HiDDeN, une architecture décisionnelle distribuée pour la coopération de véhicules individuellement autonomes

Thibault Gateau, Charles Lesire, Barbier Magali

HiDDeN, une architecture décisionnelle distribuée pour la coopération de véhicules individuellement autonomes

Thibault Gateau[1] Charles Lesire[1] Magali Barbier[1]

[1] ONERA / DCSD
31055 Toulouse, France
Prenom.Nom@onera.fr

Résumé

HiDDeN est une architecture décisionnelle, distribuée au sein d'une équipe de véhicules autonomes, qui permet leur coopération dans l'accomplissement d'une mission dont l'environnement est dynamique et dans lequel les communications ne sont pas toujours possibles. Gérer l'exécution de la mission et procurer une certaine robustesse face à l'apparition d'aléas font partie des objectifs du système HiDDeN. On admet l'hypothèse que l'on dispose d'un plan de mission hiérarchisé. L'équipe de véhicules, possiblement hétérogènes, est décomposée en une structure hiérarchique de sous-équipes. Lors de la détection d'aléa(s), une réparation de plan est menée de façon aussi locale que possible, en se basant sur les deux structures mentionnées, pour permettre à l'équipe d'atteindre ses objectifs initiaux. La faisabilité d'un tel système a été évaluée en simulation sur un problème de blanchiment de chenal, destiné à une expérimentation réelle prochaine.

Mots Clefs

Multi-robot, autonomie, exécution de plan, réparation de plan, architecture décisionnelle.

Abstract

The HiDDeN layer is a decisional architecture distributed on autonomous vehicles that allows them to cooperate in order to achieve a team mission in a dynamic environment under communication constraints. HiDDeN aims at facilitating execution management and at increasing fault tolerance of the global system. The mission plan is assumed to be based on the HTN framework. The team is also decomposed into hierarchically structured sub-teams. After a failure detection, plan repair is done as locally as possible, leaning upon the two above structures. The feasibility of the system is evaluated thanks to the simulation of a real world mine-hunting scenario.

Keywords

Multirobot, autonomy, plan execution, plan repair, architecture.

1 Introduction

Nombre de véhicules robotisés se sont au cours du temps dotés d'un niveau d'autonomie décisionnelle respectable. Ils sont employés dans bien des situations différentes, et dans la plupart des cas, leur capacité décisionnelle repose sur des systèmes embarqués qui ont déjà démontré leur efficacité. Beaucoup d'architectures décisionnelles ont été proposées [1, 2, 3, 4], et sont actuellement utilisées dans des situations réelles et complexes où des véhicules autonomes savent prendre en charge l'exécution des tâches qui leur ont été assignées. Depuis plusieurs années déjà, les avancées permettent de s'interroger sur la façon dont ces véhicules autonomes peuvent s'intégrer au sein d'une équipe impliquant d'autres véhicules autonomes hétérogènes, dotés de fonctionnalités complémentaires.
Cependant, la collaboration d'un ensemble de véhicules autonomes non initialement conçus pour des missions coopératives est loin d'être aisée. Des modifications de leur architecture d'origine, ou l'utilisation de solutions ad-hoc (souvent peu génériques) sont en général indispensables.
Il serait pourtant intéressant de pouvoir réutiliser toutes les capacités opérationnelles des robots déjà acquises, sans pour autant redéfinir l'ensemble de leurs architectures de contrôle, et au contraire, pouvoir en tirer parti.
Nous nous intéressons donc, au travers de cet article, à la conception d'une *architecture décisionnelle de haut niveau* dont l'objectif est de gérer l'exécution de missions par une équipe de véhicules autonomes, tout en nous appuyant sur leurs architectures individuelles déjà opérationnelles (et ayant fait l'objet de tests poussés). Cette architecture décisionnelle est distribuée. Elle vise également à doter l'équipe d'une certaine robustesse face aux aléas, qui peuvent tout aussi bien concerner un seul robot, une sous-partie de l'équipe ou l'équipe entière. Nous entendons par aléa tout événement inattendu (mais parfois connu), dont la date d'occurrence est inconnue.

1.1 État de l'art

Un certain nombre d'architectures multi-véhicules a été proposé, issues de domaines robotiques, mais aussi de la

communauté des SMA (Systèmes Multi Agents).

Architectures de contrôle multi-véhicules. BErkeley AeRobot (BEAR) [5] cherche à réaliser l'intégration de multiples véhicules autonomes hétérogènes, de capacités également hétérogènes, pour en faire un système intelligent coopératif qui serait modulaire, robuste, adaptatif en environnement dynamique et capable de mener à bien des missions complexes. Vidal et al. [6] indiquent cependant que la mise en oeuvre réelle n'est pas envisageable et propose pour y remédier et en conservant les bases de BEAR une architecture distribuée, hybride et hiérarchique. Ils insistent sur l'autonomie de chaque agent et le fait d'avoir une coordination au sein de l'équipe. Contrairement aux architectures réactives, la dynamique des agents est prise en compte au plus haut niveau du processus de décision. Un effort est également apporté pour réduire la communication au minimum. Malgré l'aspect modulaire, les connexions entre les modules sont nombreuses, ce qui implique, dans la mise en œuvre concrète du système, un processus d'adaptation à chaque type d'agent elle aussi complexe.
L'architecture ALLIANCE [7] est basée sur une approche réactive (*behavior-based*) et permet aux membres de l'équipe de robots "de réagir de façon robuste, correcte, flexible et cohérente à une évolution inattendue de l'environnement". La réactivité des robots est induite par des *motivations* comme l'impatience ou le consentement. Ces motivations sont conçues pour autoriser l'exécution d'une tâche par un membre de l'équipe tant qu'elles démontrent qu'elles ont l'effet escompté sur le monde. Des modules, appelés *Motivational Behavior*, reçoivent des informations et, en fonction de ces dernières, sélectionnent les actions adaptées à la situation. Nous verrons que les bases qu'ils ont posées inspirent notre travail, même si contrairement à cette architecture, nous nous plaçons à un niveau plus délibératif.
Avec les même principes, Chaimowicz et al. [8] proposent une architecture pour des systèmes multi-robots étroitement couplés. L'équipe de robots est organisée suivant une structure de leader-serviteurs dans laquelle l'architecture locale d'un robot est indépendante du contrôle d'exécution. Cependant, la communication doit être à tout instant possible.

Réparation de plan et planification en ligne. Un plan, ou du moins un ensemble d'objectifs, doit être fourni à l'équipe de robots afin d'accomplir une mission. Dans un souci de robustesse face à des aléas, l'existence de processus de réparation de plan est un atout.
Dans le cadre des approches HTN (Hierarchical Task Network) [9], le problème de planification est structuré de façon hiérarchique. L'architecture Retsina [10] gère l'exécution d'un tel plan. Elle utilise un module de planification qui permet d'entrelacer planification et exécution, qui établit une liste dynamique d'objectifs stockés sous la forme d'une queue à priorités. Les agents sont ainsi hautement adaptables à la dynamique de l'environnement. L'architecture est, en partie, basée sur le cadre BDI (Belief-Desire-Intention). Les agents construisent des plans partagés, et des négociations ont lieu entre eux pour se mettre d'accord sur leurs objectifs respectifs. L'absence de communication n'est cependant pas étudiée, et il n'y a pas explicitement de plan global d'équipe ce qui rend difficile les raisonnements au niveau de l'équipe, ou de son organisation.
HOTRiDE (Hierarchical Ordered Task Replanning in Dynamic Environments) [11] permet d'effectuer de l'exécution de plan et de la réparation en ligne, également sur des structures HTN. Un graphe de dépendance entre tâches est construit pour savoir ce qui doit être remis en cause en cas d'aléas, et éviter de recalculer des parties de plan déjà satisfaisantes. Par contre, si des tâches "haut niveau" doivent être traitées, et si une des tâches principales pour la mission échoue, toutes celles qui suivent dans le plan sont *a priori* compromises. Et bien que cette approche soit intéressante, elle reste dans un cadre mono-agent.
La réparation partielle de plan est explicitement étudiée par Bonnet et al. [12], notamment en s'appuyant sur des structures dynamiques de sous-équipes. Ils définissent notamment la notion de *hiérarchie d'agenticité*, et représentent leurs plans comme des réseaux de Petri hiérarchisés. Cependant, le problème des communications n'est pas abordé, et les étapes de planification et de replanification ne sont pas formalisées.
Boella et al. [13] abordent le problème de la replanification dans un cadre BDI. Ils exploitent pour cela des plans *partiels*, i.e des plans qui ne sont pas totalement instanciés. Une planification initiale a lieu, et le plan est exécuté par les agents. Durant l'exécution, lorsque les agents déterminent que l'utilité de leur plan est inférieure à un certain seuil espéré, une replanification a lieu. Même si l'intérêt théorique est certain, l'application à des plans multi-robots en environnement réel semble moins évident.
Des résultats empiriques [14] montrent par ailleurs l'intérêt d'effectuer des réparation locales plutôt qu'une replanification totale du plan.
D'autres auteurs [3] travaillent sur des scénarios qui impliquent des équipes de drones autonomes à voilure tournante. Ils utilisent un planificateur basé sur de la logique temporelle (TAL, pour Temporal Action Logic). De ce plan sont extraites des conditions qui doivent rester valides pendant l'exécution de la mission. Mais l'interaction entre les véhicules semble implicite, ou la communication totale.

Point clé : les communications. Les communications semblent être un problème récurrent et constituent un aspect à ne pas négliger. Elles sont généralement une des composantes clés pour la coordination et le partage d'informations entre véhicules. Nombre de travaux tels que ceux de Dix et al.[15] reconnaissent la problématique de la communication, mais partent bien souvent de l'hypothèse que celle-ci est parfaite (pas de perte de messages), voire totale et permanente (à chaque instant, tous les véhicules peuvent communiquer entre eux). "Plus l'équipe [d'agents]

est flexible et robuste, plus la charge de communication est élevée" [16]. Et inversement, une réduction de cette charge entraîne une perte en robustesse, d'autant plus si l'environnement est dynamique.

Dans le travail de Magnusson et al. [17], les agents perdent leur capacité de replanification locale en cas d'interruption des communications. En revanche, certains auteurs [18, 19] autorisent l'exécution d'un plan dans des situations où la communication n'est pas permanente. Mais même si les plans sont distribués aux agents, il n'y a pas de notion de réparation locale ou globale de plan, ce qui rend ce plan difficilement adaptable à un environnement dynamique incertain.

Parmi les architectures citées, certaines ([16] [10] et [17]) sont basées sur le cadre BDI, ce qui implique des problèmes de cohérence lors de la perte des liens de communication. Pour le moment, il semble difficile de surmonter simplement ce genre de problème avec ce type de structure.

Legras et Tessier [20] définissent une sous-équipe en se basant sur les liens de communication. Nous verrons que cette approche ne semble pas la mieux adaptée dans les missions que nous traitons.

Dans d'autres cas, les actions d'un véhicule distant sont prédites lorsque la communication n'est pas disponible [21]. Lors d'un retour à un état où la communication est possible, le véhicule met à jour ses connaissances.

1.2 Choix et intuitions

L'environnement, tout comme l'équipe de robots, est intrinsèquement dynamique. Comme le souligne Parker [7], le choix d'avoir une équipe de robots et une architecture distribuée semblent une solution naturelle. L'utilisation d'un unique robot monolithique serait une solution trop coûteuse comparée à un système qui met en œuvre une équipe de robots plus simples : la complexité de l'environnement et de la mission requièrent bien souvent trop de fonctionnalités à développer pour un robot unique, même si un robot bien conçu a toute son utilité lorsqu'il s'interface avec les autres membres d'une équipe de robots. De plus, un tel système est par nature distribué en temps, en espace et/ou en terme de fonctionnalités. Il doit tenir compte de parallélisme et de redondance, même si des problèmes de cohérence sont à prévoir, contrairement à des approches centralisées qui nécessitent par contre une communication permanente.

Nous avons choisi le formalisme HTN (Hierarchical Task Network) [9, 22] pour représenter notre plan de mission. Un HTN est un ensemble de tâches abstraites et élémentaires. Une ou plusieurs méthodes sont associées à chacune des tâches abstraites et décrivent la façon d'en atteindre les objectifs, en utilisant d'autres tâches, abstraites ou élémentaires. Le choix d'une méthode est contraint par des préconditions à satisfaire. Théoriquement, si la recette fournie par une méthode est déroulée, les objectifs de la tâche abstraite, associée à la méthode, sont atteints.

L'intérêt des HTN réside dans leur flexibilité, l'aspect hiérarchique de leur structure et l'intuitivité de leur représentation pour un opérateur humain. En outre, des planificateurs multi-agents tel que A-SHOP [15] ont été développés spécifiquement avec ce formalisme. Les HTN sont utilisés de façon classique pour représenter des problèmes de planification. En reprenant les définitions de Nau et al. [9], un *domaine de planification* est un couple $\mathcal{D} = (Op, Me)$ où Op est un ensemble d'opérateurs (qui définissent les tâches élémentaires), et Me un ensemble de méthodes. Un *problème de planification* est un triplet $\mathcal{P} = (d, I, \mathcal{D})$ où $\mathcal{D}$ est un domaine de planification, I l'état initial, et d le réseau de tâches pour lequel on souhaite obtenir un plan. Un planificateur HTN est capable de retourner un plan lorsqu'un problème de planification lui est fourni. Nous avons choisi de conserver dans le plan solution les relations hiérarchiques entre tâches et non pas uniquement la liste de tâches élémentaires solution (ce genre d'information est aisé à conserver lors du processus de planification).

Cela nous amène à définir une structure d'*arbre HTN instancié*, ou HTNi, et solution d'un problème de planification. Un HTNi est un arbre de tâches, ayant pour racine une tâche abstraite et pour feuilles des tâches élémentaires. Les tâches sont reliées entre elles à travers des méthodes, qui construisent à la fois des relations de parenté mère-filles entre les tâches, et des relations d'ordre partiel entre les tâches issues d'une même mère (appartenant à la même méthode).

Le HTNi est un sept-uplet $(E, V, Pr, Po, Te, Ta, M)$:

- E un ensemble d'*Etiquettes* ;
- V un ensemble de *Variables instanciées* ;
- Pr un ensemble de *Préconditions*. Une précondition est une expression booléenne évaluée en fonction des éléments de V. C'est une fonction, pour $n \in \mathbb{N}$ $V^n \rightarrow true, false$;
- Po un ensemble de *Postconditions*. Une postcondition est une expression booléenne évaluée en fonction des éléments de V. C'est une fonction, pour $n \in \mathbb{N}$ $V^n \rightarrow true, false$;
- Te un ensemble de *Tâches élementaires*. Une tâche élémentaire se compose :
  - d'une liste d'arguments (d'éléments de V) ;
  - d'une liste de postconditions ;
- Ta un ensemble de *Tâches abstraites*. Une tâche abstraite se compose :
  - d'une liste d'arguments (d'éléments de V) ;
  - d'une liste de méthodes non vide (auxquelles sont associées des préconditions) ;
  - d'une liste de postconditions ;
- M un ensemble de *Méthodes*. Une méthode $m \in M$ est un triplet $(Pm, st, \Re)$ avec :
  - $Pm \in \mathcal{P}(Pr)$ une liste de préconditions ;
  - st un ensemble de tâches (abstraites ou élémentaires), i.e d'éléments de $Ta \cup Te$;
  - $\Re$ une relation d'ordre partiel entre les éléments de st. $\Re$ permet de définir si une liste de tâches (un ensemble $\mathcal{P}(Ta \cup Te)$) doit s'exécuter de façon séquen-

tielle, parallèle, ou sans ordre particulier. $\Re$ peut par exemple définir des contraintes temporelles, telles que pour $(t_1, t_2) \in (Ta \cup Te)^2, t_1 \prec t_2$ signifie que t_1 doit être exécutée avant t_2.

Une tâche élémentaire correspond à une action que le véhicule devra exécuter. C'est à ce niveau que les arguments de la tâche élémentaire interviennent également. Ils correspondent alors aux paramètres associés à l'action. Exemple : "$goto\ wptA$", action "$goto$", appliquée avec le paramètre "$wptA$", en supposant que $wptA \in V$ et $goto \in E$.

Dans la section suivante, nous présentons notre architecture décisionnelle HiDDeN. Ensuite, nous abordons le problème des aléas, ainsi que les procédures de recouvrement associées. Enfin, nous présentons brièvement un scénario exemple sur lequel notre architecture est actuellement testée, afin de démontrer sa faisablilité et d'en évaluer les performances.

2 HiDDeN, une architecture décisionnelle distribuée

Les membres de l'équipe de robots à superviser sont tous supposés déjà *autonomes* : chacun embarque une *architecture de contrôle* qui lui est propre et qui lui confère une autonomie individuelle : ces architectures de contrôle possèdent déjà l'équivalent d'un superviseur individuel, qui leur permet d'accomplir les missions individuelles pour lesquelles le robot a été conçu. Nombre de véhicules autonomes sont actuellement opérationnels, et ont déjà subi un large éventail d'essais et de tests. Un enjeu majeur de notre travail est alors de pouvoir réutiliser au mieux ces capacités déjà opérationnelles, tout en leur permettant d'accomplir une mission en équipe.

Nous avons pour cela choisi de concevoir une architecture décisionnelle distribuée au sein de l'équipe de véhicules, que nous appelons HiDDeN (pour High-level Distributed DecisioN layer). Ce système distribué est constitué par un ensemble de *superviseurs locaux*, embarqués au sein des véhicules et interfacés avec leur architecture de contrôle respective (figure 1).

Cette architecture décisionnelle plus abstraite va permettre de choisir quelles actions doivent être exécutées par quels véhicules autonomes composant l'équipe, tout en contrôlant la coordination entre les véhicules. Chaque superviseur local est capable de communiquer avec les autres superviseurs locaux, en fonction de la disponibilité des moyens de communication.

Pour exécuter une mission, une équipe a besoin d'un plan. Celui-ci est supposé construit hors-ligne par un planificateur. Nous faisons l'hypothèse que HiDDeN dispose d'un plan hiérarchique instancié pour pouvoir lancer l'exécution de la mission. Ce plan initial est défini comme étant le *plan global d'équipe*. C'est sur ce plan que l'architecture décisionnelle plus abstraite s'appuie pour accomplir la mission.

Concrètement, en début de mission, le superviseur local reçoit le plan de mission global et va en extraire son *plan local* d'exécution : il s'agit de l'arbre HTNi particulier que le véhicule autonome doit exécuter. En effet, un véhicule n'est pas concerné par la totalité du plan global, et il n'a finalement besoin de connaître que les tâches dans lesquelles il est impliqué. Le plan est donc "distribué" aux différents véhicules, qui en extraient les parties qui leurs seront utiles.

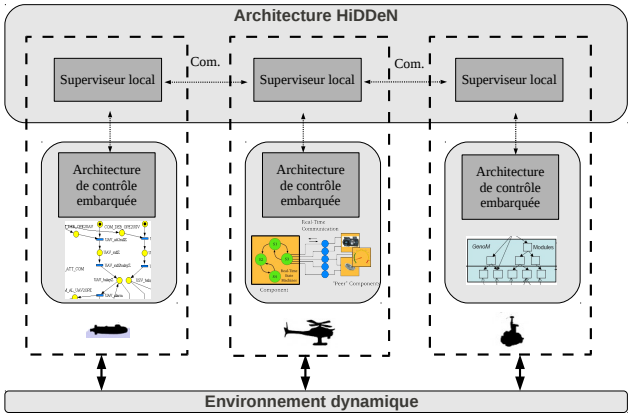


FIGURE 1 – L'architecture décisionnelle HiDDeN, superviseur l'équipe de véhicules

Le superviseur local "lance" l'exécution de la mission, via l'architecture de contrôle propre au robot en lui fournissant successivement les tâches élémentaires (élémentaires du point de vue du superviseur local) dont il a la charge, et suivant le plan HTNi local courant.

Le superviseur parcourt l'arbre HTNi de gauche à droite (sauf indication contraire, notamment dans le cas de tâches parallèles), en profondeur d'abord. Lorsqu'une tâche abstraite est atteinte, ses préconditions sont testées, et dans le cas où diverses méthodes sont disponibles, l'une d'entre elles dont les préconditions sont satisfaites est sélectionnée. Le sous-arbre HTNi de la méthode est ensuite parcouru de la même façon. Lorsqu'une tâche élémentaire (une feuille) est atteinte, le superviseur local a pour rôle de la faire exécuter par l'architecture propre du robot.

Plusieurs cas de figure apparaissent lorsque l'on souhaite "dialoguer" avec l'architecture de contrôle embarquée des véhicules, ce qui suit d'ailleurs logiquement l'hétérogénéité des architectures. En général, les architectures de contrôle disposent déjà d'un module de supervision plus ou moins explicite, voire d'une interface qui permet à un opérateur mission de s'y connecter. Cela permet de dérouler et d'exécuter le plan de mission défini. Le superviseur local prend alors "simplement" sa place : c'est lui qui dicte la tâche à accomplir, et attend le retour d'exécution. Il va ainsi traduire les tâches élémentaires figurant dans le HTNi en messages qui peuvent être interprétés par l'architecture de contrôle.

La figure 2 illustre le parcours et l'exécution d'un HTNi. La tâche racine à exécuter est R. Nous n'avons pas représenté dans cet exemple les méthodes, que nous supposons uniques par tâche abstraite. Le moteur d'exécution développe la tâche $T1$, puis la tâche $T2$. $E1$ doit être exécutée

en premier par le véhicule. Si l'exécution réussit, $E2$ est exécutée à son tour. A la fin de l'exécution de $E2$, si aucun aléa n'est survenu, $T1$ est supposée "terminée". Puis le moteur d'exécution développe ensuite le sous-arbre de la tâche $T3$, et ainsi de suite.

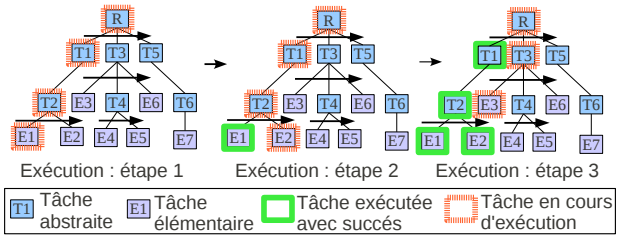


FIGURE 2 – Exécution d'un HTNi : l'ordre d'exécution des tâches élémentaires est $E_1, E_2, E_3, E_4, E_5, E_6, E_7$.

A bord d'un véhicule, le superviseur local transmet à l'architecture de contrôle les tâches à exécuter, ainsi que leurs éventuels arguments, en se basant sur le plan local courant du véhicule. Il récupère les résultats d'exécution des tâches et décide des actions à poursuivre : si l'exécution s'est bien déroulée, le plan peut continuer. En revanche, si une tâche échoue, c'est à lui de décider quoi faire. Le superviseur local peut notamment faire appel à des planificateurs embarqués ou demander à entrer en communication avec un autre vehicule autonome. Les mécanismes de réparation/re-planification seront traités section 3.
Après avoir exécuté l'action qui lui a été confiée (une tâche élémentaire du point de vue du superviseur local), l'architecture de contrôle renvoie un "accusé de réception" qui précise si l'action a pu être correctement menée, ou si un aléa est survenu durant son exécution.
Lorsqu'il n'y a pas d'aléa ou d'erreur, le processus d'exécution (géré par le superviseur local du véhicule) du HTNi local continue.
Lorsqu'un aléa survient, le superviseur local peut envoyer une requête de replanification à un module de planification embarqué, qui fournit alors les réparations locales de plan demandé, quand une telle chose est possible.

3 Gestion des aléas

3.1 Types d'aléas

Lors de l'exécution du plan de mission, des aléas peuvent avoir lieu. Nous proposons de prendre en compte dans le superviseur local quatre types d'aléas particuliers :

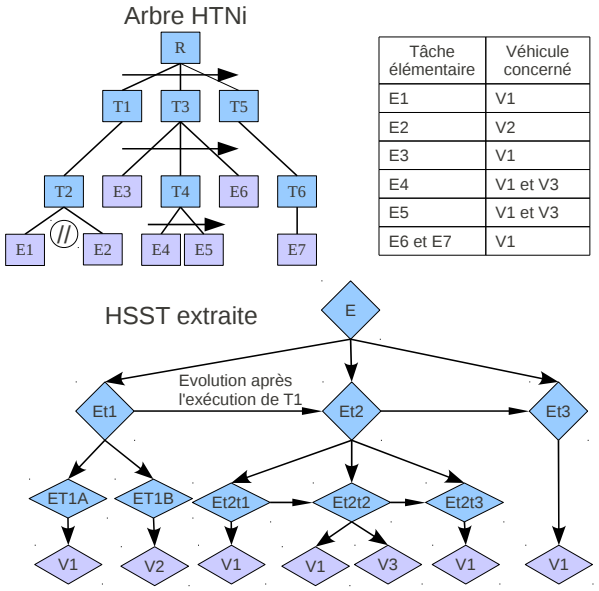
1. *Préconditions invalides* : Au moins une précondition de la tâche élémentaire en passe d'être exécutée n'est pas valide dans l'état courant (éventuellement estimé) du système.
2. *Echec d'exécution* : L'exécution d'une tâche élémentaire consiste à interagir avec l'agent physique concerné, en faisant appel à une fonction située dans une librairie, ou en exécutant un programme externe par exemple. Ainsi, l'exécution d'une action peut être considérée, du point de vue du superviseur local, comme l'exécution d'un morceau de code qui peut se terminer par un succès ou un échec. Dans les deux cas, un message (avec d'éventuels arguments) est renvoyé par l'architecture de contrôle du véhicule au superviseur local, qui peut détecter un éventuel aléa.
3. *Postconditions invalides* : Au moins une postcondition de la tâche élémentaire qui vient d'être exécutée n'est pas valide dans l'état courant (éventuellement estimé) du système.
4. *Timer terminé* : Avant de faire exécuter une tâche élémentaire à l'architecture de contrôle, le superviseur local arme un timer. Si l'action nécessite une durée supérieure à une certaine valeur (associée à la tâche, ou définie par défaut), l'exécution est considérée comme trop longue et interrompue.

Lorsqu'un aléa est détecté, un processus de réparation locale est déclenché.

3.2 Une réparation aussi locale que possible

Replanifier l'ensemble de la mission (coût élevé en temps et en processeur) à chaque occurence d'aléa n'est probablement pas la meilleure solution, d'autant plus que la communication entre robots n'est pas toujours disponible. L'approche que nous proposons consiste à tirer parti de la structure hiérarchique du HTNi et de celle d'une structure d'équipe, construite pour faciliter la réparation au sein de l'équipe.

Structure hiérarchique de sous-équipe. Lorsqu'une réparation est nécessaire, il peut être pertinent de savoir quel véhicule autonome est concerné par quelle tâche, mais également quel équipier est impliqué par l'aléa rencontré ou n'est simplement plus capable d'accomplir une tâche du plan dépendante de celle ayant échouée. Dans un cas idéal, tous les véhicules autonomes concernés devraient être avertis de l'aléa, et devraient pouvoir modifier leur plan d'exécution en fonction de cet aléa. Mais l'aspect incertain des communications rend la procédure plus ardue.
Notre but est de nous appuyer sur une structure organisée d'équipe, suffisamment robuste pour résister à des ruptures de communication et qui puisse évoluer dynamiquement tout au long de la mission. Tout cela dans l'intention de faciliter une réparation de plan en cas d'aléas.
Hors-ligne (i.e avant le début de la mission), une structure hiérarchique de sous-équipe (ou HSST - Hierarchical Structure of Sub-Teams) est extraite du plan local initial sous forme de DAG (Directed Acyclic Graph). L'algorithme chargé de sa construction est détaillé dans [23].
Cette structure autorise les véhicules autonomes à appartenir à plusieurs sous-équipes au même instant, et prend en compte des contraintes d'ordonnancement temporel qui peuvent exister entre les tâches durant leur exécution (deux sous-équipes peuvent notamment travailler simultanément sur des tâches différentes). La figure 3 illustre une évolution temporelle de sous-équipes.



Tâche élémentaire	Véhicule concerné
E1	V1
E2	V2
E3	V1
E4	V1 et V3
E5	V1 et V3
E6 et E7	V1

FIGURE 3 – Exemple de HSST, extraite de l'arbre HTNi situé en haut à gauche. La sous-équipe $Et1$ est chargée de la tâche $E1$ et $E2$ (tâches qui sont supposées être exécutées simultanément), respectivement confiées aux véhicules $V1$ et $V2$. Après l'exécution de $T1$, la structure est modifiée en $Et2$: tout d'abord, cette sous-équipe est uniquement constituée par $V1$, puis $V3$ rejoint la sous-équipe $Et2$ pour exécuter $T4$ ($E4$ et $E5$) en collaboration avec $V1$, et enfin, $V3$ quitte la sous-équipe puisque $E6$ implique seulement $V1$.

Il est à noter que la formation des sous-équipes n'est pas basée sur les contraintes de communication, mais plutôt orientée sur les tâches à accomplir : deux véhicules autonomes qui sont chargés de l'exécution d'une même tâche abstraite (à haut niveau) seront dans la même sous-équipe, même s'ils ne seront pas nécessairement capables de communiquer entre eux pendant l'exécution de leurs tâches élémentaires respectives [1].

Avec une telle structure, chaque véhicule à chaque instant est capable de savoir à quelle sous-équipe il appartient, en particulier pour restreindre une réparation éventuelle à la plus petite sous-équipe possible. Après une phase de réparation, la HSST doit éventuellement être mise à jour, suivant encore une fois les contraintes de communication (plus de détails dans [23]).

Réparation mono-robot. Lorsqu'un aléa est détecté par un véhicule v, nous proposons de réparer le plan aussi localement que possible. L'algorithme 1, dont le fonctionnement est illustré figure 4, décrit le processus d'exécution d'une tâche. L'exécution de la mission correspond à l'application de la fonction EXECUTE à la tâche racine du plan HTNi. L'exécution est différente suivant si la tâche est abstraite (ligne 1) ou élémentaire (ligne 14). Dans le premier cas, un test (ligne 3) permet de savoir s'il existe une méthode dont les préconditions sont valides. Dans cet ensemble, une méthode est choisie pour exécuter la tâche t (ligne 5). Un appel récursif de EXECUTE (ligne 8) est lancé sur l'ensemble des tâches st de m, suivant l'ordre $\Re$ de m. En cas d'échec (FAILURE), la procédure REPAIR(t) (ligne 9) est appelée, et correspond à une requête vers un planificateur : il s'agit de remplacer la partie du plan ayant échouée (dont la racine est la tâche courante t) par un plan alternatif, obtenu en lançant un planificateur sur un domaine et un problème particulier

Si un plan solution est trouvé, l'exécution peut se poursuivre. Sinon, le FAILURE est propagé (ligne 12), pour tenter une replanification de la première tâche hiérarchiquement supérieure (*backtrack* dans le HTNi). S'il s'agit de la tâche racine du HTNi, cela revient à une replanification quasi-totale de la mission, mais en tenant compte des buts déjà atteints.

Dans le second cas, (ligne 14), il s'agit d'envoyer l'action associée à la tâche appelée (rôle de la fonction '**call** t', ligne 17) à l'architecture de contrôle, tout en gérant la détection d'aléas potentiels, décrits à la section 3.1. Il est à noter que l'aléa de type *Timer terminé* n'est pas intégré à l'algorithme 1.

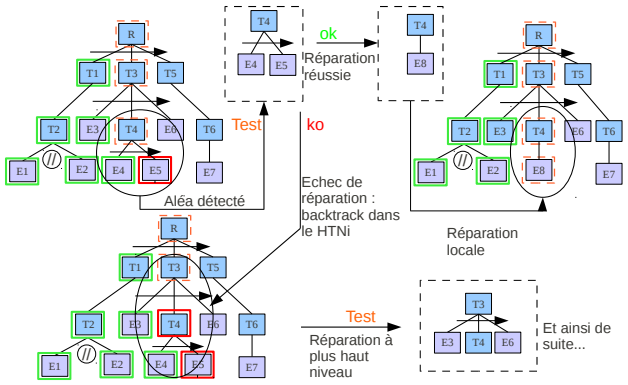


FIGURE 4 – Exemple de processus de réparation : la tâche élémentaire E_5 échoue, ce qui entraine une tentative de réparation de la tâche T_4 (en haut à gauche). Si un nouveau plan est disponible, la branche du HTNi correspondante est remplacée (en haut à droite) ; sinon, la tâche de plus haut niveau directement supérieure à T_4, ici T_3, est réparée (en bas).

Réparation de sous-équipe. Lorsque la réparation d'une tâche implique plus d'un véhicule, le superviseur local de v doit d'abord vérifier quels véhicules sont concernés, en s'auppuyant sur la HSST. Par exemple, sur la figure 3, si le véhicule $V1$ échoue lors de l'exécution de $E3$, et ne dispose pas d'un plan alternatif pour atteindre les objectifs de $E3$, le processus de réparation remet en cause $T3$. Et donc la sous-équipe $Et2$ (composée de $V1$ et $V3$) est impliquée, et $V1$ doit entrer en communication avec $V3$.

1. Une tâche élémentaire peut aussi consister en une tâche de communication ou peut représenter une action conjointe de plusieurs véhicules

Algorithme 1: EXECUTE(t)

```
1  if t ∈ T_A then
2      EM = ensemble des méthodes de t dont les
       préconditions sont valides
3      if EM = ∅ then
4          return FAILURE
5      m = (Pm, st, ℜ), un élément de EM (choix d'une
       méthode)
6      while st ≠ ∅ do
7          t' = min(st)
8          if EXECUTE(t') returns FAILURE then
9              if REPAIR(t) returns SUCCESS then
10                 return EXECUTE(t)
11             else
12                 return FAILURE
13         st = st − {t'}
14 else if t ∈ T_E then
15     if t's preconditions are not valid then
16         return FAILURE
17     else if call t returns FAILURE then
18         return FAILURE
19     else if t's effects are not valid then
20         return FAILURE
21 return SUCCESS
```

Soit S_t la sous-équipe associée à la tâche qui doit être réparée. Si la communication est possible entre tous les véhicules impliqués pour la tâche S_t, l'ensemble de la sous-équipe effectue une réparation de plan. Cette dernière peut s'effectuer de façon centralisée (v effectue la réparation, puis l'envoie à ses coéquipiers, dans S_t) ou distribuée, via des modèles de négociation ou d'enchères par exemple. Dans notre implémentation, nous avons choisi l'approche centralisée qui nous permet d'utiliser directement un planificateur embarqué classique.

Sinon, si la communication est indisponible, le véhicule doit effectuer une procédure de recouvrement de communication : une requête est envoyée vers son planificateur embarqué avec pour but l'obtention d'un plan qui autoriserait le véhicule à atteindre un état ou la disponibilité des communications est estimé "meilleure".

Une fois la réparation locale effectuée, la HSST est elle aussi possiblement modifiée (localement). Cependant, ce genre de modification concerne en général la même sous-équipe, et n'altère pas l'ensemble de la HSST.

Etant donné que les véhicules sont plongés dans un environnement réel, la valeur des variables de planification peut évoluer depuis la planification initiale hors-ligne. La réparation à effectuer par le planificateur embarqué est loin d'être triviale : l'état courant du véhicule a changé, l'environnement a été modifié (et pas uniquement à cause des actions du véhicule lui-même), et les aléas détectés doivent être pris en compte. Chaque véhicule autonome doit donc disposer de sa propre connaissance de l'environnement et de l'équipe dont il fait parti, y compris les plans, les états et la disponibilité des communications. Cette base de connaissance *distribuée* mène inévitablement à des problèmes de cohérence des données. Par exemple, certains véhicules autonomes peuvent avoir à disposition une connaissance différente du plan global de mission en fonction des aléas rencontrés et de la non disponibilité des communications. Pour limiter autant que possible l'impact de ces incohérences, au niveau du plan, la base de données de chaque véhicule autonome (et donc de son superviseur local) est restreinte aux tâches qui concerne le véhicule (le plan local) et n'utilise qu'une vision réduite de la structure d'équipe (limitée aux pères et aux oncles du robot dans la HSST). Cet ensemble de connaissances est la minimale requise pour être capable de réparer le plan de mission en suivant la procédure mentionnée ci-dessus.

4 Expérimentation

HiDDeN a été testé en simulation sur un scénario de blanchiment de chenal. La zone à "blanchir" est divisée en plusieurs sous-zones. Deux véhicules sont employés, un AAV (Autonomous Aerial Vehicle) à voilure tournante ainsi qu'un AUV (Autonomous Underwater Vehicle). Ils sont individuellements autonomes. La première étape de nettoyage d'une sous-zone consiste à en faire dégager les bâtiments de surface par l'AAV. Lorsqu'une sous-zone est vide de bâtiments de surface, l'AUV peut y intervenir. Il va, en plongée, effectuer un balayage du fond à la recherche de mines. S'il en détecte, l'AUV remonte en surface, puis transmet la localisation de la mine à un opérateur mission, l'AAV pouvant servir de relai de communication. Celles-ci sont en effet contraintes par la distance entre les véhicules, et par la nature des milieux (en plongée, on considère que l'AUV ne peut pas communiquer avec l'AAV). Une vidéo, disponible comme matériel supplémentaire, en montre une version simplifiée. A titre indicatif, en utilisant le même planificateur (comparable à SHOP2 [22]), et la même machine, le coût de la replanification lors du premier aléa était de 12 ms (modification mineure du plan) à comparer aux 8232 ms nécessaire pour la planification initiale (hors-ligne). Sans le processus de réparation local, le coût de replanification lors de l'occurence de ce même aléa aurait été de 7420 ms.

5 Conclusion

Nous avons conçu une architecture décisionnelle de haut niveau, HiDDeN, pour prendre en charge l'exécution de missions par des systèmes multi-robots, composés de véhicules individuellement autonomes mais non initialement conçus pour travailler ensemble. HiDDeN est distribuée sur les véhicules de l'équipe au travers de superviseurs locaux embarqués, eux-mêmes interfacés avec l'architecture

de contrôle propre du véhicule. HiDDeN prend en compte l'occurence d'aléas, ce qui lui permet d'effectuer des processus de recouvrement et de contribuer ainsi à la robustesse de l'équipe face aux aléas lors de l'exécution d'une mission. Pour les phases d'exécution et de réparation, les superviseurs locaux s'appuient sur un plan hiérarchique, un HTNi et une structure d'équipe, la HSST, ce qui leur permet d'effectuer des réparations de plan aussi locales que possible, mais aussi de limiter le recours aux communications.

HiDDeN est implémentée et prête à être embarqué pour des campagnes d'expérimentations sur robots réels, sur le scenario de blanchiment de chenal présenté, mais aussi sur des scénarios aéroterrestres impliquant jusqu'à douze véhicules. Nous voulons évaluer notre architecture à travers des métriques orientées robotique [24]), mais aussi tenant compte de paramètres tel que la structure du domaine HTN, la stabilité de nos plans successifs [14], la perte de véhicules, la durée des missions réelles, etc. Nous avons également plusieurs types de planificateurs en développement, ce qui souligne par ailleurs la généricité de notre approche.

Références

[1] R. Alami, R. Chatila, S. Fleury, M. Ghallab, and F. Ingrand, "An architecture for autonomy," *IJRR*, vol. 17, no. 4, 1998.

[2] C. McGann, F. Py, K. Rajan, H. Thomas, R. Henthorn, and R. McEwen, "A deliberative architecture for AUV control," in *ICRA'08*, Pasadena, CA, USA, 2008.

[3] P. Doherty, J. Kvarnström, and F. Heintz, "A temporal logic-based planning and execution monitoring framework for unmanned aircraft systems," *JAAMAS*, vol. 19, no. 3, 2009.

[4] Y. Watanabe, C. Lesire, A. Piquereau, P. Fabiani, M. Sanfourche, and G. Le Besnerais, "The Onera ReSSAC unmanned autonomous helicopter : Visual air-to-ground target tracking in an urban environment," in *AHS Forum*, Phoenix, AZ, USA, 2010.

[5] BEAR, "Berkeley aerobot (bear) project homepage," Website, 1996, http://robotics.eecs.berkeley.edu/bear.

[6] R. Vidal, O. Shakernia, H. J. Kim, D. H. Shim, and S. Sastry, "Probabilistic pursuit-evasion games : Theory, implementation and experimental evaluation," *IEEE T-RA*, vol. 18, 2002.

[7] L. E. Parker, "ALLIANCE : An Architecture for Fault Tolerant Multi-Robot Cooperation," *IEEE T-RA*, vol. 14, 1998.

[8] L. Chaimowicz, T. Sugar, V. Kumar, and M. Campos, "An architecture for tightly coupled multi-robot cooperation," *ICRA'01*, 2001.

[9] K. Erol, J. Hendler, and D. Nau, "HTN planning : complexity and expressivity," in *AAAI'94*, Seattle, WA, USA, 1994.

[10] M. Paolucci, O. Shehory, and K. Sycara, "Interleaving planning and execution in a multiagent team planning environment," *Linköping Elec. Articles in CIS*, vol. 5, no. 18, 2000.

[11] N. Fazil Ayan, U. Kuter, F. Yaman, and R. Goldman, "HOTRiDE : hierarchical ordered task replanning in dynamic environments," in *ICAPS'07*, Providence, RI, USA, 2007.

[12] O. Bonnet-Torrès and C. Tessier, "From teamplan to individual plans : a Petri net-based approach," in *AAMAS'05*, Utrecht, The Netherlands, 2005.

[13] G. Boella and R. Damiano, "A replanning algorithm for decision theoretic hierarchical planning : principles and empirical evaluation," *Applied Artificial Intelligence*, vol. 22 : 10, 2008.

[14] M. Fox, A. Gerevini, D. Long, and I. Serina, "Plan stability : Replanning versus plan repair," in *ICAPS*, 2006.

[15] J. Dix, H. Muñoz-Avila, D. Nau, and L. Zhang, "IMPACTing SHOP : putting an AI planner into a multi-agent environment," *Annals of Mathematics and AI*, vol. 37, no. 4, 2003.

[16] M. Tambe, "Towards flexible teamwork," *JAIR*, vol. 7, 1997.

[17] M. Magnusson, D. Landén, and P. Doherty, "Planning, executing, and monitoring communication in a logic-based multi-agent system," in *ECAI*, Patras, Greece, 2008.

[18] S. Joyeux, R. Alami, and S. Lacroix, "A plan manager for multi-robot systems," in *FSRS*, Chamonix, France, 2007.

[19] R. Micalizio and P. Torasso, "Supervision and diagnosis of joint actions in multi-agent plans," in *AAMAS*, Estoril, Portugal, 2008.

[20] F. Legras and C. Tessier, "LOTTO : group formation by overhearing in large teams," in *AAMAS*, Melbourne, Australia, 2003.

[21] C. C. Sotzing, N. Johnson, and D. M. Lane, "Improving multi-AUV coordination with hierarchical blackboard-based plan representation," in *PlanSIG*, Heriot-Watt University, Edinburgh, 2008.

[22] D. Nau, T.-C. Au, O. Ilhami, U. Kuter, W. Murdock, D. Wu, and F. Yaman, "SHOP2 : an HTN planning system," *JAIR*, vol. 20, 2003.

[23] T. Gateau, C. Lesire, and M. Barbier, "Local plan execution and repair in a hierarchical structure of sub-teams of heterogeneous autonomous vehicles," in *ICAPS'10 DC*, Toronto, Canada, 2010.

[24] B. Kannan and L. E. Parker, "Metrics for quantifying system performance in intelligent, fault-tolerant multi-robot systems," in *IROS'07*, San Diego, CA, USA, 2007.