante à différents niveaux d'abstraction.

Dans l'état actuel de l'implémentation de l'architecture, la base de connaissances et le superviseur SHARY sont écrits à l'aide du système de raisonnement procédural OpenPRS [Ingrand 96]. L'agenda est implanté, mais il ne gère qu'une liste précise et limitée de tâches. Les schémas de communication sont également décrits sous une forme exploitable par OpenPRS. Le planificateur HATP est quant à lui implanté sous la forme d'un exécutable se connectant au système OpenPRS utilisé par SHARY, et communiquant avec celui-ci par l'intermédiaire de messages sous forme de chaînes de caractères. Les requêtes échangées et leurs syntaxes entre SHARY et HATP sont les suivants :

Les messages de SHARY vers HATP

Il existe deux types de requêtes qui passent du superviseur vers HATP :

Message de mise à jour de la base de faits :

La mise à jour de la base de faits est réalisée par l'envoi d'un message d'initialisation puis d'une série de messages qui mettent à jour les valeurs des attributs, et enfin un message de fin de mise à jour.

Message de début de mise à jour : ce message permet de sortir le planificateur de sa mise en veille et aussi de vider tous les attributs de type dynamique. Pour cela on utilise la syntaxe :

(HATP-START-UPDATE)

Message de fin de mise à jour : ce message permet de remettre le planificateur en mode veille. Pour cela on utilise la syntaxe :

(HATP-END-UPDATE)

Messages de mise à jour : pour ce message on utilise la syntaxe :

(WP-(axt-id <ID>) <attribut> <entité> <valeur>)

On peut constater que dans le message, on reçoit l'identifiant de requête, les noms de l'entité et de l'attribut concernés et la nouvelle valeur de l'attribut. Si le couple (<entité>, <attribut>) est un atome dans HATP, sa valeur sera instanciée à <valeur>. Si le couple est un ensemble dans HATP, l'élément <valeur> sera ajouté dans l'ensemble.

On considère qu'il est de la responsabilité du superviseur de renvoyer une mise-à-jour complète des éléments dynamiques du monde, sinon l'état du monde de HATP sera différent de celui de SHARY, ce qui peut conduire à la production de plans erronés s'ils existent.

Demande de planification :

Cette requête permet de définir le but et aussi de lancer la planification. La requête prend cette syntaxe :

(HATP-SEARCH (parent-id <pID>) (task-type <task name>) (p (<param type> $< param_1 >$**) (<param type>** $< param_2 >$**)... (<param type>** $< param_n >$**)))**

Une fois que le superviseur a envoyé la requête à HATP, il va attendre un message de fin de planification.

Les messages de HATP vers SHARY

Le seul message que SHARY reçoit de HATP est un message de fin de planification. Ce message peut prendre deux formes :

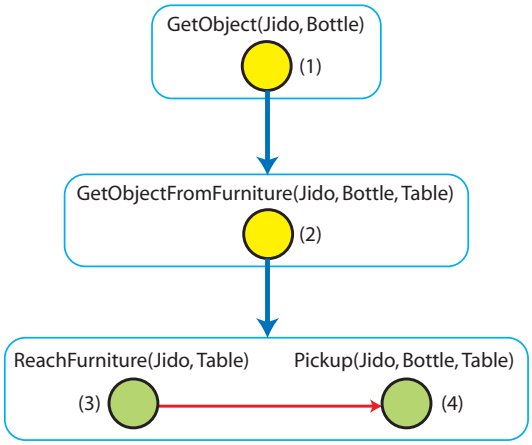
– Échec de planification : le message est émis par HATP quand aucun plan n'a été trouvé durant le processus de planification. La syntaxe du message est :

(HATP-END (axt-id <ID>) (report NO-PLAN))

– Retour d'un plan : le message est émis par HATP quand un plan est retenu par le processus de planification. Dans ce message, on décrit tout l'arbre de raffinement, c'est-à-dire toutes les tâches présentes, les liens de précédence et les liens de décomposition. Pour cela on utilise la syntaxe suivante :

1. Les tâches :

(HATP-TASK (hatp-id <ID>) (axt-id <ID>) (parent-task <taskID>) (task-type <task name>) (Actor (A <agent $name_1 >$**)... (A <agent** $name_n >$**)) (p (<param type>** $< param_1 >$**)... (<param type>** $< param_n >$**)) (coop-type <JOINT OR AUTONOMOUS>))**



(a) Représentation graphique d'un plan produit par HATP

```
(HATP-TASK (hatp-id 1) (axt-id 0) (parent-task -1) (task-type GetObject) (Actor (A Jido)) (p (Object Planche)) (coop-type AUTONOMOUS))
(HATP-LINK (axt-id 0) (t-1 1) (t-st 2))
(HATP-TASK (hatp-id 2) (axt-id 0) (parent-task 1) (task-type GetObjectFromFurniture) (Actor (A Jido)) (p (Object Planche) (Furniture PlancheCupboard)) (coop-type AUTONOMOUS))
(HATP-LINK (axt-id 0) (t-1 2) (t-st 3))
(HATP-TASK (hatp-id 3) (axt-id 0) (parent-task 2) (task-type ReachFurniture) (Actor (A Jido)) (p (Furniture PlancheCupboard)) (coop-type AUTONOMOUS))
(HATP-TASK (hatp-id 4) (axt-id 0) (parent-task 2) (task-type PickupIn) (Actor (A Jido)) (p (Object Planche) (Furniture PlancheCupboard)) (coop-type AUTONOMOUS))
(HATP-LINK (axt-id 0) (t-1 3) (t-2 4))
```

(b) Plan HATP traduit pour SHARY

FIGURE 6.12 – Exemple de traduction d'un plan produit par HATP pour SHARY : Dans cet exemple, le but est que le robot Jido prenne une bouteille.

Dans ces tâches, on retrouve l'identifiant de la requête de planification *axt-id*, l'identifiant que HATP donne à la tâche *hatp-id*, *parent-task* va servir à décrire la hiérarchie des tâches en identifiant la tâche mère de la tâche en cours. On retrouve ensuite le nom de la tâche, les acteurs et la liste des paramètres de celle-ci, et enfin le type de la tâche si c'est une tâche jointe ou individuelle.

2. les liens de précédence :

   (HATP-LINK (axt-id <ID>) (t-1 <taskID>) (t-2 <taskID>))

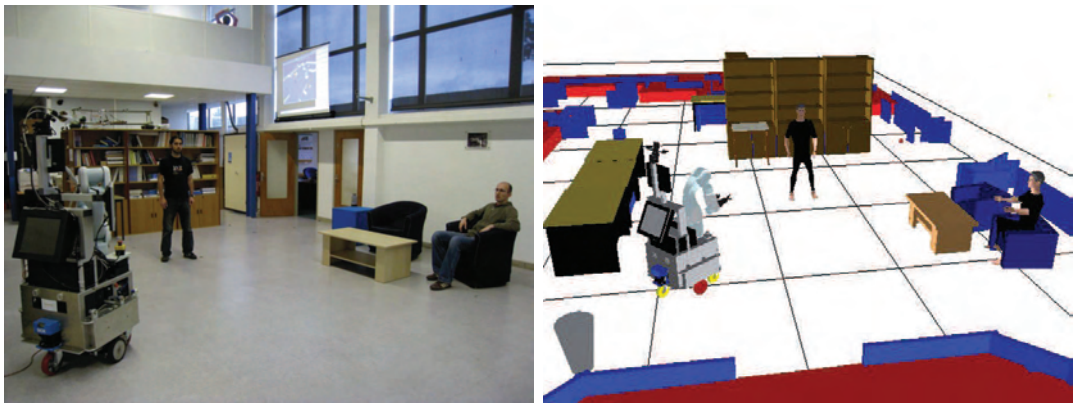
   Dans la description on a l'identifiant de la requête de planification *axt-id* et les identifiant HATP des deux tâches. Le sens de la précédence va de la tâche *t-1* vers *t-2*

3. les liens de décomposition :

   (HATP-LINK (axt-id <ID>) (t-1 <taskID>) (t-st <taskID>))

   C'est exactement la même description que pour les liens de précédence. La seule différence est que le symbole précédant la deuxième tâche change de *t-2* à *t-st*, ce nouveau symbole identifie la première tâche de la décomposition puisque les autres tâches sont toutes identifiées dans la description de la tâche par **(parent-task <taskID>)**.

La traduction du plan de HATP vers SHARY est illustrée par les figures 6.12(a) et 6.12(b) dans lesquelles on peut voir la représentation d'un plan et sa traduction.



(a) L'environnement où évolue le robot (b) Le modèle 3D de l'environnement

FIGURE 6.13 – Environnement de travail du robot et sa représentation en 3D

FIGURE 6.14 – Situation de départ du scénario

6.3.2 Scénarios de l'expérimentation

Ce scénario se déroule dans un environnement illustré par la figure 6.13. Du point de vue symbolique cet environnement est composé de quatre zones : la zone d'attente dans laquelle Jido doit se rendre s'il n'a plus rien à faire, de cette zone il peut observer tout ce qui se passe à proximité de lui. La zone sofa où les humains peuvent s'asseoir. Trois zones où on peut déposer et prendre les objets : la zone table, la zone bar et la zone bibliothèque. Le robot accomplit plusieurs tâches de haut-niveau telles que servir à boire, nettoyer les tables, rapporter des objets, etc. Le défi pour le robot est de réaliser ses buts de façon robuste tout en exhibant ses qualités sociales et interactives, c'est-à-dire générer des comportements sûrs, socialement acceptables et intelligibles pour tous ses partenaires humains.

Nous allons présenter deux scénarios qui illustrent les capacités du robot à accompagner un humain dans sa vie courante. Le premier est un scénario type "pick-and-place" où le robot doit manipuler des objets : prendre, ranger et transmettre. Le deuxième scénario porte sur la capacité

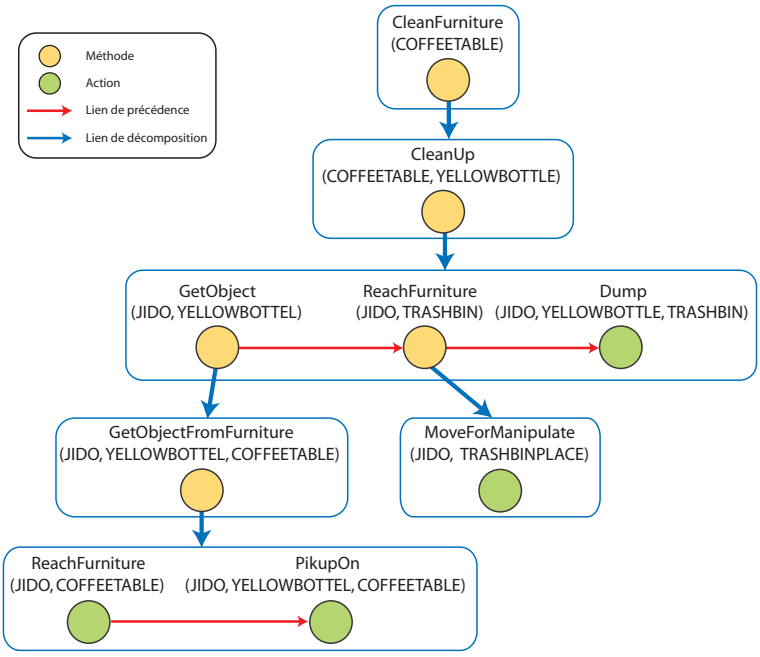


FIGURE 6.15 – Plan produit par HATP

du robot à générer un comportement proactif en analysant et en réagissant aux activités des humains partenaires pour satisfaire leurs besoins.

6.3.2.1 Scénario pour la manipulation

La situation de départ de ce premier scénario est illustrée par la figure 6.14. L'humain est assis sur le sofa, le robot est à sa position de base, et sur la table devant l'humain il y a une bouteille. L'humain va demander au robot de nettoyer la table. Cet ordre va être transformé par l'agenda des tâches en tâche prioritaire. Le superviseur SHARY va ensuite envoyer une requête de mise à jour de la base de faits de HATP. Une fois la base de faits mise à jour, HATP recevra le but à réaliser sous forme d'une seule tâche *CleanFurniture(COFFEETABLE)* et commencera la planification. La structure HTN et les règles sociales utilisées sont semblables à celles utilisées dans le scénario pour évaluer l'influence des règles sur la production de plans. La figure 6.15 illustre le plan produit par HATP. Dans ce plan, on peut constater que toutes les règles sont respectées et qu'il n'y a pas d'intervention de l'humain. On attire l'attention sur le fait que, dans ce modèle, on utilise quatre actions de déplacements différentes qui sont :

1. *MoveForManipulate* : Action de déplacement qui a pour but la manipulation d'objet.
2. *MoveForLook* : Action de déplacement qui a pour but l'observation.
3. *MoveForGetOrGive* : Action de déplacement qui a pour but l'échange d'objets avec un autre agent.
4. *MoveForTalk* : Action de déplacement qui a pour but le dialogue avec un autre agent.

Au niveau symbolique tous ces déplacements sont exactement modélisés de la même manière, mais l'intérêt d'une telle distinction se trouve au niveau de l'exécution, puisqu'un déplacement

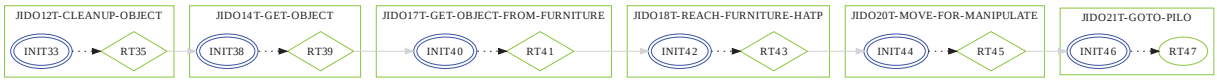


FIGURE 6.16 – Exemple de la transformation du plan opéré par SHARY

pour observer ou donner un objet ne va pas être réalisé de la même façon. Cette différence réside dans le choix de la configuration finale du robot. Cette distinction permet donc au superviseur d'appliquer des contraintes supplémentaires sur le choix de cette configuration.

Après la production d'un plan par HATP, celui-ci va être transmis au superviseur qui va se charger de l'exécution. Ce plan, sous sa forme actuelle, n'est pas directement exploitable puisque les actions, considérées comme étant de bas niveau, sont en réalité des actions de haut-niveau pour la supervision. Pour cela, le plan va subir un raffinement pour être transformé en un schéma de communication interne [1]. La figure 6.16 illustre la transformation d'une branche du plan. Les rectangles verts sont la représentation des tâches. Ils sont composés d'un contrôleur d'exécution (les cercles bleus) et de la tâche proprement dite (losange vert). Les flèches grises représentent les liens de décomposition et les flèches en pointillé noir représentent les transitions.

(a) Situation de dépare : humain sur le sofa robot sur sa base

(b) Le robot va à la table et prend la bouteille

(c) Le robot va à la poubelle

(d) Le robot jette la bouteille dans la poubelle

FIGURE 6.17 – Story-board exécution du plan HATP pour le but CleanFurniture(COFFEETABLE)

Les étapes de réalisation du scénario sont décrites par les figures 6.17. Le robot se déplace pour aller chercher l'objet, le prend et va le jeter dans la poubelle. Une fois le plan entièrement réalisé et si le robot ne reçoit pas de requête de la part de l'humain, il va sur sa base et se met en mode observation. Cette tâche est toujours présente dans la TODO liste de l'agenda et si le

1. Un schéma de communication est un automate à état fini, ou chaque transition représente des conditions sur l'état du monde et chaque état est une action à réaliser.

robot n'a rien à faire, il l'exécute. Cela lui permet de garder sa base de faits à jour.

Maintenant, on considère pour le même problème le cas où il y a échec de l'exécution. On considère deux sources d'échecs possibles : (1) les causes environnementales, c'est-à-dire les changements sur les états des entités, que le robot n'a pas observé ou qu'il n'a pas pu observer, par exemple un autre agent qui déplace un objet. Si le but de notre robot porte sur cet objet, cela mènera inévitablement à un échec. (2) les causes internes, c'est-à-dire les défaillances du système. Par exemple, si notre robot doit prendre un objet et qu'il n'arrive pas à le localiser sur une table à cause d'un défaut au niveau de ses caméras, cela mènera également à un échec.

La figure 6.18 illustre les différentes étapes de la réalisation de la tâche *CleanFurniture(COFFEETABLE)*, les communications entre HATP et SHARY et les streams des deux agents robot et humain. Au début et comme précédemment, le robot reçoit un ordre verbal de l'humain lui demandant de nettoyer la table. SHARY envoie la requête à HATP après avoir mis à jour la base de faits. Celui-ci lui retourne un plan que le superviseur commence à exécuter. Arrivé à la tâche *PickUp* (prendre l'objet), le superviseur commence par localiser l'objet et s'aperçoit que l'objet est inaccessible. Dans ce cas, le superviseur SHARY met à jour le fait (YELLOWBOTTLE.reachable=false) et ré-envoie une requête de planification à HATP, avec la mise à jour de sa base de faits. Le planificateur envoie le plan trouvé correspondant à $(AskHelp(Jido, H1, YELLOWBOTTLE) \rightarrow Replan())$. La première action est une demande d'aide qui se traduit par une communication verbale du robot vers l'humain concernant l'objet *YELLOWBOTTLE*. La deuxième action est une action interne entre le planificateur et le superviseur, qui lui indique que le but n'est pas atteint et qu'il faudra redemander une planification après exécution de la séquence actuelle.

Une question vient tout de suite à l'esprit : "pourquoi le planificateur n'a-t-il pas produit de plan où il fait intervenir directement l'humain ?". La réponse est simple, on a fait le choix d'interdire au planificateur de faire intervenir l'humain dans le cas où c'est celui-ci qui a formulé la requête au robot. Ce qui correspond au comportement des humains. Cette interdiction est modélisée par l'attribut booléen "*Involved*" rattaché aux entités de type *Agent*.

Dans ce cas de figure, la suite de la réalisation du plan dépend de la réponse de l'humain. S'il refuse d'intervenir alors le but est abandonné et notifié à l'humain. Dans le cas contraire, le superviseur met à jour l'attribut "*Involved*" à vrai et relance une demande de planification. Le planificateur HATP, dans ce cas, va produire un plan faisant intervenir l'humain. Dans ce plan, l'humain prend la bouteille et la donne au robot qui va la jeter à la poubelle.

6.3.2.2 Scénario pour la pro-activité

Ce scénario se déroule dans le même environnement que précédemment. On a toujours deux agents : le robot et l'humain. Le but de cet exemple est de démontrer l'aptitude du système à

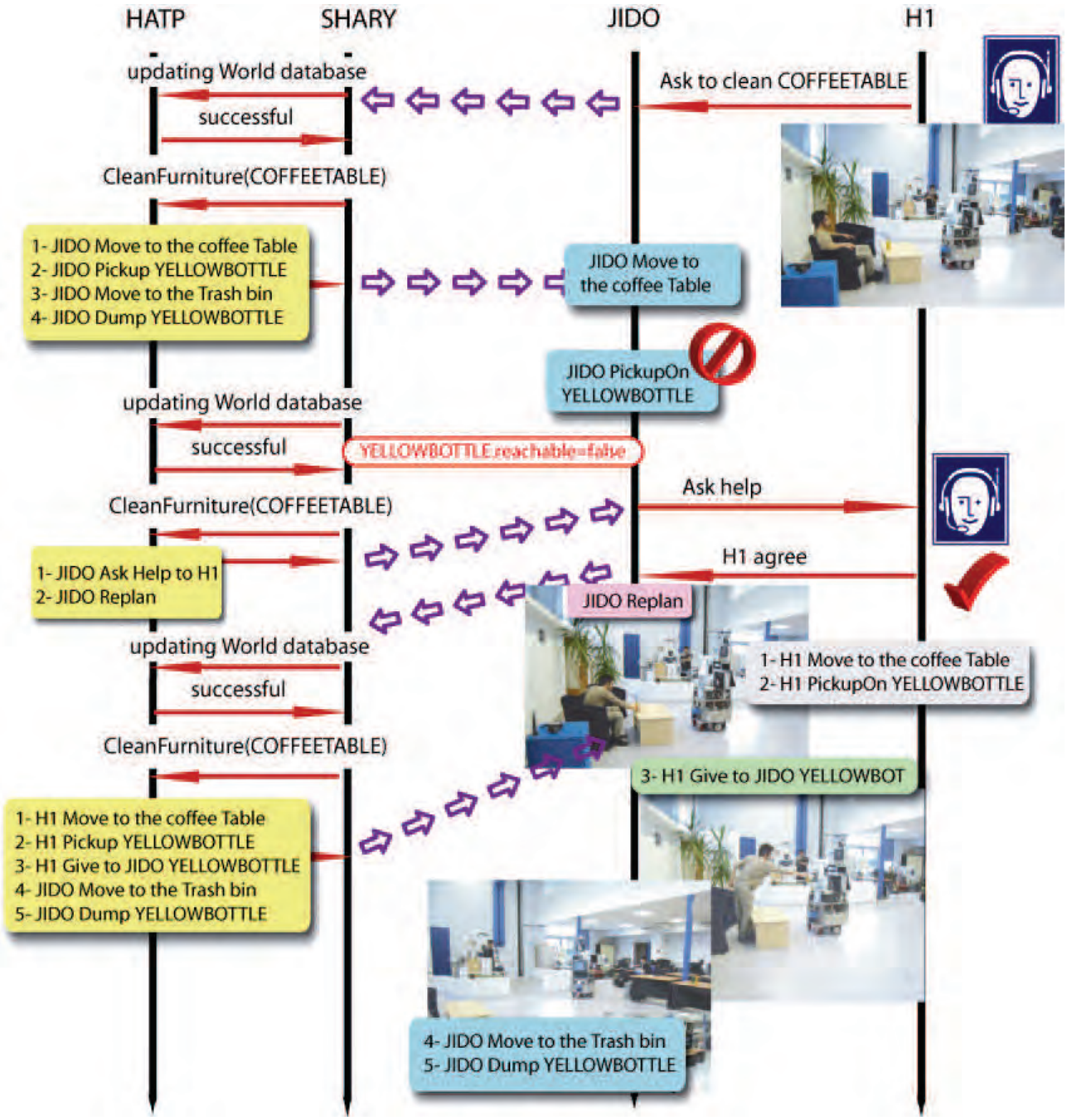
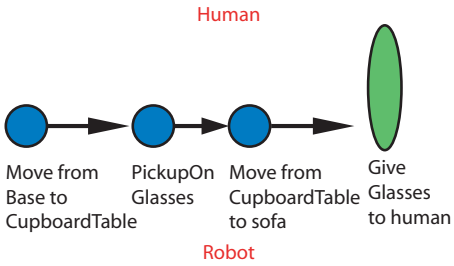
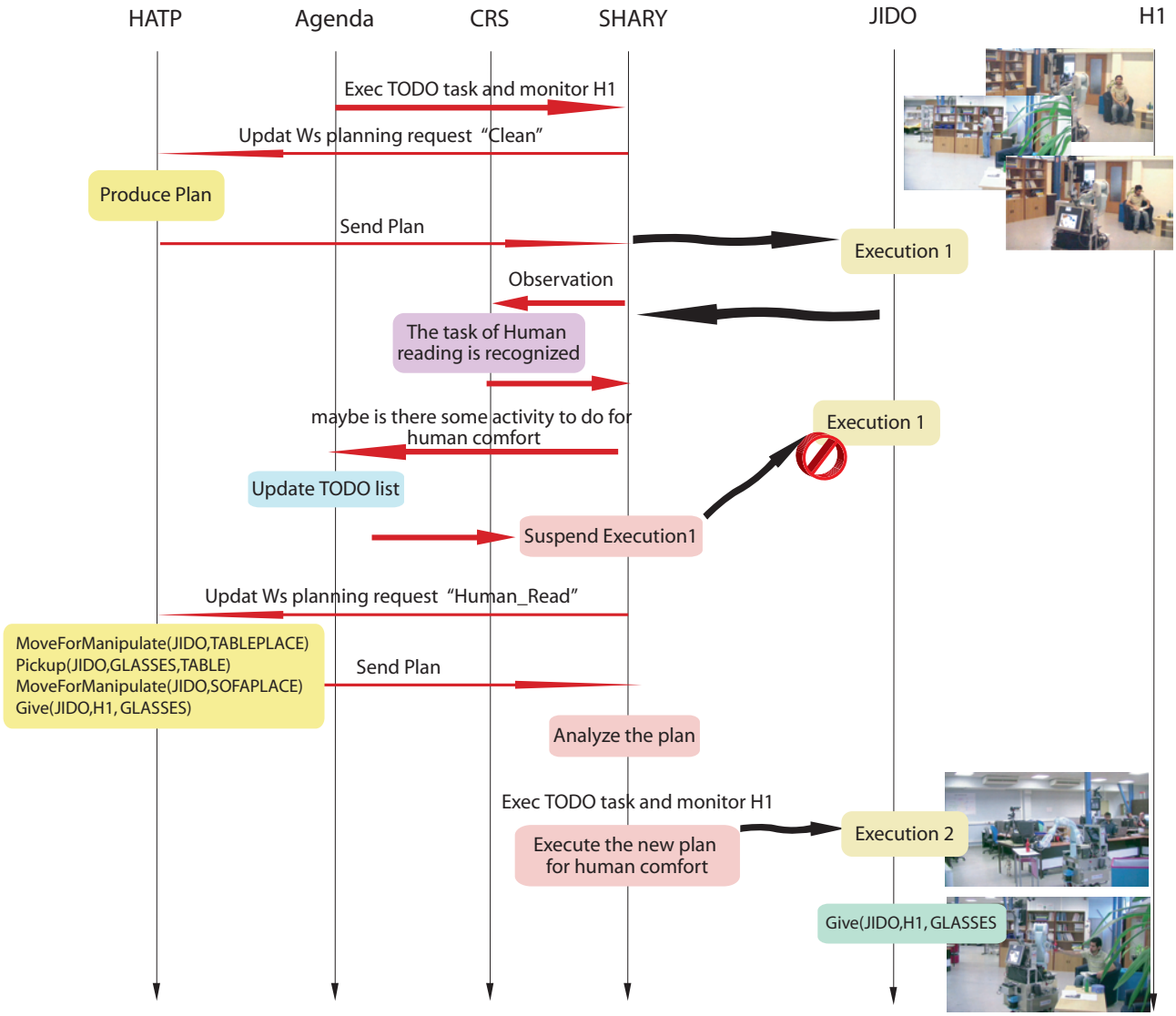


FIGURE 6.18 – Les différentes étapes de la réalisation de la tâche CleanFurniture(COFFEETABLE) avec toutes les interactions entre l'humain et le robot, mais également entre le superviseur SHARY et le planificateur HATP.



(a) Le plan produit par HATP pour la tâche Human_read



(b) Les différentes interactions les modules pour générer un comportement proactif.

FIGURE 6.19 – Plan et étape de la réalisation de la tâche Human_read

générer un comportement proactif du robot pour aider l'humain à partir de ses observations et les possibilités qu'offre son niveau décisionnel. Comme on l'a défini précédemment, les entrées de l'agenda sont soit un ordre direct venant des humains soit générées via CRS. Dans ce scénario, CRS est utilisé exclusivement pour la reconnaissance des activités des humains se trouvant dans le voisinage du robot. La liste de tâches qu'il peut reconnaître est très réduite et prédéterminée.

L'idée est qu'à partir des tâches reconnues par CRS sur les activités des humains, on donne la possibilité au robot d'intervenir pour apporter son aide. Quand une activité humaine est reconnue, celle-ci est transformée en tâche. Pour juger si le robot peut apporter son aide, le superviseur demande une planification de la tâche. Si le plan trouvé fait intervenir le robot le superviseur exécute le plan, sinon cette tâche est abandonnée.

(a) Situation de départ humain sur le sofa le robot sur sa base

(b) L'humain se dirige vers la bibliothèque et prend un livre

(c) L'humain va se rasseoir sur le sofa

FIGURE 6.20 – Story-board du plan que CRS peut reconnaître

La figure 6.19(b) illustre les différentes étapes et les différentes interactions entre SHARY, HATP, l'agenda et CRS, pour générer ce comportement proactif. L'une des tâches que CRS peut reconnaître est la tâche (*Human_read*) illustrée par la figure 6.20. Elle se traduit par la séquence de tâches : "L'humain va à la bibliothèque pour prendre un livre et va s'asseoir". Quand CRS reconnaît cette activité de l'humain, il en informe le superviseur SHARY. Comme le superviseur n'est pas apte à juger de la priorité de la tâche, il envoie l'information à l'agenda qui va la classer dans sa TODO liste en fonction de sa priorité et des tâches déjà présentes. Si la tâche n'est pas prioritaire, le superviseur continue l'exécution de sa tâche en cours. Dans le cas contraire, la tâche (*Human_read*) se retrouve en tête de liste des tâches à réaliser. SHARY envoie une requête de planification à HATP en mettant à jour sa base de faits. Le planificateur produit un plan en fonction de l'état du monde, de son domaine de planification et du modèle des agents. Dans ce scénario, le modèle de l'agent possède l'attribut "*NeedGlasses*", qui définit si un agent a besoin ou non de lunettes. Cet attribut lié à l'agent va conditionner le choix de la décomposition de la tâche (*Human_read*), si l'humain a besoin de lunettes la décomposition sera "Aller chercher le livre et aller chercher les lunettes". Dans le cas contraire, l'humain n'a pas besoin de lunettes on supprime totalement la tâche d'aller chercher les lunettes. Dans le scénario, on suppose que l'humain partenaire a besoin de lunettes. Le plan produit par HATP est illustré par la figure 6.19(a) (Du fait que notre robot ne peut manipuler des objets de petite taille, l'objet *Glasses* est définit comme une bouteille dans le monde réel), ce plan est produit en moins de $400ms$. Dans ce plan, on remarque que le planificateur fait intervenir le robot pour aller chercher les lunettes et les transmettre à l'humain. Ce plan va être transmis au superviseur, qui procédera à une analyse

de ce dernier, pour chercher des tâches impliquant le robot. Si ces tâches existent, ce qui est notre cas, le superviseur exécutera le plan. Dans le cas contraire ou si le planificateur ne trouve pas de plan, le superviseur abandonne la tâche et va sélectionner une autre tâche dans la TODO liste.

Ces scénarios démontrent l'aptitude du planificateur HATP à fonctionner en temps réel de façon embarquée sur un robot interactif. Il démontre aussi la capacité de HATP à générer des plans corrects et cohérents qui correspondent aux problèmes et aux données qui lui sont soumis. Les plans produits par HATP prennent explicitement en compte les règles sociales définies dans son domaine, et permettent donc au robot de produire des plans adaptés à une situation où l'interaction homme-robot est un point essentiel de l'application.

6.4 Utilisation de HATP pour la planification multi-agents

Dans la section précédente, nous avons vu l'utilisation de HATP pour la planification et l'interaction avec l'humain et nous avons vu ses applications sur un robot réel. Néanmoins, HATP a été conçu comme un planificateur multi-agents, qui permet d'avoir un contrôle sur le comportement de plusieurs robots. Pour cela, nous avons construit des domaines pour des problèmes multi-agents dans le cadre du projet SCA2RS (Supervision et Coopération Autonome entre Aéronefs de Recherche et Sauvetage). Le scénario du projet concerne la recherche et le sauvetage en cas d'inondation de grande ampleur, par exemple suite à un tsunami ou à une rupture de digue. Le site d'intervention est un milieu urbain peu dense comprenant des maisons et quelques petits immeubles. Les survivants sont majoritairement sur les toits des maisons et d'immeubles, mais il n'est pas exclu que certains survivants dérivent sur des objets flottants. Le niveau d'eau n'est pas stabilisé, ce qui implique que le réseau des voies terrestres praticables est mal connu et peut évoluer. Il en est de même pour le réseau des voies navigables. Le but est d'exploiter une flotte de robots terrestres et volants pour effectuer des missions sur le terrain. On peut distinguer trois types de mission :

1. Mission d'exploration dans laquelle on va utiliser des drones pour explorer et prendre des images du terrain. Ces photos seront envoyées à la base où des opérateurs humains vont les examiner et décider s'il y a une cible ou non sur la zone photographiée.
2. Largage de kit, si une cible ou un survivant a été identifié, on lui envoie un kit de survie soit avec un drone soit avec un robot mobile.
3. Reconstitution du réseau des voies de navigation terrestre avec des drones à partir d'une cartographie déjà existante du réseau, afin de faciliter l'intervention des robots mobiles et des équipes de sauvetage.

HATP a été intégré dans l'architecture illustrée dans la figure 6.21. C'est une architecture pour un contrôle centralisé, elle est constituée de cinq éléments :

– Le simulateur : C'est une application développée par le partenaire du projet ROBOSOFT. Cette application crée un monde virtuel (figure 6.22) qui reproduit presque tous les

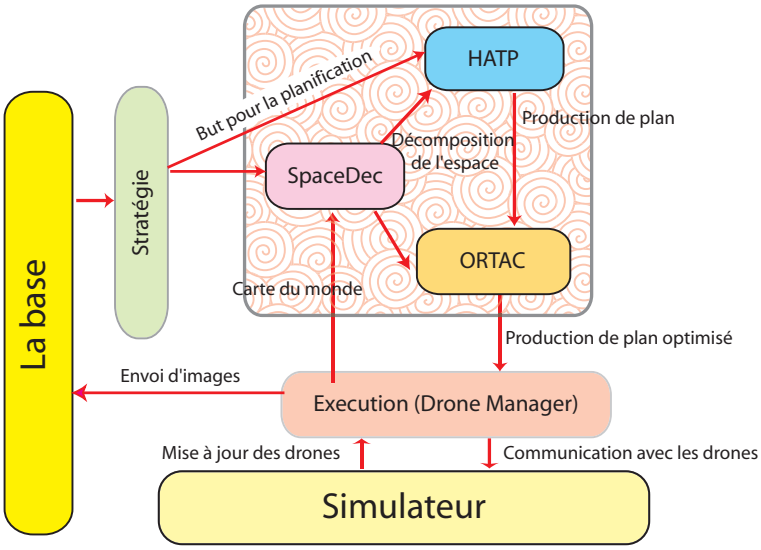


FIGURE 6.21 – Architecture décisionnelle du projet SCA2RS

phénomènes du monde réel tels que les phénomènes physiques comme la gravité, ou bien les phénomènes météorologiques comme la pluie ou le vent, etc. Il reproduit également les structures physiques des robots, leurs comportements, leurs actions, leurs capteurs, etc. Le simulateur intègre aussi le fait que les robots peuvent avoir des erreurs sur leurs données telles que des erreurs sur la position, etc.

– Le drone manager : c'est l'interface entre l'architecture décisionnelle et le simulateur. Elle permet de transmettre les plans calculés aux robots, d'acquérir les observations des robots pour constituer la base de faits. Elle permet également d'établir les communications entre la base et les robots.

– Stratégie : C'est un agenda des tâches. Il fixe la priorité des tâches et interprète les requêtes des opérateurs.

– SpaceDec (Space Decomposition) : C'est un module qui fait de la décomposition topologique 3D, il transforme les zones topologiques en un graphe de déplacement pour les drones, où chaque nœud représente une position en 3D dans l'espace. Les arcs sont des chemins ou couloirs aériens reliant les nœuds.

– ORTAC (Optimisation du Réseau Tactique des Acteurs au Contact) : C'est un ordonnanceur temporisé qui permet de trouver des plans optimaux à partir d'un réseau de contraintes. Il prend en charge la gestion du temps et des ressources.

– La base : Représente un ou plusieurs opérateurs humains qui supervisent les robots. Ils identifient les cibles sur les photos, créant ainsi automatiquement de nouveaux buts. Ils valident les voies praticables sur les photos envoyées par les drones, ce qui permet de

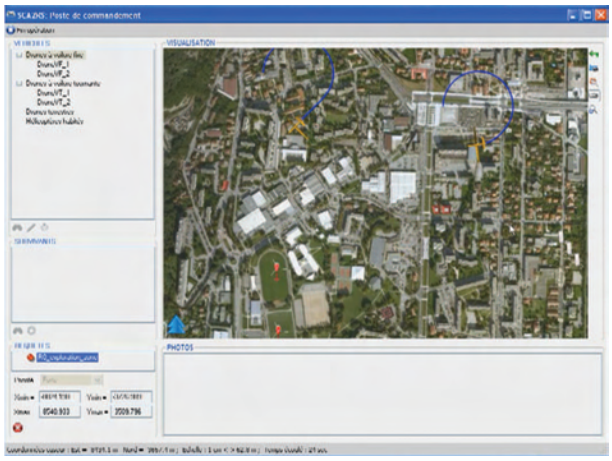


FIGURE 6.22 – Image extraite du simulateur

reconstruire la carte du réseau des voies de navigation terrestres.

L'architecture telle quelle est présentée ci-dessus n'est pas complètement achevée. Cependant, des tests d'intégration entre modules sont en cours de réalisation. Actuellement, HATP est connecté avec SpaceDec et ORTAC. SpaceDec est utilisé comme une entrée pour le planificateur en nous fournissant le graphe de navigation des drones et leur position, ainsi que la position des survivants. Avec toutes ces données, HATP va produire un plan qui tient compte des consommations en carburant et du nombre de kits de survie pour chaque drone. HATP gère également les communications entre les drones et la base, puisque le nombre de canaux de communication est limité. Cependant, le plan produit reste purement symbolique, puisque dans le plan produit, il n'y'a pas de gestion explicite du temps ce qui est crucial dans ce genre d'applications. Pour cela, HATP a été connecté à ORTAC, qui gère parfaitement ces contraintes. ORTAC prend en entrée le plan produit par HATP et aussi le graphe de navigation de SpaceDec. Il produit un nouveau plan dynamique qui prend en compte les contraintes de navigation ainsi que les contraintes temporelles qu'elles impliquent. On dit que le plan produit par ORTAC est dynamique puisqu'il est reconfigurable en ligne.

6.4.1 Interface entre HATP, SpaceDec et ORTAC

La communication entre HATP et les deux modules SpaceDec et ORTAC se fait via des sockets UDP et en utilisant des fichiers, suivant la syntaxe :

Les messages de HATP vers SpaceDec

GraphReady chemin/Nom_du_fichier.nodes chemin/Nom_du_fichier.links chemin/Nom_du_fichier.drones

Ce message informe HATP que le graphe de navigation est prêt. HATP retrouve le chemin des fichiers, les parsent et récupère les données nécessaires à la construction de sa base de faits. Dans tous les fichiers on utilise le (;) comme séparateur de données. les fichiers sont constitués de la façon suivante :

– Pour le fichier .nodes, on ordonne les informations comme suit :

Id du noeud ; position en x ; position en y ; position en z

– Pour le fichier .links, on ordonne les informations comme suit :

Id de l'arc ; id du noeud source ; id du noeud destination ; Distance entre les noeuds

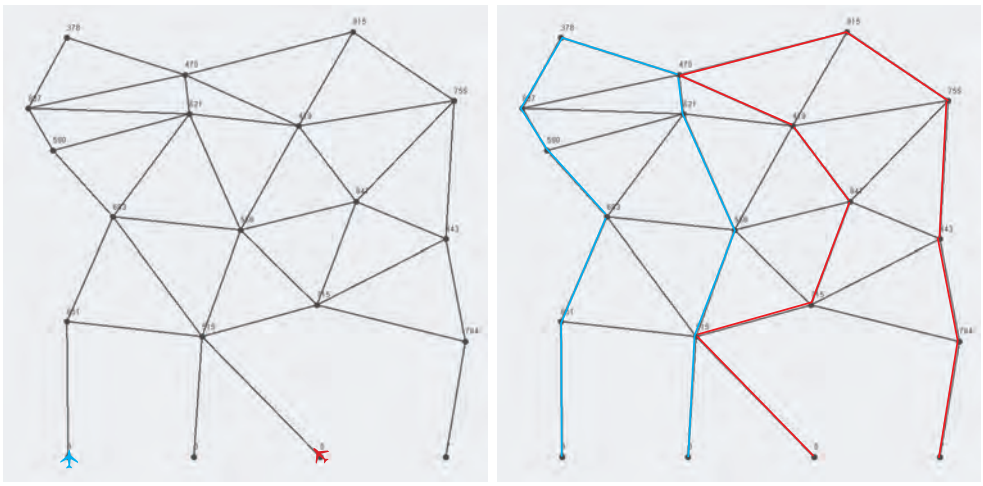
– Pour le fichier .drones, on ordonne les informations comme suit :

Id drone ; Id noeud de départ ; Id noeud d'arrivée

Les messages entre HATP et ORTAC

HATPReady chemin/Nom_du_fichier.pl

Ce message informe ORTAC qu'un plan a été trouvé. Le fichier que HATP produit est le fichier d'entrée ou domaine d'ORTAC. Il contient toutes les données nécessaires (ensemble des actions, ensemble des contraintes, graphe de navigation, etc.) au bon fonctionnement d'ORTAC.



(a) Situation de départ fourni par SpaceDec (b) Plan solution trouvé par HATP

FIGURE 6.23 – Exemple d'un problème et un plan pour une mission d'exploration

Nous allons illustrer le fonctionnement de HATP dans un exemple qui implique deux drones dans une mission d'exploration. Le graphe de navigation des drones et qui est également l'entrée de HATP est fourni par SpaceDec (figure 6.23(a)). Le plan produit par HATP est illustré par la figure 6.23(b). Les lignes rouges et bleues représentent les parcours des deux drones. Le temps

de calcul pour cette mission avec l'algorithme de HATP est de 20mn, pour trouver le plan optimal. Cela s'explique par : (1) pour ce type de problème, le facteur de branchement est très grand ce qui implique un grand espace de recherche. (2) la modélisation d'une méthode pour le parcours de graphe implique une récursivité sur celle-ci, et de ce fait, nous ne pouvons plus utiliser l'heuristique.

6.5 Conclusion

Ce chapitre nous a permis de mettre en avant les capacités du planificateur à produire des plans respectant les règles sociales définies dans son domaine de planification.

Nous avons également mis en avant la difficulté de produire des plans socialement acceptables en temps réel. En effet, avec le planificateur HATP, nous cherchons à trouver la meilleure solution. Pour cela, nous devons explorer toutes les solutions existantes, c'est-à-dire parcourir tout l'espace de recherche associé. Mais la taille de l'espace peut exploser si le problème contient plusieurs tâches n'ayant aucune contrainte de précédence entre elles. Nous avons également présenté les résultats de l'exploration avec une heuristique. L'utilisation d'une heuristique permet de réduire considérablement le temps de calcul et de nous rapprocher le plus rapidement possible de la solution optimale, tout en respectant la contrainte du temps réel.

Finalement, nous avons montré l'intégration du planificateur sur deux architectures différentes, ainsi que son implantation et son fonctionnement en temps réel sur un robot.

7

Conclusion et perspective

L'objectif de cette thèse porte sur le contrôle du comportement d'un robot qui évolue dans un environnement humain. Cette problématique soulève de nombreux défis pour tous les aspects de la robotique et notamment pour les composantes décisionnelles. Dans ce manuscrit, nous nous sommes intéressés à la conception et à la réalisation d'un planificateur de tâche pour un robot qui cohabite et coopère avec des humains. Nous avons commencé dans le Chapitre 2 par donner une vue globale sur l'état de l'art en interaction homme-robot, et nous avons présenté plusieurs approches qui traitent du problème de l'interaction homme-robot sur plusieurs niveaux. Nous nous sommes intéressés à la prise en compte de l'homme au niveau décisionnel et nous avons présenté plusieurs modèles conçus pour l'interaction avec des humains. Tous ces modèles utilisent la planification comme moteur central dans la production de plans pour l'interaction homme-robot. Ce chapitre nous a également permis de conclure que la planification délibérative est la mieux adaptée à notre problématique, et cela en se basant sur deux critères : (1) l'autonomie décisionnelle que le planificateur va donner au robot. (2) Une certaine optimalité des plans produits.

Dans le chapitre 3, nous avons pu expliquer notre choix pour la planification HTN. Nous avons donné la description du domaine de planification, mais également la description des règles sociales qui sont une extension de ce domaine. Nous avons pu observer que HATP produit des

plans qui prennent explicitement en compte l'humain, en considérant ses désirs, ses compétences, ses handicaps, etc. Ces plans sont dits socialement acceptables, car ils respectent les règles de comportement des humains. Nous avons également détaillé la métrique utilisée pour la production de plans socialement acceptables et décrit l'algorithme de planification de HATP. Enfin, nous avons conclu que les plans produits sont adaptés à l'interaction homme-robot, mais présentent deux types d'insuffisances : (1) le planificateur ne prend pas en charge les croyances des autres agents pendant le processus de planification. (2) le planificateur n'a aucun contrôle sur la réalisation des tâches en interaction.

Le chapitre 4 apporte la première réponse aux critiques du modèle HATP. En effet, dans ce chapitre, nous avons présenté une extension du modèle HATP permettant la gestion des croyances des différents agents intervenant dans le plan. Nous avons étendu le formalisme de définition des tâches et proposé l'ajout d'actions de communication dans le modèle. Nous avons montré comment utiliser la structure HTN de HATP pour produire des plans qui tiennent compte des différentes croyances des agents, et nous avons noté que l'utilisation de ces structures hiérarchisées n'impliquent pas la modification de l'algorithme de planification, ce qui est un avantage puisqu'à tout moment on peut réutiliser le modèle de HATP classique. Nous avons pu constater l'intérêt de l'inclusion des croyances multiples dans le modèle pour l'interaction homme-robot. Cette inclusion permet la production de plans plus compréhensibles par l'humain, mais également une gestion plus efficace et plus souple des liens croisés. Néanmoins, ce modèle rejoint la critique du modèle HATP faite dans le chapitre 3, du fait qu'il utilise des notions géométriques dans sa représentation symbolique comme, par exemple, la notion de coprésence qui permet une gestion plus réaliste du transfert d'information entre agents.

La seconde réponse aux critiques du modèle HATP est apportée dans le chapitre 5 où nous avons présenté notre approche pour un planificateur hybride, qui combine un planificateur symbolique basé sur un modèle HTN et un planificateur géométrique qui peut fournir des primitives de haut niveau. Cette combinaison n'est pas une nouveauté, puisqu'il existe plusieurs travaux qui traitent de la problématique, comme nous l'avons montré au début de ce chapitre. L'originalité de notre approche réside au niveau de :

- la modélisation des tâches et des contraintes qui combine une association souple entre contrainte et action, et un mécanisme de propagation de contrainte.
- la gestion des backtracks et un mécanisme de révision des contraintes.
- son utilisation pour l'interaction homme-robot.

Nous avons également noté que le modèle présente quelques insuffisances. En effet, le traitement séquentiel des actions durant la phase de planification peut générer des backtracks inutiles. Cela peut être évité en parallélisant le plan durant la phase de planification, ou en mettant en place une procédure qui effectue un ordonnancement efficace. De même, nous avons pu constater que le modèle présente une insuffisance dans le contrôle du comportement du robot pour l'interaction avec des humains. Le modèle est conçu pour le contrôle des configurations (robot et objet) et la gestion de certains états géométriques, mais un des éléments importants

dans l'interaction homme-robot est le mouvement et le déplacement du robot. Il est nécessaire de contrôler les mouvements du robot pour arriver à des comportements et des déplacements agréables pour l'humain.

Les principales perspectives de recherche qui apparaissent à l'issue de cette thèse sont :

Fusion des modèles hybride-HATP et Belief-HATP

Nous pouvons constater que les deux modèles sont complémentaires. En effet, dans le chapitre 4 (HATP avec croyance mutuelle), nous avons pu constater que pour l'échange d'information entre agents, nous utilisons des faits géométriques tels la coprésence. L'utilisation de la planification géométrique facilite grandement la mise à jour des croyances des agents, puisqu'à partir de leurs positions par exemple, le planificateur géométrique peut énumérer tous les objets qui sont dans leurs champs de vision et de ce fait mettre à jour leurs croyances.

De la fusion de ces deux approches va naître une nouvelle définition des plans que nous appellerons "*Plan partagé et coopératif*" (shared cooperative plans), qui va apporter beaucoup à l'interaction homme-robot. En effet, ce type de plan va regrouper toutes les qualités des différentes extensions de HATP.

– Les agents vont avoir un modèle plus riche. Un modèle symbolique où nous pourrons exprimer leurs états à plusieurs niveaux, un modèle géométrique et un modèle de leurs croyances.

– Au niveau géométrique, cela permettra de produire des plans socialement acceptables grâce au modèle flexible des associations action-contrainte, ce qui permet un contrôle du comportement du robot en fonction des situations et des interactions.

– Au niveau symbolique, cela permettra également de produire des plans qui seront socialement acceptables grâce aux règles sociales, mais grâce également à la prise en charge des croyances des humains qui permettent que les plans soient plus compréhensibles. Le modèle Hybride-HATP permet également que la mise à jour des croyances soit effective, et cela en associant aux actions des contraintes qui leur sont adaptées. Prenons l'exemple d'une tâche où un robot doit poser un objet pour que l'humain puisse le prendre. Nous supposons, au niveau de l'action "poser", que dès que l'objet est posé l'humain peut le voir. En utilisant des contraintes géométriques, nous pouvons contraindre la position où l'objet doit être posé (Posé tel que l'objet soit visible pour l'humain).

La gestion du temps

La gestion du temps au niveau de l'algorithme de HATP va agir comme une heuristique et va permettre l'élimination de branches d'exploration plus tôt. Cela permettra également une parallélisation du plan pendant la phase de planification ce qui permettra la détection des indépendances entre tâches. En effet, dans le chapitre 5 pour la planification hybride, nous avons pu observer que du fait du calcul séquentiel du plan, cela génère de nombreux backtracks au niveau géométrique, que nous pouvions éviter avec un plan parallélisé. Un point de départ pour cette extension sont les travaux de [Yorke-Smith 05] et [Castillo 06] qui combinent un algorithme de planification hiérarchique avec un algorithme STN. L'idée dans cette approche est de construire un réseau de contraintes à partir de la structure HTN. Les contraintes sont déduites à partir des durées des actions et des contraintes de précédence, mais également de la hiérarchie puisque les contraintes subies par une tâche sont propagées sur sa descendance.

L'Apprentissage

Il est intéressant d'inclure un processus d'apprentissage sur deux niveaux, apprentissage des règles sociales et apprentissage de nouvelles décompositions sur les méthodes. En effet, dans la mesure où le robot côtoie plusieurs humains différents, il est intéressant qu'il puisse apprendre de nouvelles règles sociales utilisables par HATP (*état indésirable*, *séquences indésirables*, etc.), il doit également pouvoir les appliquer en fonction des humains avec qui le robot interagit. De même pour les méthodes, au fur et à mesure des contacts avec les humains, le robot pourra apprendre de nouvelles manières de réaliser certaines tâches. Par exemple le robot pourra apprendre qu'une manière de transmettre un objet à un humain est de pousser l'objet vers l'humain. L'apprentissage de nouvelles décompositions ne peut pas être dissocié de l'apprentissage de certaines règles sociales. En effet, Dans le chapitre 3 nous avons évoqué la règle *mauvaise décomposition* qui permet de garder une cohérence entre les séquences actions du robot et son intention. Il est donc nécessaire que l'apprentissage de nouvelles décompositions soit lié à l'apprentissage de certaines règles sociales.

Dans la littérature il existe plusieurs algorithmes pour l'apprentissage d'un HTN, nous pouvons citer HTNlearner [Zhuo 09] qui peut faire de l'apprentissage sur des actions sur des méthodes et les préconditions des décompositions dans un environnement partiellement observable, HLS [Qiang Yang 07] qui peut faire de l'apprentissage sur des méthodes et les préconditions des décompositions dans un environnement totalement observable, mais qui intègre la notion de récursivité sur les méthodes. Il existe également HTN-MAKER [Hogg 08] qui fait de l'apprentissage sur des méthodes et les préconditions des décompositions. Pour pouvoir intégrer de l'apprentissage dans notre approche, il faut commencer déjà par déterminer nos besoins, et aussi explorer un peu plus l'état de l'art dans ce domaine.

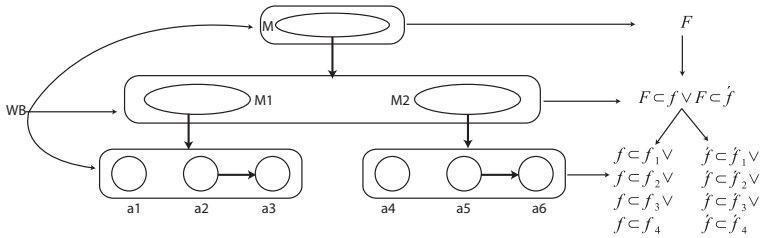


FIGURE 7.1 – Utilisation des méta-effets comme heuristique

Amélioration des Heuristiques

Dans le chapitre 6, nous avons pu constater qu'un des problèmes avec l'algorithme de HATP est l'explosion combinatoire qui vient du fait de l'indépendance de tâches (i.e. des tâches qui ne possèdent pas de liens de précédence entre elles). Il serait intéressant de doter l'algorithme de HATP d'une procédure qui permettra de détecter la dépendance entre tâches. Une piste que nous sommes entrain d'explorer est l'exploitation des "méta-effets" des méthodes. Comme nous l'avons présenté dans le chapitre 3, chaque tâche M_i a un ensemble de "méta-effets" f_{M_i} et chaque décomposition D_i de la tâche a un ensemble de tâches, et de ce fait a également un ensemble de "méta-effets" f_{D_i}. Du fait que l'ensemble des "méta-effets" est un ensemble croissant dans le sens de la hiérarchie, notre hypothèse est que l'ensemble des "méta-effets" f_{M_i} de la tâche doit être inclues dans l'ensemble des "méta-effets" f_{D_i} de la décomposition. L'idée est résumée par la figure 7.1. Pour chaque séquence de tâches de même niveau, un test d'inclusion de "méta-effets" est effectué avec le niveau qui le précède. Nous pensons que cette procédure va permettre de détecter la dépendance entre tâches, ce qui va avoir pour effet de réduire l'espace de recherche.

Références bibliographiques

[**Ambite 98**] J.L. Ambite. *Planning by rewriting.* PhD thesis, university of southern California, 1998.

[**Ambros-Ingerson 90**] J. A. Ambros-Ingerson & S. Steel. *Integrating Planning, Execution and Monitoring.* Dans J. Allen, J. Hendler & A. Tate, editeurs, Readings in Planning, pages 735–740. Kaufmann, San Mateo, CA, 1990.

[**Bailly 01**] G. Bailly. *Audiovisual Speech Synthesis.* International Journal of Speech Technology, vol. 6, pages 6–331, 2001.

[**Bäckström 93**] Christer Bäckström & Bernhard Nebel. *Complexity Results for SAS+ Planning.* Computational Intelligence, vol. 11, pages 625–655, 1993.

[**Belta 07**] C. Belta, A. Bicchi, M. Egerstedt, E. Frazzoli, E. Klavins & G. J. Pappas. *Symbolic Planning and Control of Robot Motion : State of the Art and Grand Challenges.* Robotics and Automation Magazine, vol. 14, pages 61–70, 2007.

[**Ben-Mena 00**] Sami Ben-Mena. *Introduction aux méthodes multicritères d'aide à la décision.* Biotechnologie, Agronomie, Société et Environnement, vol. 4(2), pages 83–93, 2000.

[**BEYNIER .A 08**] SZER .D. MOUADDIB . A.-I. BEYNIER .A CHARPILLET .F. *Chapitre 4. DEC-MDP/POMDP.* Processus décisionnels de Markov en intelligence artificielle, vol 2, METHODES AVANCEES ET APPLICATIONS TRAITE IC2 SERIE INFO, éditeurs O. Buffet et O. Sigaud, 2008, Hermès, 2008.

[**Beynier 06**] Aurélie Beynier. *Une contribution à la résolution des Processus Décisionnels de Markov Décentralisés avec contraintes temporelles.* PhD thesis, l'Université de Caen Basse-Normandie, 2006.

[**Blythe 96**] Jim Blythe. *A Representation for Efficient Planning in Dynamic Domains with External Events.* Dans in the AAAI workshop on &quot ;Theories of Action, Planning and Control : Bridging the, 1996.

[**Bouguerra 04**] Abdelbaki Bouguerra & Lars Karlsson. *Hierarchical Task Planning under Uncertainty.* Dans In 3rd Italian Workshop on Planning and Scheduling (AI*IA, 2004.

[**Bouguerra 07**] A. Bouguerra, L. Karlsson & A. Saffiotti. *Semantic Knowledge-Based Execution Monitoring for Mobile Robots.* Dans Robotics and Automation, 2007 IEEE International Conference on, pages 3693 –3698, apr. 2007.

[**Breazeal 04**] C. Breazeal, G. Hoffman & A. Lockerd. *Teaching and Working with Robots as a Collaboration.* Dans International Conference of Autonomous Agents and Multiagent Systems (AAMAS), 2004.

[**Brenner 03**] Michael Brenner. *MAPL : a Framework for multiagent planning with partially ordered temporal plans.* Dans IJCAI'03 : Proceedings of the 18th international joint conference on Artificial intelligence, pages 1513–1514, San Francisco, CA, USA, 2003. Morgan Kaufmann Publishers Inc.

[**Brenner 08**] Michael Brenner & Ivana Kruijff-Korbayova. *A Continual Multiagent Planning Approach to Situated Dialogue.* Dans Proceedings of the 12th Workshop on the Semantics and Pragmatics of Dialogue (LonDial 2008), 2008.

[**Brenner 09**] Michael Brenner & Bernhard Nebel. *Continual Planning and Acting in Dynamic Multiagent Environments.* Journal of Autonomous Agents and Multiagent Systems, vol. 19, pages 297–331, 2009.

[**Bresina 05**] John L. Bresina, Ari K. Jónsson, Paul H. Morris & Kanna Rajan. *Mixed-Initiative Activity Planning for Mars Rovers.* Dans Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, pages 1709–1710, Edinburgh, Scotland, UK, July 30-August 5 2005.

[**Broz 08**] Frank Broz, Illah Nourbakhsh & Reid Simmons. *Planning for human-robot interaction using time-state aggregated POMDPs.* Dans AAAI'08 : Proceedings of the 23rd national conference on Artificial intelligence, pages 1339–1344. AAAI Press, 2008.

[**Burger 08**] B. Burger, I. Ferrane & F. Lerasle. *Multimodal Interaction Abilities for a Robot Companion.* Dans ICVS, pages 549–558, 2008.

[**Burstein 03**] et J. Allen Burstein G. Ferguson. *Integrating agent-based mixed-initiative control with an existing multi-agent planning system.* Rapport technique, The University of Rochester Computer Science Department Rochester, New York, 2003.

[**Caldiran 10**] Ozan Caldiran, Kadir Haspalamutgil, Abdullah Ok, Can Palaz, Esra Erdem & Volkan Patoglu. *From Discrete Task Plans to Continuous Trajectories.* Dans In Proc. of the 19th International Conference on Automated Planning and Scheduling (ICAPS'09) Workshop, 2010.

[**Cambon 05**] S. Cambon. *Planifier avec les contraintes géométriques du mouvement et de la manipulation.* PhD thesis, Université de Toulouse, 2005.

[**Castillo 06**] Luis A. Castillo, Juan Fernández-Olivares, Óscar García-Pérez & Francisco Palao. *Efficiently Handling Temporal Knowledge in an HTN Planner.* Dans Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling, ICAPS 2006,, pages 63–72, Cumbria, UK., June 2006.

[**Chanthery 05**] E. Chanthery. *Planification de mission pour un véhicule aérien autonome.* PhD thesis, ENSAE, 2005.

[**Choi 09**] Jaesik Choi & Eyal Amir. *Combining planning and motion planning.* Dans ICRA'09 : Proceedings of the 2009 IEEE international conference on Robotics and Automation, pages 4374–4380, Piscataway, NJ, USA, 2009. IEEE Press.

[**Cirillo 10**] Marcello Cirillo. *Planning in Inhabited Environments : Human-Aware Task Planning and Activity Recognition.* PhD thesis, Örebro university Sweden, 2010.

[**Clement 03**] Bradley J. Clement & Anthony C. Barrett. *Continual coordination through shared activities.* Dans AAMAS '03 : Proceedings of the

second international joint conference on Autonomous agents and multiagent systems, pages 57–64, New York, NY, USA, 2003. ACM.

[Clodic 06] A. Clodic, S. Fleury, R. Alami, R. Chatila, G. Bailly, L. BrÃ¨thes, M. Cottret, P. DanÃ¨s, X. Dollat, F. EliseÃ⁻, I. FerranÃ©, M. Herrb, G. Infantes, C. Lemaire, F. Lerasle, J. Manhes, P. Marcoul, P. Menezes & V. Montreuil. *Rackham : An interactive Robot-Guide*. Dans IEEE International Symposium on Robot and Human Interactive Communication, Septembre 2006.

[Clodic 07] A. Clodic. *Supervision pour un robot interactif : action et interaction pour un robot autonome en environnement humain*. PhD thesis, Université Paul Sabatier Toulouse III, 2007.

[Clodic 09] Aurélie Clodic, Hung Cao, Samir Alili, Vincent Montreuil, Rachid Alami & R. Chatila. *SHARY : A Supervision System Adapted to Human-Robot Interaction*. Dans Oussama Khatib, Vijay Kumar & George Pappas, editeurs, Experimental Robotics, volume 54 of *Springer Tracts in Advanced Robotics*, pages 229–238. Springer Berlin / Heidelberg, 2009.

[Cohen 90] P. R. Cohen & H. J. Levesque. *Intention is Choice with Commitment*. Artificial Intelligence, vol. 42, pages 213–361, 1990.

[Cohen 91] P. R. Cohen & H. J. Levesque. *Teamwork*. Nous, vol. 25(4), pages 487–512, 1991.

[Conner 07] David C. Conner, Hadas Kress-gazit, Howie Choset & Alfred A. Rizzi. *Valet parking without a valet*. Dans In IEEE/RSJ Int. Conf. on Intelligent Robots & Systems, 2007.

[Cortellessa 08] Scopelliti M. Tiberio L. Koch Svedberg G.-Loutfi A. Cortellessa G. & F. Pecora. *A Cross-Cultural Evaluation of Domestic Assistive Robots*. Dans Proceedings of AAAI Fall Symposium on AI in Eldercare : New Solutions to Old Problems, Arlington, VA, USA, November 2008.

[Dousson 93] C. Dousson, P. Gaborit & M. Ghallab. *Situation Recognition : Representation and Algorithms.* In proc. of the 13th IJCAI, pages 166–172, 1993.

[Erol, K. 94] Erol, K., Hendler, J. & Nau, D. S. *HTN planning : Complexity*

and expressivity. Dans In Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94), volume 2, pages 1123–1128, 1994.

[**Fainekos 07**] Georgios E. Fainekos, Antoine Girard & Hadas Kress-gazit. *Temporal logic motion planning for dynamic mobile robots*. Rapport technique, Department of Computer and Information Science, Univ. of Pennsylvania, 2007.

[**Fikes 71**] R.E. Fikes & N.J. Nilsson. *STRIPS : A New Approach to the Application of Theorem Proving to Problem Solving*. Artificial Intelligence, vol. 2, pages 189–208, 1971.

[**Flavell 92**] John H. Flavell. *Perspectives On Perspective-Taking*. Dans H. Beilin and P.B. Pufall Eds. Piaget's Theory : Prospects and possibilities. The Jean Piaget Symposium series, pages 107–139, Hillsdale, NJ Erlbaum, 1992.

[**Fong 03**] Terrence Fong, Illah Nourbakhsh & Kerstin Dautenhahn. *A survey of socially interactive robots*. Robotics and Autonomous Systems, vol. 42, pages 143 – 166, 2003.

[**Fong 06**] T. Fong, C. Kunz, L. M. Hiatt & M. Bugajska. *The human-robot interaction operating system*. Dans HRI '06 : Proceeding of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction, pages 41–48, New York, NY, USA, 2006. ACM Press.

[**Forman 01**] Ernest Forman & Mary Ann Selly. Decision by objectives (how to convince others that you are right). World Scientific, 2001.

[**Fox 03**] Maria Fox & Derek Long. *PDDL2.1 : An extension to PDDL for expressing temporal planning domains*. Journal of Artificial Intelligence Research, vol. 20, pages 61–124, 2003.

[**Galindo 04**] C. Galindo, J. Gonzalez & J.A. Fernandez-Madrigal. *Interactive task planning in assistant robotics*. Dans Proceedings of International IFAC Symposium of Intelligent Autonomous Vehicules (IAV), Juillet 2004.

[**Ghallab 04**] M. Ghallab, D. Nau & P. Traverso. Automated planning - theory and practice. Morgan Kaufmann Publishers, ed. Elvesier, 2004.

[**Gockley 06**] R. Gockley, J. Forlizzi & R. Simmons. *Interactions with a moody robot.* Dans HRI '06 : Proceeding of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction, pages 186–193. ACM Press, 2006.

[**Gottschlich 93**] Susan Gottschlich, Carlos Ramos & Damian Lyons. *Assembly and Task Planning : A Taxonomy and Annotated Bibliography.* IEEE Robotics Automation Magazine, vol. 2, pages 4–12, 1993.

[**Gravot 04**] F. Gravot. *aSyMov : Fondation d'un planificateur robotique intégrant le symbolique et le géométrique.* PhD thesis, Université de Toulouse, 2004.

[**Gravot 05**] Fabien Gravot, Stephane Cambon & Rachid Alami. *aSyMov :A Planner That Deals with Intricate Symbolic and Geometric Problems.* Robotics Research, vol. 15, pages 100–110, 2005.

[**Grosz 96**] B. J. Grosz & S. Kraus. *Collaborative Plans for Complex Group Action.* Artificial Intelligence, vol. 86, pages 269–358, 1996.

[**Grosz 98**] Barbara J. Grosz & Sarit Kraus. *The Evolution of SharedPlans.* Dans IN A. RAO AND M. WOOLDRIDGE, FOUNDATIONS AND THEORIES OF RATIONAL AGENCY, pages 227–262. Kluwer Academic Publishers, 1998.

[**Guitton 09**] J. Guitton & J.-L. Farges. *Taking into account geometric constraints for task-oriented motion planning.* Dans Workshop on Bridging the Gap between Task and Motion Planning, pages 26–33, 2009.

[**Guitton 10**] Julien Guitton. *Architecture hybride pour la planifcation d'actions et de déplacements.* PhD thesis, Université de Toulouse, 2010.

[**Guéré 01**] E. Guéré & R. Alami. *Let's reduce the gap between task planning and motion planning.* Dans IEEE International Conference on Robotics and Automation, pages 15–20, 2001.

[**Hart 68**] Peter Hart, Nils Nilsson & Bertram Raphael. *A Formal Basis for the Heuristic Determination of Minimum Cost Paths.* IEEE Transactions on Systems Science and Cybernetics, vol. 4, pages 100–107, 1968.

[**Hiolle 09**] A. Hiolle, K.A. Bard & L. Canamero. *Assessing human reactions to different robot attachment profiles.* Dans Robot and Human Interactive Communication, 2009. RO-MAN 2009. The 18th IEEE International Symposium on, pages 251 –256, sep. 2009.

[**Hoffmann 02**] Jörg Hoffmann. *Extending FF to Numerical State Variables.* Dans Proceedings of the 15th European Conference on Artificial Intelligence (ECAI-02), 2002.

[**Hogg 08**] Chad Hogg, Héctor Muñoz-avila & Ugur Kuter. *HTN-MAKER : Learning HTNs with minimal additional knowledge engineering required.* Dans Proceedings of the TwentyThird Conference on Artificial Intelligence. AAAI Press, 2008.

[**Huettenrauch 06**] H. Huettenrauch, K.S. Eklundh, A. Green & E.A. Topp. *Investigating Spatial Relationships in Human-Robot Interaction.* Dans Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on, pages 5052 –5059, 9-15 2006.

[**Huttenrauch 02**] H. Huttenrauch & K. Severinson-Eklund. *Fetch-and-carry with CERO : observations from a long-term user study.* Dans Proceedings of the IEEE International Workshop on Robot and Human Communication (RO-MAN), 2002.

[**Ilghami 03**] O. Ilghami & D. Nau. *A General Approach to Synthesize Problem-Specific Planners.* Rapport technique CS-TR-4597, Department of Computer Science, University of Maryland, Octobre 2003.

[**Ingrand 96**] F. F. Ingrand, R. Chatila, R. Alami & F. Robert. *PRS : A High Level Supervision and Control Language for Autonomous Mobile Robots.* Dans Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), pages 43–49, Minneapolis, USA, 1996.

[**Karami 10**] Abir-Beatrice Karami, Laurent Jeanpierre & Abdel-Illah Mouaddib. *Human-robot collaboration for a shared mission.* Dans HRI '10 : Proceeding of the 5th ACM/IEEE international conference on Human-robot interaction, pages 155–156, New York, NY, USA, 2010. ACM.

[**Kennedy 07**] W.G. Kennedy, M. D. Bugajska, M. Marge, W. Adams, B. R. Fransen, D. Perzanowski, A. C. Schultz & J. G. Trafton. *Spatial*

Representation and Reasoning for Human-Robot Collaboration. Dans Proceedings of AAAI Conference, pages 1554–1559, 2007.

[**Knoblock 95**] Craig A. Knoblock. *Planning, Executing, Sensing, and Replanning for Information Gathering*. Dans IN PROCEEDINGS OF THE FOURTEENTH INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE, 1995.

[**Koay 07**] K.L. Koay, E. A. Sisbot, D. A. Syrdal, M.L. Walters, K. Dautenhahn & R. Alami. *Exploratory Study of a Robot Approaching a Person in the Context of Handling Over an Objec*. Dans International Conf. on Artificial Intelligence, volume Spring Symposia AAAI, 2007. Palo Alto, California, USA.

[**Kruijff 07**] G.J.M. Kruijff & M. Brenner. *Modelling Spatio-Temporal Comprehension in Situated Human-Robot Dialogue as Reasoning about Intentions and Plans*. Dans Proceedings of the Symposium on Intentions in Intelligent Systems, Stanford University, Palo Alto, CA, USA, March 2007. AAAI Spring Symposium Series 2007.

[**Kvarnstrom 01**] J. Kvarnstrom & P. Doherty. *TALplanner : A temporal logic based forward chaining planner*. Annals of Mathematics and Artificial Intelligence, vol. 30, pages 119–169, 2001.

[**Laugier 85**] C. Laugier & J. Troccaz. *Sharp, a system for automatic programming of manipulation robots*. Dans Proceedings of the 3rd International Symposium on Robotics Research, 1985.

[**LaValle 98**] S. M. LaValle. *Rapidly-exploring Random Trees : A New Tool for Path Planning*. Rapport technique, Computer Science Dept., Iowa State University, 1998.

[**Levesque 90**] H. J. Levesque, P. R. Cohen & J. H. T. Nunes. *On Acting Together*. Dans Proceedings of the Annual Meeting of the American Association for Artificial Intelligence (AAAI), 1990.

[**Lewis 02**] David Lewis. Convention : A philosophical study. Wiley-Blackwell, May 2002.

[**Lochbaum 98**] K. E. Lochbaum. *A collaborative planning model of intentional structure*. Comput. Linguist., vol. 24, pages 525–572, 1998.

[**Lozano-Pérez 87**] J. L. Mazer E. O'Donnell-P. A. Grimson W. E. L. Tournassoud P. Lozano-Pérez T. Jones & A. Lanusse. *Handey : A robot system that recognizes, plans and manipulates.* Dans IEEE Int. Conf. on Robotics and Automation, pages 843–849, 1987.

[**Luis F. Marin-Urias 08**] Emrah Akin Sisbot Luis F. Marin-Urias & Rachid Alami. *Geometric tools for Perspective Taking for Human-Robot Interaction.* Dans Proceedings of the 7th Mexican International Conference on Artificial Intelligence, 2008.

[**Marin-Urias 08**] Luis F. Marin-Urias, E. Akin Sisbot & Rachid Alami. *Geometric Tools for Perspective Taking for Human-Robot Interaction.* Dans 7th International Conference on Artificial Intelligence, 2008.

[**Mazon 90**] Alami R. Mazon I. & P. Violero. *Automatic planning of pick and place operations in presence of uncertainties.* Dans Intelligent Robots and Systems (IROS'90), pages 33 – 40, 1990.

[**McDermott 98**] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld & D. Wilkins. *PDDL - The Planning Domain Definition Language.* Technical Report CVC TR-98-003/DCS TR-1165 1.2, Yale Center for Computational Vision and Control, 1998.

[**Moll 06**] Henrike Moll & Michael Tomasello. *Level 1 perspective-taking at 24 months of age.* British Journal of Developmental Psychology, vol. 24, pages 603–613, 2006.

[**Montreuil 08**] Vincent Montreuil. *Intéraction decisionelle homme-robot : La planification de tâches au service de la sociabilité du robot.* PhD thesis, Université Toulouse III - Paul Sabatier, 2008.

[**Myers 97**] K. Myers. *Abductive Completion of Plan Sketches.* Dans Proceedings of the Fourteenth National Conference on Artificial Intelligence. AAAI Press, 1997.

[**Myers 02**] K. L. Myers, W. M. Tyson, M. J. Wolverton, P. A. Jarvis, T. J. Lee & M. Desjardins. *PASSAT : A User-centric Planning Framework.* Dans Proceedings of the Third International NASA Workshop on Planning and Scheduling for Space, Octobre 2002.

[**Myers 03**] K. L. Myers, P. Jarvis, W. M. Tyson & M. Wolverton. *A Mixed-initiative Framework for Robust Plan Sketching.* Dans

Proceedings of the International Conference on Automated Planning ans Scheduling, pages 256–266. AAAI, 2003.

[**Nau 03**] D. Nau, T.-C. Au, O. Ilghami, U. Kuter, J. W. Murdock, D. Wu & F. Yaman. *SHOP2 : An HTN Planning System.* Artificial Intelligence Research, vol. 20, pages 380–404, 2003.

[**Nourbakhsh 05**] Illah R. Nourbakhsh & Terrence Fong. *Human-robot teams on the moon : Challenges and plans (extended abstract).* Dans International Conference on Robotics and Automation (ICRA), Workshop on Fielding Multi-Robot Systems, 2005.

[**Pacchierotti 05**] E. Pacchierotti, H. I. Christensen & P. Jensfelt. *Human-Robot Embodied Interaction in Hallway Settings : a Pilot User Study.* Dans 14th IEEE International Workshop on Robot and Human Interactive Communication (RO-MAN), 2005.

[**Pandey 09**] A.K. Pandey & R. Alami. *A framework for adapting social conventions in a mobile robot motion in human-centered environment.* Dans International Conference Advanced Robotics. ICAR 2009., pages 1 –8, jun. 2009.

[**Pednault 94**] E. Pednault. *ADL and the State-Transition Model of Action.* Journal of logic and computation, vol. 4, pages 467–512, 1994.

[**Pollack 99**] M. E. Pollack & J. F. Horty. *There's more to life than making plans : Plan management in dynamic environments.* AI Magazine, vol. 20, pages 71– 84, 1999.

[**Pollack 02**] M. Pollack, C. McCarthy, S. Ramakrishnan, I. Tsamardinos, L. Brown, S. Carrion, D. Colbry, C. Orosz & B. Peintner. *Autominder : A Planning, Monitoring, and Reminding Assistive Agent.* Dans Seventh International Conference on Intelligent Autonomous Systems., 2002.

[**Pollack 03**] M. E. Pollack, L. Brown, D. Colbry, C. E. McCarthy, C. Orosz, B. Peintner, S. Ramakrishnan & I. Tsamardinos. *Autominder : An Intelligent Cognitive Orthotic System for People with Memory Impairment.* Robotics and Autonomous Systems, vol. 44, no. 3-4, pages 273 – 282, 2003. Best papers presented at IAS-7.

[**Qiang Yang 07**] Rong Pan Qiang Yang & Sinno Jialin Pan. *Learning Recursive*

HTN-Method Structures for Planning. Dans In Proceedings of the ICAPS-07 Workshop on AI Planning and Learning, pages 22–26, Providence, Rhode Island, USA, September 2007.

[**Rich 97**] C. Rich & C. L. Sidner. *COLLAGEN : when agents collaborate with people.* Dans Proceedings of the first international conference on Autonomous agents (AGENTS), pages 284–291. ACM, 1997.

[**Rich 98**] C. Rich & C. L. Sidner. *COLLAGEN : A Collaboration Manager for Software Interface Agents.* User Modeling and User-Adapted Interaction, vol. 8, no. 3-4, pages 315–350, 1998.

[**Saaty 00**] T.L. Saaty. Fundamentals of the analytic hierarchy process. RWS Publications, 4922 Ellsworth Avenue, Pittsburgh, PA 15413, 2000.

[**Saint Aimé 10**] Sébastien Saint Aimé, Céline Jost, Brigitte Le Pévédic & Dominique Duhaut. *Dynamic behaviour conception for EmI companion robot.* Dans 41st International Symposium on Robotics – ISR 2010 41st International Symposium on Robotics – ISR 2010, page 8, Munich Germany, 2010.

[**Scerri 03**] P. Scerri, D. V. Pynadath, L. Johnson, P. Rosenbloom, N. Schurr, M. Si & M. Tambe. *A Prototype Infrastructure for Distributed Robot-Agent-Person Teams.* Dans The Second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS), pages 433–440, New York, NY, USA, 2003. ACM.

[**Scerri 04**] P. Scerri, D. Pynadath, N. Schurr, A. Farinelli, S. Gandhe & M. Tambe. *Team Oriented Programming and Proxy Agents : The Next Generation.* Dans In Proceedings of 1st international workshop on Programming Multiagent Systems, 2004.

[**Sebanz 06**] N. Sebanz, H. Bekkering & G. Knoblich. *Joint action : bodies and minds moving together.* Trends in Cognitive Sciences, vol. 10, no. 2, pages 70–76, February 2006.

[**Shiomi 06**] Masahiro Shiomi, Takayuki K, A Hiroshi Ishiguro & Norihiro Hagita. *Interactive humanoid robots for a science museum.* Dans In Proc. Human robot Interaction HRI'06, pages 305–312. ACM Press, 2006.

[**Sidner 03a**] C. L. Sidner & C. Lee. *An Architecture for Engagement in*

Collaborative Conversations between a Robot and Humans. Rapport technique TR2003-13, Mitsubishi Electric Research Labs, Avril 2003.

[**Sidner 03b**] C. L. Sidner, C. Lee & N. Lesh. *Engagement Rules for Human-Robot Collaborative Interaction*. Dans Proceedings of the international conference on Systems, Man and Cybernetics (SMC), volume 4, pages 3957–3962, Octobre 2003.

[**Sisbot 05**] E. A. Sisbot, R. Alami, T. Simeon, K. Dautenhahn, M. Walters, S. Woods, K. L. Koay & C. Nehaniv. *Navigation In Presence of Humans*. Dans Proceedings of IEEE-RAS International Conference on Humanoid Robots (Humanoids), 2005.

[**Sisbot 06**] E. A. Sisbot, A. Clodic, L. F. Marin Urias, M. Fontmarty, L. Brethes & R. Alami. *Implementing a Human-Aware Robot System*. Dans Proceedings of the 15th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN), Septembre 2006.

[**Sisbot 07a**] E. A. Sisbot, L. F. Marin & R. Alami. *Spatial Reasoning for Human Robot Interaction*. Dans Proceedings of International Conference on Robots and Systems (IROS), 2007.

[**Sisbot 07b**] E. Akin Sisbot, Luis F. Marin Urias, Rachid Alami & Thierry Siméon. *Spatial Reasoning for Human-Robot Interaction*. Dans IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, USA, November 2007.

[**Sisbot 07c**] E.A. Sisbot, L.F. Marin-Urias, R. Alami & T. Simeon. *A Human Aware Mobile Robot Motion Planner*. IEEE Transactions on Robotics, vol. 23, no. 5, pages 874–883, october 2007.

[**Sisbot 08**] Emrah Akin Sisbot. *Towards Human-Aware robot Motions*. PhD thesis, LAAS/CNRS, Universite Paul Sabatier, October 2008.

[**Sisbot 10**] Emrah Akin Sisbot, Luis F. Marin-Urias, Xavier Broquere, Daniel Sidobre & Rachid Alami. *Synthesizing Robot Motions Adapted to Human Presence*. International Journal of Social Robotics, vol. 2, no. 3, pages 329–343, 2010.

[**Tambe 97**] M. Tambe. *Towards Flexible Teamwork*. Journal of Artificial Intelligence Research, vol. 7, pages 93–124, 1997.

[**Trafton 05**] J. Trafton, N. L. Cassimatis, M. D. Bugajska, D. P. Brock, F. E. Mintz & A. C. Schultz. *Enabling effective human-robot interaction using perspective-taking in robots*. Dans Proceedings of IEEE transactions on Systems, Man and Cybernetics, pages 460–470, Novembre 2005.

[**van Vugt 07**] Henriette C. van Vugt, Elly A. Konijn, Johan F. Hoorn, I. Keur & Anton Eliëns. *Realism is not all ! User engagement with task-related interface characters*. Interacting with Computers, vol. 19, no. 2, pages 267–280, 2007.

[**Wolfe 10**] Jason Wolfe, Bhaskara Marthi & Stuart Russell. *Combined Task and Motion Planning for Mobile Manipulation*. Dans International Conference on Automated Planning and Scheduling, Toronto, Canada, May 2010.

[**Yorke-Smith 05**] Neil Yorke-Smith. *Exploiting the Structure of Hierarchical Plans in Temporal Constraint Propagation*. Dans Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, pages 1223–1228, Pittsburgh, Pennsylvania, USA, July 2005.

[**Zhuo 09**] Hankz Hankui Zhuo, Derek Hao Hu, Chad Hogg, Qiang Yang & Hector Munoz-Avila. *Learning HTN method preconditions and action models from partial observations*. Dans Proceedings of the 21st international jont conference on Artifical intelligence, pages 1804–1809, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.