Planification et contrôle d'exécution temporels : IxTeT-eXeC*

Interleaving Temporal Planning and Execution : IxTeT-eXeC

Solange Lemai Félix Ingrand
LAAS/CNRS
7 Avenue du Colonel Roche, F-31077 Toulouse Cedex 04, France
{slemai,felix}@laas.fr

Résumé

Nous nous intéressons à la planification de missions et au contrôle de l'exécution des plans pour des robots mobiles autonomes évoluant dans des environnements dynamiques. Dans un tel contexte, il est souhaitable d'utiliser des processus de planification et de contrôle d'exécution qui interagissent tout en tenant compte explicitement du temps (représentation et raisonnement). Nous proposons une approche pour intégrer planification délibérative, réparation de plan et contrôle d'exécution. Elle est basée sur des techniques de planification temporelle dans l'espace des plans partiels. Ces techniques produisent des plans flexibles (partiellement ordonnés et partiellement instanciés) et permettent, sous certaines conditions présentées dans ce papier, d'effectuer en parallèle l'exécution d'un plan et sa réparation. Cette approche a été implémentée en utilisant le planificateur temporel IxTeT, intégrée dans l'architecture LAAS en interaction avec un exécutif procédural et testée sur une plate-forme robotique (ATRV).

Mots Clef

Planification, Architectures et intégration de systèmes, Robotique.

Abstract

Execution control of plans is a very active domain of research, but remains a major challenge when performed on board real autonomous mobile robots. In such a context, where execution concurrency, resources contention and environment dynamic characterize the domain, the use of a temporal planner and a temporal executive whose processes are interleaved is desirable. In this paper, we propose a framework to integrate deliberative planning, plan repair and execution control in a dynamic environment with real-time constraints. It is based on lifted partial order temporal planning techniques which produce flexible plans and allow, under certain conditions discussed in the paper, plan repair interleaved with plan execution. This framework has been implemented using the planning system IxTeT, integrated in the LAAS architecture in interaction with a procedural executive, tested in simulation and deployed on an ATRV robotic platform.

Keywords

Planning, Architectures and systems integration, Robotics.

1 Introduction

Le contrôle d'exécution de plans est un problème particulièrement difficile lorsqu'il doit être effectué à bord de systèmes autonomes tels que des robots ou des satellites. Un tel système doit disposer de processus de délibération pour générer des plans qui réalisent les buts de la mission tout en respectant des délais et des contraintes de ressources sur le long terme. Il doit également disposer de mécanismes d'adaptation des plans au cours de leur exécution.

Dans ce papier, nous proposons une approche pour intégrer planification délibérative, réparation de plan et contrôle d'exécution dans un environnement dynamique avec des contraintes temporelles. Elle est basée sur deux composants en interaction : un planificateur et un exécutif qui tiennent compte explicitement du temps (représentation et raisonnement).

Le planificateur produit un plan complet. Ce plan est ensuite déroulé par l'exécutif temporel selon un cycle "Perception/Réparation de plan/Action" : intégrer les messages externes, réparer le plan si nécessaire, décider des actions à exécuter. Ce cycle permet d'être réactif aux échecs et dépassements de délai du système contrôlé et d'adapter le plan en conséquence. Réparer et exécuter le plan en parallèle est intéressant si certaines parties du plan restent valides et exécutables (branches parallèles du plan indépendantes des actions échouées), et si le plan est temporellement flexible, permettant ainsi de retarder ou insérer des actions. Notre approche nécessite un planificateur avec les propriétés suivantes :

– une représentation temporelle décrivant le monde comme un ensemble de variables d'état fonctions du temps,
– la recherche s'effectue dans l'espace des plans partiels,
– les plans générés sont flexibles et basés sur des CSP[1], en

*Travail réalisé en collaboration avec le CNES et ASTRIUM.

[1]Nous utilisons ici le terme CSP, qui originellement signifie :

particulier le réseau temporel repose sur un STN[2] [Dechter, Meiri, & Pearl 1989].

Cette approche a été implémentée dans le système IxTeT-eXeC en utilisant le planificateur IxTeT et un nouveau composant : l'exécutif TeXeC. IxTeT-eXeC a été intégré dans le niveau décisionnel de l'architecture LAAS pour contrôler un robot mobile autonome.

Plusieurs études ont déjà traité du problème de l'interaction entre les processus de planification et de contrôle d'exécution. Certaines reposent sur des exécutifs procéduraux auxquels ont été ajoutés quelques mécanismes délibératifs : l'expansion de buts dans le cas de TDL [Simmons & Apfelbaum 1998], l'anticipation des choix à venir par simulation des exécutions futures dans le cas de PropicePlan [Despouys & Ingrand 1999].

Parmi les approches basées principalement sur la délibération, les techniques de planification dans l'espace des plans partiels sont fréquemment utilisées. Ainsi IPEM [Ambros-Ingerson & Steel 1988], en planification classique, et Sage [Knoblock 1995], dans un contexte de recherche d'information, adaptent un algorithme de recherche non linéaire pour que l'exécution fasse partie intégrante du processus de planification.

Plusieurs approches ont été développées pour des plateformes robotiques. Le système ROGUE [Haigh & Veloso 1998] planifie des buts asynchrones et contrôle l'exécution des plans. Dans [Beetz 2000] par contre, les auteurs proposent une approche basée sur des processus d'adaptation de plan, spécifiés par les plans eux-mêmes sous forme de sous-plans.

Mais finalement, peu d'approches tiennent compte explicitement du temps. Le système Autominder aborde surtout le problème de l'exécution réactive d'un plan temporellement flexible. Dans [McCarthy & Pollack 2002], les auteurs proposent une représentation temporelle riche et des procédures locales de réparation de plan en cas d'échec à l'exécution. Le système CASPER [Chien *et al.* 2000] effectue une planification "continue", entrelacée avec l'exécution. Le planificateur propage des mises à jour régulières de l'état, des ressources et des instants. Les conflits sont résolus incrémentalement grâce à des techniques de réparation itérative. Notons que cette approche repose sur des plans complètement ordonnés et instanciés et qu'elle ne permet pas de réagir aux mises à jour intervenant pendant la réparation. Avec IDEA [Muscettola *et al.* 2002], les auteurs proposent une intégration continue de la planification temporelle et du contrôle d'exécution. Cette approche repose sur deux principes : (1) la plupart des composants peuvent être représentés comme des agents partageant une machine virtuelle commune qui définit leur comportement en planification réactive (la planification devant être comprise ici dans un sens large) (2) ces agents partagent des portions d'un modèle temporel global spécifiant leur comportement interne ainsi que la communication entre eux. Cette approche semble prometteuse, mais ne permet pas pour l'instant la gestion de ressources.

L'article est organisé comme suit. La section 2 présente l'organisation globale du système : l'intégration d'IxTeT-eXeC dans l'architecture LAAS et plus précisément ses interactions avec l'exécutif procédural. Après une brève description du planificateur sous-jacent, la section 3 détaille notre approche de replanification dynamique en s'intéressant particulièrement aux problèmes soulevés par l'exécution et la réparation simultanées d'un même plan. La dernière section présente un exemple illustratif ainsi que les premières expérimentations effectuées sur un robot mobile.

2 Organisation globale

Cette section présente l'architecture dans laquelle s'insère IxTeT-eXeC, puis explicite la répartition du contrôle d'exécution entre IxTeT-eXeC et un exécutif procédural.

2.1 L'architecture LAAS

Le développement de systèmes embarqués complexes tels que des robots ou des satellites est généralement basé sur une architecture et des outils associés. Dans notre cas, nous intégrons IxTeT-eXeC dans l'architecture LAAS [Alami *et al.* 1998]. Cette architecture a initialement été conçue pour des robots mobiles autonomes. Elle repose cependant sur des principes assez généraux, et un ensemble d'outils et de méthodologies servent de support à la conception, l'intégration et la validation de ces systèmes. L'architecture

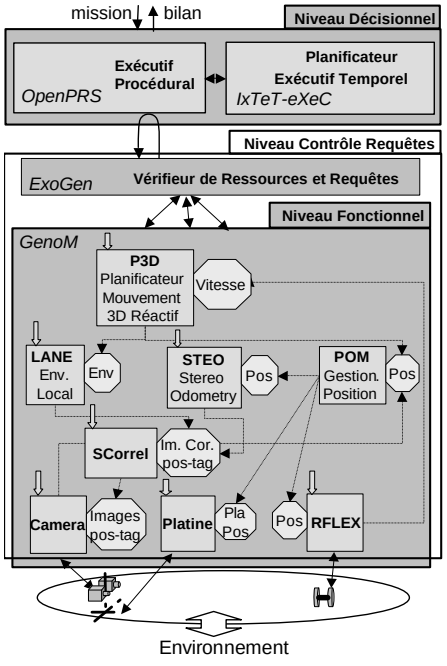


FIG. 1 – Une instance de l'architecture LAAS.

Constraint Satisfaction Problem, pour désigner le ou les réseaux de contraintes utilisés pour la satisfaction des contraintes entre les variables d'un plan.

[2]Un *Simple Temporal Network* peut être représenté par un graphe orienté de contraintes dont les noeuds sont les variables temporelles et les arêtes $i \rightarrow j$ représentent les contraintes de durée $(t_j - t_i)$. Une arête est étiquetée par un intervalle numérique continu.

comporte trois niveaux. Chaque niveau obéit à des contraintes temporelles différentes et utilise des représentations de données différentes. La Figure 1 décrit un exemple d'utilisation de cette architecture avec une instance particulière du niveau fonctionnel.

Le **niveau décisionnel** intègre les capacités délibératives de l'agent : produire des plans d'actions, reconnaître des situations, détecter des fautes, etc. Dans notre cas, il comprend :

– un *exécutif procédural* (OpenPRS [Ingrand *et al.* 1996]) qui est connecté au niveau inférieur auquel il envoie des requêtes qui vont lancer des actions (capteurs / actionneurs) ou démarrer des traitements. Il est responsable de la supervision des actions tout en étant réactif aux événements provenant du niveau inférieur et aux commandes de l'opérateur. Cet exécutif a un temps de réaction garanti.
– un *planificateur* (IxTeT) et un *exécutif temporel* (TeXeC). Ces composants sont chargés de la planification de la mission globale de l'agent, gérant les objectifs et les ressources sur un horizon à long terme. Ils doivent être réactifs et prendre en compte les nouveaux buts ainsi que les échecs d'exécution (échec d'une action et time-out).

Le **niveau fonctionnel** comprend toutes les actions et fonctions de perception de base de l'agent. Ces boucles de contrôle et traitements de données sont encapsulés dans des modules contrôlables (développés avec GenoM [Fleury, Herrb, & Chatila 1994]). Chaque module fournit des services et traitements accessibles par des requêtes envoyées par le niveau supérieur ou un autre module. Le module renvoie en retour un bilan lorsqu'il se termine correctement ou est interrompu. Ces modules sont complètement contrôlés par le niveau supérieur et leurs contraintes temporelles dépendent du type de traitement qu'ils ont à gérer (servo-contrôle, algorithmes de localisation, etc.).

Le **niveau de contrôle des requêtes** est situé entre les deux niveaux précédents. Le R^2C *"Requests and Resources Checker"* [Ingrand & Py 2002] vérifie les requêtes envoyées aux modules fonctionnels (par l'exécutif procédural ou entre modules) et l'utilisation des ressources. Il est synchrone avec les modules (il connaît toutes les requêtes envoyées et tous les bilans retournés et construit en ligne l'état du niveau fonctionnel). Il agit comme un filtre qui éventuellement rejette des requêtes en fonction de l'état et d'un modèle formel donné par l'opérateur spécifiant les états autorisés ou interdits. Les bilans retournés par le niveau fonctionnel sont transmis à l'exécutif procédural après mise à jour de l'état interne. Les contraintes temporelles de ce niveau sont de type temps réel dur.

2.2 Contrôle de l'exécution des plans

Le contrôle de l'exécution des plans est réparti entre l'exécutif temporel TeXeC, et l'exécutif procédural OpenPRS. TeXeC décide des dates de lancement et éventuellement d'arrêt des actions (relativement abstraites) du plan global. Plus précisément, à chaque action a sont associés ses paramètres instanciés p_a, un instant de début représenté par le timepoint[3] dt^a, un instant de fin représenté par le timepoint ft^a et un identifieur i_a. On distingue également trois types d'actions : *non interruptible*, *interruptible au plus tôt*, *interruptible au plus tard*. Une action est lancée par envoi de la commande (*START* a p_a i_a) à l'exécutif procédural. Si l'action est *non interruptible*, ft^a est une variable non contrôlable et TeXeC peut juste vérifier si l'action se termine à temps. Dans les autres cas ft^a est contrôlable. Si l'action ne s'est pas terminée d'elle-même, elle est arrêtée par la commande (*END* i_a) envoyée à OpenPRS le plus tôt possible si l'action est *interruptible au plus tôt*, le plus tard possible si l'action est *interruptible au plus tard*.

De la flexibilité est laissée à l'exécutif procédural pour mener à bien l'exécution des actions fournies par TeXeC. On remarque dans la description du niveau décisionnel que l'exécutif procédural est le composant qui communique avec le niveau inférieur (envoi de requêtes, réception de bilans) et avec l'opérateur (au niveau buts et missions). Ce composant est effectivement conçu pour réagir aux événements et buts et agir en conséquence (voir [Ingrand *et al.* 1996] pour plus de détails sur ses fonctionnalités). Il peut cependant avoir besoin d'un nouveau plan pour accomplir un nouveau but ou récupérer d'un échec quand aucune procédure prédéfinie n'est disponible. Ce plan est fourni (action par action) par TeXeC.

Lors de la réception d'une demande de lancement d'action par TeXeC, OpenPRS l'affine (si nécessaire), et/ou choisit la procédure adéquate pour la réaliser en fonction du contexte courant. Cela entraîne généralement l'envoi de requêtes aux modules fonctionnels. Des échecs peuvent se produire au niveau fonctionnel. OpenPRS dispose d'une certaine flexibilité pour pallier à ces échecs avant qu'ils ne soient transmis, en dernier recours, à TeXeC.

Tout d'abord, dans OpenPRS, tout sous-but est automatiquement repris en compte tant qu'on n'a pas atteint un échec "complet" (quand toutes les procédures/unificateurs applicables ont échoué). Par exemple, l'action *Prendre une image* peut échouer en utilisant une première caméra (défaillante) mais réussir en utilisant une caméra de secours. Ce choix peut être fait par OpenPRS, si cette flexibilité lui a été laissée.

Ensuite, les procédures sont généralement écrites de façon à tester en ligne la meilleure stratégie d'exécution à suivre en fonction du contexte.

Ces recouvrements locaux d'échecs ne sont pas le fruit de processus de planification, mais correspondent plus à de bonnes pratiques de programmation et participent à la robustesse globale de l'approche.

OpenPRS envoie un bilan à TeXeC à chaque fin d'action. Ce bilan comprend un statut de fin (*nominal*, *interruption* ou *échec*). Si le bilan n'est pas nominal, il fournit en plus une description partielle de l'état du système (les valeurs finales

[3] Un *timepoint* est une variable temporelle. Une durée entre deux timepoints t_1 et t_2 est donnée par la contrainte $(t_2 - t_1) \in [min, max]$. L'intervalle d'exécution d'un timepoint t est donné par la contrainte $(t - origine) \in [t_{min}, t_{max}]$.

des variables d'état présentes dans la définition de l'action). OpenPRS communique également à TEXEC les nouveaux buts si il y a lieu.

TEXEC intègre ces messages dans le plan. Certaines situations empêchent la poursuite de l'exécution du plan sans l'adapter :

– l'arrivée d'un nouveau but,
– l'échec d'une action,
– les échecs temporels – Le réseau temporel contraint l'exécution de chaque timepoint t dans l'intervalle temporel $[t_{min}, t_{max}]$. Ainsi deux échecs peuvent rendre le plan inconsistant : l'événement correspondant (typiquement, le bilan de fin d'une action) arrive trop tôt ou trop tard (time-out).

Lorsque le plan global n'est plus valide, TEXEC interagit avec le planificateur pour l'adapter. Pour tirer profit de la flexibilité temporelle du plan, la stratégie de replanification comporte deux étapes. Un premier essai consiste à réparer le plan tout en exécutant sa partie valide. Si cette réparation échoue ou si un timepoint est exécuté trop tard, l'exécution du plan est arrêtée (si besoin par lancement d'une procédure de mise en standby au niveau de OpenPRS). Le plan courant est abandonné et une replanification complète à partir du nouvel état et des buts non exécutés est lancée.

3 IXTET-EXEC

Après un bref rappel des propriétés et du fonctionnement du planificateur sous-jacent, cette section développe l'approche choisie pour combiner exécution et replanification.

3.1 Le planificateur IXTET

IXTET est un planificateur temporel basé sur des CSPs qui produit des plans partiellement ordonnés et partiellement instanciés. Sa représentation temporelle décrit le monde comme un ensemble d'*attributs* : des attributs logiques (par ex. ROBOT_POS($?r$) représentant la position du robot $?r$) qui sont des fonctions du temps multivaluées (constantes par morceaux), et des attributs de ressource (par ex. BATTERY_LEVEL()) sur lesquels peuvent être spécifiés des emprunts, consommations ou productions. On note respectivement LgA et $RscA$ les ensembles d'attributs logiques et de ressource. LgA_i et $RscA_i$ désignent les ensembles de toutes les instanciations possibles des ces attributs.

L'évolution de la valeur des attributs logiques est spécifiée par le prédicat *hold*, qui impose la persistance d'une valeur sur un intervalle temporel, et le prédicat *event* qui représente un changement de valeur instantané. Les prédicats *use*, *consume* et *produce* spécifient, respectivement, l'emprunt sur un intervalle, la consommation ou la production à un instant donné d'une certaine quantité de ressource.

Une action (ou *tâche* - voir l'exemple Figure 3) comporte : un ensemble d'*events* décrivant les changements du monde induits par l'action, un ensemble d'assertions *hold* exprimant les conditions requises ou la protection d'un fait entre deux événements, un ensemble de propositions d'utilisation de ressources, un ensemble de contraintes temporelles entre les timepoints de l'action et un ensemble de contraintes entre les variables atemporelles.

Un plan repose sur deux CSPs (temporel et atemporel). Un algorithme de propagation de type Floyd-Warshall assure la consistance globale du réseau temporel. Un autre CSP gère des variables atemporelles symboliques avec des contraintes de type restriction de domaine, égalité ou différence. Une extension récente a amélioré l'expressivité du planificateur ([Trinquart & Ghallab 2001]) en permettant la gestion de variables atemporelles numériques de domaine infini et des contraintes numériques complexes. De plus, il est possible d'exprimer des contraintes mixtes entre variables temporelles et atemporelles, par exemple : $?distance = ?vitesse * (t_2 - t_1)$. Elles sont gérées par un superviseur qui transfère les informations d'un CSP à l'autre. Ces CSPs participent à l'élaboration de plans flexibles : ils calculent pour chaque variable un domaine de valeur minimal qui ne reflète que les contraintes nécessaires du plan.

La recherche explore un arbre dans l'espace des plans partiels. Un plan partiel est défini par un 4-tuple (A, C, L, D). A représente l'ensemble des actions partiellement instanciées, C est l'ensemble des contraintes entre les variables temporelles et atemporelles apparaissant dans les actions de A, L est un ensemble de liens causaux[4] et D est un ensemble de défauts. Un plan partiel représente une famille de plans. Un plan partiel est une solution valide si tous ces plans sont cohérents. Ce qui équivaut à vérifier que D est vide.

Le noeud racine comprend : l'état initial (les valeurs initiales des attributs instanciés), les événements prévus pour certains attributs contingents (non contrôlés par le système), les profils de disponibilité des ressources prévus, les buts à accomplir (valeurs souhaitées pour des attributs instanciés spécifiques) et un ensemble de contraintes temporelles entre ces éléments. Les branches de l'arbre correspondent aux résolvantes (nouvelles actions ou contraintes) insérées dans le plan partiel courant pour résoudre un de ses défauts. On distingue trois types de défauts :

– *les sous-buts* sont des événements ou assertions pas encore établis (la valeur de l'attribut n'étant pas justifiée par l'état initial ou par un événement antérieur). La résolvante correspondante consiste à trouver un événement établisseur (dans le plan ou grâce à l'insertion d'une nouvelle action) et à ajouter un lien causal (une assertion *hold*) qui protège la valeur de l'attribut entre l'*event* établisseur et le sous-but.

– *les menaces* correspondent à des paires de *event* et *hold* dont les valeurs sont potentiellement en conflit (un attribut instancié ne peut avoir qu'une seule valeur à un instant donné). Ce conflit est résolu par l'insertion de contraintes temporelles et de différenciation des paramètres.

[4] Un lien causal $a_i \xrightarrow{p} a_j$ implique que la proposition p de l'action a_j est établie par un effet de l'action a_i. La contrainte de précédence $a_i \prec a_j$ et les contraintes d'unification entre les variables de a_i et a_j apparaissant dans p sont dans C.

– *les conflits de ressources* sont détectés comme des cliques sur-consommatrices dans un graphe particulier. Les résolvantes peuvent être des contraintes de précédence, l'insertion d'une action productrice, etc. [Laborie & Ghallab 1995; Trinquart, Lemai, & Cambon 2002].
La recherche effectue d'abord une analyse des défauts du plan partiel courant ; s'il n'y a pas de défaut, un plan solution a été trouvé ; sinon, un défaut est sélectionné, et une résolvante est choisie dans la liste associée des résolvantes possibles ; elle est insérée dans le plan partiel qui devient le nouveau plan courant, etc. L'algorithme de recherche est complet. Par ailleurs, les choix des défauts et résolvantes sont guidés par diverses heuristiques non discutées ici (cf. [Garcia & Laborie 1995; Ghallab & Laruelle 1994; Trinquart 2003]).
Les avantages de la planification basée sur une représentation par fonctions et sur des CSPs sont nombreux dans un contexte d'exécution des plans. Outre l'expressivité de la représentation (gestion du temps, des ressources), la flexibilité des plans produits (partiellement ordonnés et instanciés, avec des contraintes minimales) est bien adaptée à leur exécution dans un environnement dynamique et incertain. Les plans seront contraints un peu plus au fur et à mesure de leur exécution. Enfin, nous utilisons le fait que ce planificateur effectue une recherche dans l'espace des plans partiels pour l'adapter dans un contexte de planification incrémentale et de réparation de plan.

3.2 Coupler exécution et replanification

Exécuter et planifier simultanément peut introduire des défauts dans le plan. Nous définissons formellement sous quelles conditions un tel plan reste exécutable.

Définitions. Nous étendons la définition précédente d'un plan partiel à celle de P_t : **plan partiel partiellement exécuté jusqu'à l'instant** t.

Definition 1 $P_t = (AC_t, AF_t, E_t, B_t, C_t, L_t, D_t)$.

AC_t est l'ensemble des actions en cours d'exécution ($a \in AC_t$ si $dt^a_{max} < t$ et $ft^a_{max} > t$), AF_t est l'ensemble des actions futures ($a \in AF_t$ si $dt^a_{max} \geq t$). E_t représente l'état du monde à l'instant t et contient la dernière valeur pour chaque attribut $la \in LgA_i$.[5] B_t est l'ensemble des buts non encore réalisés à l'instant t (et éventuellement non établis).[6] C_t est l'ensemble des contraintes sur les variables apparaissant dans AC_t, AF_t, E_t et B_t. L_t est l'ensemble des liens causaux protégeant l'établissement des actions futures. D_t est l'ensemble des défauts présents dans le plan partiel à l'instant t.
Les actions passées ne font plus partie du plan. En effet, leurs timepoints et paramètres sont complètement instanciés et donc n'apparaissent plus dans C_t. De plus les informations utiles sur les effets de ces actions sont conservées dans E_t.
Les timepoints pris en compte par l'exécutif dans le réseau temporel sont les timepoints de but, les timepoints de début d'action ou les timepoints de fin d'action.

Definition 2 (timepoint exécutable) *Un timepoint T est* exécutable *à l'instant t si tous les timepoints T^p qui le précèdent directement dans le réseau temporel ont déjà été exécutés ($T^p_{min} = T^p_{max} < t$) et si $t \in [T_{min}, T_{max}]$.*

Un but est réalisé instantanément ou est persistant (réaliser et maintenir une propriété entre dt^b et ft^b).

Definition 3 (but réalisable) *Un but b est* réalisable *à l'instant t si dt^b est exécutable et si $b \notin D_t$.*

On note A^d_t l'ensemble des actions qui sont concernées par les défauts dans D_t.[7]

Definition 4 (action exécutable) *Une action future a est* exécutable *à l'instant t si dt^a est exécutable et si $a \notin A^d_t$.*

Finalement, un plan partiel est exécutable si les actions en cours d'exécution sont complètement supportées.

Definition 5 (plan exécutable) *Un plan partiel P_t est* exécutable *à l'instant t si les réseaux de contraintes sont consistants et si $AC_t \cap A^d_t = \emptyset$.*

Cycle d'exécution. Le plan solution produit par le planificateur est exécuté par TEXEC selon un cycle "Perception/Réparation de plan/Action". L'exécutif se réveille quand il est temps d'exécuter un timepoint, quand un message a été reçu ou quand une réparation de plan est en cours. On note $ExecPlan$ le plan en cours d'exécution. La partie **Perception** du cycle intègre les messages dans $ExecPlan$, aboutissant éventuellement à un plan partiellement invalidé. Si $ExecPlan$ contient des défauts, une **Réparation de Plan** est requise. On conserve la structure du plan (l'ordre des actions) et on profite de la flexibilité temporelle pour planifier et tenter de trouver un plan solution. Cette planification est distribuée sur plusieurs cycles pour rester réactif aux événements et exécuter simultanément les parties valides du plan. La partie **Action** détermine les timepoints à exécuter dans le cycle, lance les commandes correspondantes ou détecte un time-out.
On note *DurCycle* une estimation de la durée maximale du cycle. Ce paramètre est défini par l'utilisateur en fonction de l'application. L'incertitude sur la durée d'un cycle n'est pas sans conséquence sur l'instant d'exécution exact des débuts et fins d'actions. En effet, deux timepoints qui doivent être exécutés dans un intervalle inférieur à *DurCycle* seront exécutés pendant le même cycle, en respectant toutefois leurs contraintes de précédence. Cependant,

[5] Avec IXTET, cet ensemble contient le dernier *event* exécuté pour chaque la. Nous ne considérons pas pour l'instant les attributs de ressource.

[6] Dans IXTET, un but est représenté par une proposition instanciée *hold(AttBut(b) :ValBut,(dt^b,ft^b))*. B_t contient les buts tels que $ft^b_{max} \geq t$.

[7] La détermination de A^d_t est directe dans le cas de sous-buts ou de conflits de ressource. Dans le cas d'une menace, une action a_k a des effets en contradiction avec l'établissement de la proposition p par le lien causal $a_i \xrightarrow{p} a_j$ et la contrainte $(a_i \prec a_k \prec a_j)$ est vérifiée. A^d_t contient a_k et a_j.

le cycle prend souvent moins de temps et l'exécutif peut réagir plus rapidement aux messages.
Distribuer la planification sur plusieurs cycles soulève deux problèmes importants :

1. **Sur quel plan est basée l'exécution dans la partie *Action* du cycle, en particulier si une solution n'a pas encore été trouvée ?** Ce plan doit être *exécutable* (les actions en cours d'exécution sont complètement supportées dans $ExecPlan$). A chaque étape de planification, si le plan partiel courant est *exécutable*, le noeud correspondant de l'arbre de recherche est marqué. Lorsque le temps maximum alloué à la réparation de plan dans le cycle est atteint et si le plan courant n'est pas acceptable, le dernier noeud marqué est récupéré et le plan associé devient $ExecPlan$.
2. **Sur quel plan et quel arbre de recherche est basée la planification au prochain cycle ?** Si aucune décision n'a été prise entre temps (pas d'exécution de timepoint, pas de message reçu), l'arbre de recherche peut être conservé tel quel et son développement poursuivi au cycle suivant. Il est même possible de revenir sur les décisions prises aux cycles précédents. Cependant, si $ExecPlan$ a été modifié, le développement d'un nouvel arbre est nécessaire, dont le noeud racine est le plan modifié. Les décisions prises précédemment sont alors fixées.

Nous détaillons maintenant les différentes phases du cycle. En fait, toutes les modifications apportées à $ExecPlan$ doivent garantir qu'un plan *exécutable* est disponible à la fin de chaque phase. Si cette condition n'est pas vérifiée, le cycle est interrompu et une replanification complète est nécessaire. Pendant un cycle sans réparation de plan, $ExecPlan$ est un plan solution.

Perception. Un message peut être soit un bilan sur l'achèvement d'une action, soit un nouveau but.
Bilan - Un bilan est associé au timepoint de fin ft^a de l'action correspondante a. Si le bilan est reçu dans l'intervalle $[ft^a_{min}, ft^a_{max}]$, le timepoint est instancié[8] à l'instant courant t. Si le bilan est reçu trop tard, le plan n'est plus exécutable. Si le bilan est reçu trop tôt (souvent le cas lorsqu'une action échoue), un nouveau timepoint de fin de l'action, instancié à t, est créé et les timepoints échoués de l'action sont relâchés (les contraintes les contenant sont supprimées). Des contraintes de précédence entre le nouveau timepoint et tous les timepoints exécutables sont ajoutées. Le réseau temporel est alors repropagé. Dans le cadre d'IxTeT, cette opération ne rend pas le réseau inconsistant car la seule contrainte qui peut être spécifiée entre deux actions a et a' est une contrainte de précédence dont la borne maximale est flexible : $(dt^{a'} - ft^a) \in]0, +\infty[$.
Par ailleurs, les informations sur l'état du système sont intégrées de la façon suivante. E_t contient la dernière valeur pour chaque attribut logique instancié. Si le bilan est nominal, E_t est mis à jour avec les effets de a prévus dans le plan. Sinon, il est mis à jour avec les valeurs retournées par le bilan. Une telle valeur n'est pas insérée si elle conduit à un plan non exécutable (menace une proposition d'une action en cours d'exécution a_c). Dans ce cas, et si a_c est interruptible, son arrêt est requis. Si la valeur est compatible avec les actions en cours d'exécution, elle est insérée et les liens causaux en conflit avec elle sont supprimés.
Nouveau but - Avec chaque but sont associés : une priorité, une durée (intervalle I^d), une estimation de la durée minimale nécessaire pour réaliser le but (d_r) et un intervalle I^b d'exécution de son timepoint de début dt^b. Un but est rejeté si les contraintes suivantes ne sont pas consistantes avec $ExecPlan$ à l'instant t : $((ft^b - dt^b) \in I^d)$, $((dt^b - origine) \in I^b)$, $(dt^b - origine \geq t + d_r)$. De plus, pour chaque action en cours d'exécution a_c dont une proposition est menacée par la proposition but, la contrainte de précédence $(dt^b - ft^{a_c}) \in]0, +\infty[$ est imposée.
Après la réception des messages, le plan est exécutable et peut contenir des sous-buts à (r)établir par une réparation de plan. Cette réparation peut nécessiter l'insertion de nouvelles actions. Il faut pour cela invalider une partie du plan en cassant des liens causaux supplémentaires.
Suppression de liens causaux - Nous avons adapté les travaux présentés dans [Gaborit 1996] pour déterminer les liens causaux à supprimer. A partir de la liste des sous-buts et du graphe d'abstraction des actions [9], on recherche les attributs instanciés présents dans les sous-buts et dans les actions (partiellement instanciées) potentiellement insérées. On extrait ensuite l'ensemble des liens causaux portant sur ces attributs instanciés et ayant lieu dans le futur, mais avant les sous-buts. Enfin, parmi ces liens causaux, on retire du plan ceux qui n'appartiennent pas à l'*extension* de leur événement établisseur. L'extension d'un $event$ correspond à l'intervalle pendant lequel la valeur établie est imposée (par l'assertion d'une action...).
Le plan n'est finalement que partiellement invalidé. Aucun lien causal portant sur les actions en cours d'exécution n'a été supprimé. Certaines actions, indépendantes de l'action échouée ou de la réalisation du but, restent valides. La réparation est donc lancée, tout en poursuivant l'exécution de la partie valide.

Réparation de plan. La réparation de plan est similaire à la recherche d'un plan par IxTeT dans l'espace des plans partiels. Le noeud racine est le plan partiellement invalidé $ExecPlan$. L'arbre de recherche est développé selon une stratégie de recherche en profondeur ordonnée. Si nécessaire, la planification est distribuée sur plusieurs cycles et à chaque fois qu'un nouveau timepoint est inséré dans le plan, il est contraint de se produire après la fin du cycle courant. Pendant un cycle, la planification est faite étape par étape jusqu'à l'obtention d'une solution, ou la détermination qu'il n'existe pas de solution, ou le dépassement d'un temps limite. Ce temps limite est un paramètre (μ) défini par l'utilisateur et correspond à la portion de $DurCycle$

[8] La contrainte $(ft^a - origine) = t$ est postée dans le STN.

[9] Calculé hors-ligne, ce graphe explicite notamment les attributs principaux justifiant l'insertion de chaque action.

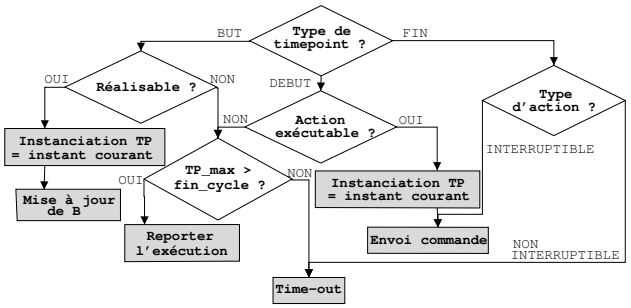


FIG. 2 – Exécution d'un timepoint.

allouée aux parties Perception et Réparation de Plan du cycle.

Des mécanismes d'agrégation (correspondant à un "oubli du passé") permettent de réduire significativement l'espace de recherche. Plus particulièrement, dans IXTET, les événements établisseurs sont recherchés dans E_t.

Le processus de réparation de plan ne garantit pas l'obtention d'un plan solution (décisions gelées par l'exécution, contraintes temporelles trop justes pour ajouter de nouvelles actions...), mais peut éviter l'arrêt de l'exécution et une replanification complète à chaque échec. La réparation de plan est particulièrement efficace et utile pour les plans avec une certaine flexibilité temporelle et un certain parallélisme (des ensembles d'actions peuvent être exécutés indépendamment).

Action. A chaque timepoint est associé un *instant d'exécution* t_{exec}. Si T est un timepoint de début d'action ou de but, ou un timepoint de fin d'une action interruptible au plus tôt, $t_{exec} = T_{min}$. Si T est un timepoint de fin d'une action interruptible au plus tard, $t_{exec} = T_{max} - DurCycle$. Si T est un timepoint de fin d'une action non interruptible, $t_{exec} = T_{max}$. Pendant la phase Action du cycle, TEXEC détermine l'ensemble $ExecTPs$ des timepoints à exécuter pendant le cycle courant : ces timepoints sont exécutables et leur instant d'exécution a lieu avant la fin du cycle. $ExecTPs$ est mis à jour après chaque exécution de timepoint pour prendre en compte les nouveaux timepoints exécutables. La figure 2 résume en quoi consiste l'exécution d'un timepoint, selon son type. Si un but n'est pas réalisable ou si une action n'est pas exécutable et si il reste une certaine flexibilité, leur exécution est reportée. Un timepoint de fin d'action interruptible est instancié quand le bilan correspondant est reçu (attendu pour le prochain cycle). Finalement, un time-out est détecté quand les bilans n'ont pas été reçus à temps.

Pendant l'exécution du plan, instancier un timepoint est équivalent à ajouter une contrainte numérique entre l'origine et le timepoint. Une nouvelle contrainte entre deux timepoints est propagée à tous les "triangles" contenant au moins un des deux timepoints. De cette façon, le réseau reste minimal et garantit qu'une exécution complète est possible.

Replanification complète. Lorsque un plan exécutable n'est plus disponible, une replanification complète est faite. On attend l'arrêt de toutes les actions en cours. On obtient alors le plan $P_{t_s} = (\emptyset, AF_{t_s}, E_{t_s}, B_{t_s}, C_{t_s}, L_{t_s}, D_{t_s})$. Le plan initial P_{t_i} pour la replanification est extrait de P_{t_s} :

$$
\begin{aligned}
P_{t_i} &= (\emptyset, \emptyset, E_{t_i}, B_{t_i}, C_{t_i}, \emptyset, D_{t_i}) \\
E_{t_i} &= E_{t_f}, \\
B_{t_i} &= \{b \in B_{t_s} / \text{ les contraintes temporelles de } b \\
&\quad \text{sont cohérentes avec l'instant courant}\}, \\
C_{t_i} &= \{c \in C_{t_s} / c \text{ est une contrainte sur} \\
&\quad \text{les variables apparaissant dans } E_{t_i} \text{ et } B_{t_i}\} \\
&\quad (C_{t_i} \text{ contient notamment les contraintes} \\
&\quad \text{sur les timepoints origine et horizon}), \\
D_{t_i} &\quad \text{contient les buts de } B_{t_i}.
\end{aligned}
$$

Un timepoint spécifique T^{finP} est ajouté. Il représente la fin du processus de planification. Les bornes initiales de ce timepoint sont flexibles (seules contraintes : avoir lieu après t_i et avant la fin de l'horizon). Chaque nouveau timepoint inséré par le processus de planification est contraint d'avoir lieu après T^{finP}. Ainsi, T^{finP}_{max} décroît lorsque de nouvelles actions ou de nouvelles contraintes temporelles sont ajoutées, et il ne reste pas assez de temps pour exécuter le nouveau plan si ($T^{finP}_{max} <$ instant courant). T^{finP}_{max} peut toutefois croître lors de retour en arrière dans l'arbre de recherche.

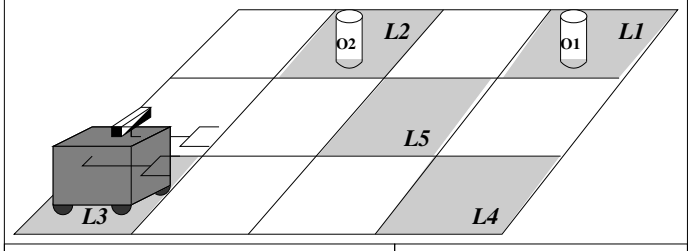
La stratégie consiste alors à planifier étape par étape jusqu'à l'obtention d'une solution, ou la détermination qu'il n'existe pas de solution, ou le dépassement d'un temps limite l avec $l = T^{finP}_{max} - d$. d est un paramètre réglé par l'utilisateur, qui correspond au temps nécessaire à la fin de la planification pour initialiser le cycle d'exécution. l est mis à jour après chaque étape de planification. La planification est interrompue quand l est atteint, sauf si la prochaine étape correspond à un noeud de backtrack. Dans ce cas, et si la prochaine étape augmente l, la planification est poursuivie.

Si la planification est arrêtée sans avoir trouvé de solution, certains buts sont rejetés et une nouvelle tentative est lancée. Un nouveau plan P_{t_i} est extrait de P_{t_s}. Cette opération peut rejeter certains buts, si leurs contraintes temporelles ne sont pas consistantes avec l'instant courant. Sinon, un but est sélectionné et abandonné. Le but rejeté est celui qui a la plus faible priorité, et si plusieurs buts ont la même priorité, celui qui a le moins de flexibilité pour sa réalisation[10]. Ce critère a été choisi pour conserver les buts qui ont le plus de chance d'être réalisés en temps voulu.

4 Exemple et tests sur une plate-forme robotique

Cette section illustre dans un premier temps la réparation et l'exécution simultanées d'un plan à travers un exemple

[10] Cette flexibilité est mesurée par $(dt^b_{max} - d_r)$.



Initial State
event(ON(O1): (?,L1),tS)
event(AT_OBJECT(O1): (?,L1),tS)
event(ON(O2): (?,L2),tS)
event(AT_OBJECT(O2): (?,L2),tS)
event(AT_ROBOT(): (?,L3),tS)
event(AVAILABLE(LH):(?,T),tS)
event(AVAILABLE(RH):(?,T),tS)

Goals
hold(ON(O1): L4,(tG,tE))
hold(ON(O2): L4,(tG,tE))

```
task CARRY(?l1,?l2,?obj,?arm) (tstart,tend){
    ?l1,?l2 in LOCATIONS
    ?obj in OBJECTS
    ?arm in {LH,RH}
    timepoint t1

    ?l1 != ?l2
    (tend−t1) in [1,1]
    (tend − tstart) in [1,1000]

    hold(AT_ROBOT(): ?l2,(t1,tend))              (1)
    hold(ON(?obj): on_robot,(tstart,tend))       (2)
    event(AT_OBJECT(?obj): (?l1,at_object_idle),tstart)
    hold(AT_OBJECT(?obj): at_object_idle,(tstart,tend))
    event(AT_OBJECT(?obj): (at_object_idle,?l2),tend)
    hold(AVAILABLE(?arm):F,(tstart,tend))
} early preemptive
```

FIG. 3 – Exemple de formalisme IxTeT.

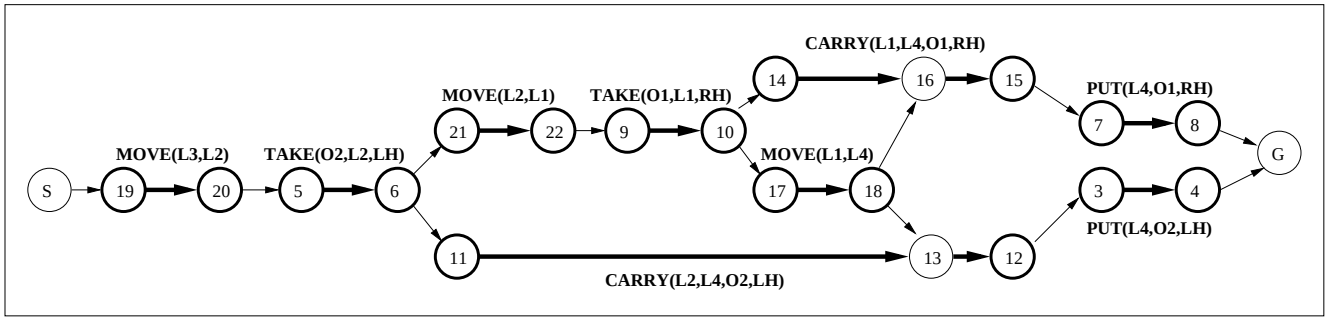


FIG. 4 – Plan initial.

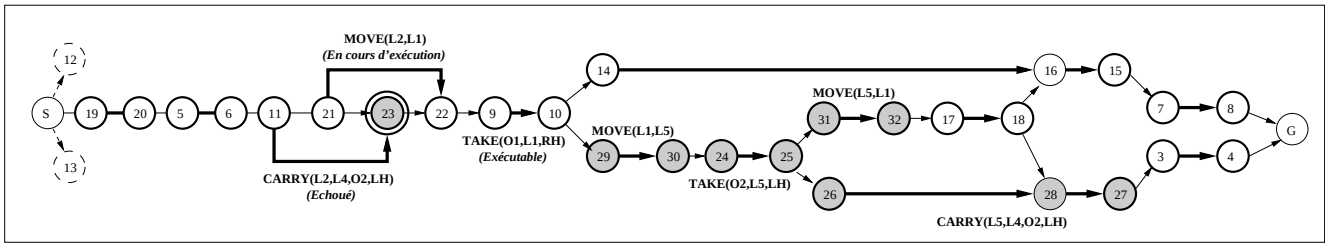


FIG. 5 – Plan après réparation.

testé en simulation, et présente dans un deuxième temps les premiers tests menés sur une plate-forme robotique.

4.1 Exemple de réparation et exécution simultanées de plans

On considère un robot avec deux bras (LH et RH), dans la position initiale $L3$. Il doit prendre deux objets ($O1$ et $O2$, respectivement situés en $L1$ et $L2$), et les amener en $L4$. Quatre actions décrivent les fonctionnalités du robot : MOVE (aller d'une position à une autre), TAKE (prendre un objet avec un de ses bras), CARRY (porter l'objet d'une position à une autre) et PUT (poser l'objet). La Figure 3 illustre la description de l'état initial, des buts et d'une action dans le formalisme IxTeT. L'action CARRY est interruptible au plus tôt. Elle sera terminée dès que le robot arrive dans la position finale $?l2$ avec l'objet $?obj$. La proposition (1) garantit que le robot arrive en $?l2$ 1s avant la fin possible de l'action et la proposition (2) impose que l'objet soit porté par le robot pendant toute la durée de l'action.

La Figure 4 représente le plan initial trouvé par IxTeT (les cercles foncés correspondent aux timepoints de début et fin d'actions, les flèches représentent les contraintes de précédence entre les timepoints). En cours d'exécution, un échec se produit : en allant vers $L1$, le robot laisse tomber l'objet $O2$ sur le sol à la position $L5$. La Figure 5 représente le plan réparé par IxTeT-EXEC. L'échec est survenu au début de l'action CARRY, un timepoint d'échec (23) a été créé et les timepoints 12 et 13 ont été relâchés. La partie du plan concernée par la suppression de liens causaux est liée seulement aux attributs représentant la position de $O2$ (et,

pour l'insertion de nouvelles actions, à l'attribut représentant la position du robot). L'action TAKE(O1,L1,RH) reste valide et peut-être lancée dès que le robot arrive en $L1$. Les timepoints hachurés correspondent aux actions ajoutées par la réparation. On remarque que ce plan n'est pas optimal puisque toutes les actions initiales sont conservées. Deux paramètres sont réglés par l'utilisateur : *DurCycle* et sa portion allouée à la réparation μ. La durée du cycle dépend de plusieurs facteurs, dont : le nombre de timepoints exécutés pendant le cycle, la durée de propagation des contraintes (dans le STN, la complexité est en (n^2+n), n étant le nombre de timepoints du réseau), la durée de l'invalidation du plan, la durée de reconstruction du plan exécutable, la durée maximale d'une étape de planification. L'exemple simple présenté ci-dessus a été testé sur une SunBlade100. L'invalidation du plan (intégration du message d'échec) prend 190ms. La réparation est faite en 29 étapes avec 1 backtrack, distribuées sur 2 cycles pour *DurCycle* = 1s (μ= 85%), sur 4 cycles pour *DurCycle* = 600ms (μ= 80%). Pendant les autres cycles, une fréquence de contrôle de 5Hz est atteinte.

4.2 Tests sur une plate-forme robotique

FIG. 6 – L'ATRV Dala.

IXTET-EXEC a été intégré dans l'architecture LAAS et embarqué sur le robot d'extérieur Dala (un ATRV d'iRobot – cf. Figure 6). Les premiers tests sont basés sur un scénario de mission d'exploration. Les buts sont une liste de cibles que le robot doit atteindre et prendre en photo. L'ATRV embarque un PentiumIII sous Linux et utilise les capteurs suivants : odométrie des roues et une paire de caméras stéréo montées sur une platine. Trois actions sont définies au niveau planification de mission : take-picture, move-platine et move. La séquence d'actions pour réaliser un but est : aller à la cible (les caméras sont orientées vers l'avant), modifier l'orientation des caméras, prendre l'image. L'exécution d'une action move dans un environnement inconnu met en oeuvre plusieurs traitements complexes : localisation, construction de cartes, génération de mouvement, etc. [Lacroix *et al.* 2003]. Ces traitements sont réalisés dans les différents modules du niveau fonctionnel (cf. Figure 1). Ainsi le module POM calcule la meilleure estimation de position à partir des données de l'odométrie visuelle (STEO) et de celles de l'odométrie des roues (RFLEX). Le module RFLEX contrôle également la vitesse des roues en fonction de la consigne produite par le planificateur de mouvement réactif P3D.

Sous le contrôle d'IXTET-EXEC (avec DurCycle = 1s), le robot a pris 4 images aux endroits requis (1 des buts étant reçu en cours de route). IXTET-EXEC a réagi aux échecs suivants : une tâche de navigation a pris plus de temps que prévu, échec de commande de la platine, échec de construction de la carte. Ces deux derniers échecs nécessitent la ré-initialisation de certains modules. Ces reprises d'échec sont prises en compte par TEXEC car les initialisations prennent du temps. Deux actions (platine-init et mvt-init) et deux attributs spécifiques (platine-init-done() et mvt-init-done()) ont été ajoutés au modèle. Ainsi l'action move impose la condition hold(mvt-init-done() :true,(dt,ft)). Pour un certain type d'erreur retourné par le module P3D (ici : "pas de carte disponible"), OpenPRS a renvoyé à TEXEC un bilan d'échec de l'action move avec la valeur false pour l'attribut mvt-init-done() et la position courante du robot pour l'attribut robot-pos(). La réparation de plan a inséré en un cycle l'action de ré-initialisation et une nouvelle action move de la position courante à la destination.

On remarque que les plans produits dans le contexte de cette application sont séquentiels. Un échec entraîne donc l'interruption de l'exécution du plan. L'invalidation partielle et la réparation du plan permet cependant de trouver un plan solution plus rapidement que par une replanification complète et TEXEC reste réceptif à l'arrivée d'un nouveau but. Par ailleurs, les temps de réaction obtenus sont satisfaisants.

5 Conclusion et perspectives

Ce papier présente nos premiers résultats sur IXTET-EXEC qui permet le couplage entre le planificateur temporel et l'exécutif procédural intégrés dans l'architecture LAAS, et autorise la replanification dynamique en cas d'échec ou de time-out ainsi que la prise en compte de nouveaux buts en ligne. L'approche adoptée de réparation de plan permet, dans une certaine mesure, de poursuivre l'exécution pendant la planification. Elle est bien adaptée pour les applications avec des sous-systèmes relativement indépendants dont des actions peuvent être exécutées en parallèle. Dans les situations critiques, où le système n'a pas le temps de réparer, l'utilisation de procédures d'urgence permet de mettre le système dans un état sûr (pour amorcer un processus plus long de replanification complète).

L'implémentation d'IXTET-EXEC se poursuit. Nous désirons ajouter certaines fonctionnalités, parmi lesquelles :

– Un des intérêts du planificateur IXTET est sa capacité à gérer l'utilisation de ressources. Les quantités peuvent être définies comme des variables sur des domaines continus. Nous voulons exploiter cette flexibilité en mettant à jour régulièrement pendant l'exécution le niveau des ressources et détecter ainsi de futurs conflits potentiels.

Le système devra également réagir aux modifications soudaines de capacité. La réparation (insertion d'actions productrices...) et la replanification devraient se faire selon le même processus.

– IXTET n'a pas de gestion spécifique des variables temporelles non contrôlables et les durées sur ces variables peuvent être réduites abusivement lors de la propagation. IXTET-EXEC va seulement détecter quand le temps imparti à un timepoint non contrôlable est dépassé. Cependant, cet aspect peut être amélioré en adaptant les travaux récents sur les STN avec incertitude (STNU[11]) et en modifiant le gestionnaire du réseau pour qu'il vérifie la Propriété de Contrôlabilité Dynamique décrite dans [Morris, Muscettola, & Vidal 2001].

Les premiers tests menés sur une plate-forme robotique sont encourageants. Nous allons maintenant tester les performances de notre système dans le cadre d'applications de navigation et exploration autonomes plus complexes.

Références

Alami, R. ; Chatila, R. ; Fleury, S. ; Ghallab, M. ; and Ingrand, F. 1998. An architecture for autonomy. *International Journal of Robotics Research, Special Issue on Integrated Architectures for Robot Control and Programming* 17(4) :315–337.

Ambros-Ingerson, J., and Steel, S. 1988. Integrating planning, execution and monitoring. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*.

Beetz, M. 2000. Runtime plan adaptation in structured reactive controllers. In *Proceedings of the Fourth International Conference on Autonomous Agents*.

Chien, S. ; Knight, R. ; Stechert, A. ; Sherwood, R. ; and Rabideau, G. 2000. Using iterative repair to improve the responsiveness of planning and scheduling. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*.

Dechter, R. ; Meiri, I. ; and Pearl, J. 1989. Temporal constraint networks. In *KR'89 : Principles of Knowledge Representation and Reasoning*. Morgan Kaufmann.

Despouys, O., and Ingrand, F. 1999. Propice-plan : Toward a unified framework for planning and execution. In *Proceedings of the European Conference on Planning (ECP)*.

Fleury, S. ; Herrb, M. ; and Chatila, R. 1994. Design of a modular architecture for autonomous robot. In *IEEE International Conference on Robotics and Automation*.

Gaborit, P. 1996. Planification distribuée pour la coopération multi-agents. *Thèse de Doctorat, Université Paul Sabatier, Toulouse*.

Garcia, F., and Laborie, P. 1995. Hierarchisation of the search space in temporal planning. In *Proceedings of the European Workshop on Planning (EWSP)*, 235–249.

Ghallab, M., and Laruelle, H. 1994. Representation and Control in Ixtet, a Temporal Planner. In *Proceedings of the International Conference on AI Planning Systems*, 61–67.

Haigh, K. Z., and Veloso, M. M. 1998. Planning, execution and learning in a robotic agent. In *Proceedings of the International Conference on AI Planning Systems*.

Ingrand, F., and Py, F. 2002. An Execution Control System for Autonmous Robots. In *IEEE International Conference on Robotics and Automation*.

Ingrand, F. ; Chatila, R. ; Alami, R. ; and Robert, F. 1996. PRS : A High Level Supervision and Control Language for Autonomous Mobile Robots. In *IEEE International Conference on Robotics and Automation*.

Knoblock, C. 1995. Planning, executing, sensing and replanning for information gathering. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.

Laborie, P., and Ghallab, M. 1995. Planning with Sharable Resource Constraints. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1643–1649.

Lacroix, S. ; Mallet, A. ; Bonnafous, D. ; Bauzil, G. ; Fleury, S. ; Herrb, M. ; and Chatila, R. 2003. Autonomous rover navigation on unknown terrains, functions and integration. *International Journal of Robotics Research*.

McCarthy, C., and Pollack, M. 2002. A plan-based personalized cognitive orthotic. In *Proceedings of the International Conference on AI Planning Systems*.

Morris, P. H. ; Muscettola, N. ; and Vidal, T. 2001. Dynamic control of plans with temporal uncertainty. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.

Muscettola, N. ; Dorais, G. A. ; Fry, C. ; Levinson, R. ; and Plaunt, C. 2002. Idea : Planning at the core of autonomous reactive agents. In *Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space*.

Simmons, R., and Apfelbaum, D. 1998. A task description language for robot control. In *IRS*.

Trinquart, R., and Ghallab, M. 2001. An extended functional representation in temporal planning : towards continuous change. In *Proceedings of the European Conference on Planning (ECP)*.

Trinquart, R. ; Lemai, S. ; and Cambon, S. 2002. One step on the left, one step on the right and back to the middle : Exploring temporal domains in a pop fashion. In *Proceedings of the AIPS Workshop on Temporal Planning*.

Trinquart, R. 2003. Analyzing reachability within plan space. In *Proceedings of the ICAPS Doctoral Consortium*.

[11] Simple Temporal Network with Uncertainty