Université de Toulouse

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :

Institut Supérieur de l'Aéronautique et de l'Espace (ISAE)

Présentée et soutenue par :
Thibault GATEAU

le mardi 11 décembre 2012

Titre :

Supervision de mission pour une équipe de véhicules autonomes hétérogènes

---

École doctorale et discipline ou spécialité :
EDSYS : Systèmes embarqués

Unité de recherche :
Équipe d'accueil ISAE-ONERA CSDV

Directeur(s) de Thèse :

Mme Magali BARBIER - Directrice de thèse
M. Charles LESIRE - Co-directeur de thèse

Jury :
M. Rachid ALAMI - Examinateur
Mme Magali BARBIER - Directeur de thèse
Mme Amal EL FALLAH - SEGHROUCHNI - Rapporteur
M. Bruno JOUVENCEL - Rapporteur
M. Charles LESIRE - Examinateur
M. Serge STINCKWICH - Examinateur

Résumé

Ces dernières années, les engins robotisés n'ont cessé d'améliorer leur autonomie dans le domaine de la décision. Désormais, pour ne citer que l'exemple de véhicules aériens, nombre de drones sont largement capables, sans intervention d'un opérateur humain, de décoller, suivre un itinéraire en activant divers capteurs à des moments précis, atterrir en un lieu spécifié, suivre une cible, patrouiller sur une zone... Une des étapes suivantes consiste à faire collaborer une équipe de véhicules autonomes, de nature hétérogène (aériens, terrestres, marins...) afin de leur permettre d'accomplir des missions plus complexes. L'aspect dynamique de l'environnement réel, la non disponibilité à tout instant des moyens de communication, la coordination nécessaire des véhicules, de conceptions parfois différentes, dans l'exécution de certaines parties d'un plan de mission, sont autant d'obstacles à surmonter. Ce travail tente non seulement d'apporter quelques éléments de réponse face à ces difficultés, mais consiste aussi en la mise en place concrète d'un superviseur haut niveau, capable de gérer l'exécution d'une mission par une équipe de véhicules autonomes hétérogènes, où le rôle de l'opérateur humain est volontairement réduit.
Nous décrivons dans ce mémoire l'architecture distribuée que nous avons choisi de mettre en œuvre pour répondre à ce problème. Il s'agit d'un superviseur, réparti à bord des véhicules autonomes, interfacé avec leur architecture locale et en charge de l'exécution de la mission d'équipe. Nous nous intéressons également à la formalisation des connaissances nécessaires au déroulement de cette mission, afin d'améliorer l'interopérabilité des véhicules de l'équipe, mais aussi pour expliciter les relations entre modèles décisionnels abstraits et réalité d'exécution concrète. Le superviseur est capable de réagir face aux aléas qui vont se produire dans un environnement dynamique. Nous présentons ainsi dans un second temps les stratégies mises en place pour parvenir à les détecter au mieux, ainsi que la façon dont nous procédons pour réparer partiellement ou totalement le plan de mission initial, afin de remplir les objectifs initiaux. Nous nous basons notamment sur la nature hiérarchique du plan de mission, mais aussi sur celle de la structure de sous-équipes que nous proposons de construire. Enfin, nous présentons quelques résultats obtenus expérimentalement, sur des missions simulées et des scénarios réels, notamment ceux du Programme d'Etudes Amont Action dans lequel s'inscrivent ces travaux de thèse.

Mots-clés : Architecture décisionnelle, coopération multirobot, replanification, véhicules autonomes hétérogènes, formalisation de connaissances.

Remerciements

Et le moment d'écrire les remerciements est arrivé ! Moment que j'attendais impatiemment (le reste est relu, ré-écrit, re-relu, re-ré-écrit,..., validé, envoyé, défendu, plié) et avec appréhension (sisi !) : non seulement il faut encore écrire, et en plus, faire en sorte de limiter au mieux le nombre de personnes oubliées. Car au cours de ces trois années, nombreux sont ceux que j'aimerais remercier, et je m'en voudrais de ne pas leur faire honneur ici, même si ces quelques lignes ne pourront jamais refléter toute la reconnaissance que j'ai pour eux.

A commencer par mes directeurs de thèse, Magali et Charles, qui sont probablement les meilleurs directeurs qu'un thésard puisse rêver avoir ! A la fois complémentaires, efficaces, disponibles, patients, au regard critique pertinent et constructif, doués d'une habileté étonnante à convertir un paragraphe constitué d'une bouillie de mots-clés sous-entendus en une phrase synthétique et à une fabuleuse capacité à prendre du recul, on va dire que j'ai eu beaucoup de chance de les rencontrer (sans compter leur gentillesse, la formidable machine anti-faute orthographique de Magali et la résolution temps-réel magique de gros bugs critiques de Charles !). J'ai énormément appris auprès de vous deux, et si les petits hélicos ont largement influencé mes choix dans mon premier stage de M2R, c'est bien grâce à vous que j'ai voulu continuer vers la voie de la thèse. Un immense merci donc, de m'avoir accompagné sur ces trois années et trois quarts. Et j'espère bien que l'on aura maintes occasions de se revoir !

Merci également à mes rapporteurs, déjà d'avoir accepté de rapporter ma thèse (et de m'avoir autorisé à soutenir !). A Bruno pour toutes ses remarques constructives, et à Amal, de m'avoir fait prendre conscience d'approches alternatives qu'il me reste à explorer. Merci également à mes deux examinateurs extérieurs, Rachid et Serge, non seulement d'avoir accepté de jouer ce rôle, mais aussi pour leurs remarques critiques, à la fois pertinentes et encourageantes.

Merci à mes deux co-bureaux latéraux, "codeurs - débuggers" hors pair, de m'avoir supporté tout ce temps, Pascal (et ses jeux de mots, ses convictions transmises, son imitation ultime de chewbacca / cousin machin) et NoWiS (qui m'a généreusement laissé lui emprunter tous ses aimants pour pouvoir finir ma thèse, dans une ambiance calme et nutritive). Merci pour votre patience, nos discussions, et bien d'autres choses...

Merci aussi à mes co-bureaux "d'en face" pour les bon moments avec eux, Quentin (une personne ensorcelée par la sorcière, peut-elle être tout de même volée ?), Fred (et ses petits satellites qui se font agresser sauvagement) et le Général Fizo (qui a osé tenter me détourner de ma rédaction).

Merci à mes "mentors" thésards de m'avoir montré la voie à suivre, Jack (et ouais, j'ai connu Jack quoi ! Vous êtes tous jaloux !... Sinon, quand est-ce qu'on retourne dans les lacs de l'Ontario ?), Alex (avec qui on n'a toujours pas fini "grow ver de terre" et que je déteste (presque) toujours autant), Zulie (qui a fini de démonter les petites alouettes pour passer aux gros camions), Mario (El doctor Mario ! gracias por las memorables "soirées" tequila, y todos los buenos momentos), mais aussi, Stéphane, Patrice, Mickaël, Mathieu...

Merci à mes petits camarades thésards, plongés dans le même bain que moi, Caro (son répertoire pompette, et ma guide favorite dans un pays magique, le Brésil !), Laure (fournisseuse officielle de posters graves trop sympas), Lara (une corde à nœud reliée à un poids, ça suffit largement pour avoir l'altitude en un point donné... ou avec un hobbit estimateur accroché en bout d'aile...)

Merci à "mes" stagiaires, Damien et Nico, d'avoir réalisé un planificateur qui ne se contente pas de déplacer des blocs sur une table virtuelle ;), j'ai sincèrement passé de bons moments à travailler à vos côtés, beaucoup appris auprès de vous mais aussi agréablement discuté. Et dans le désordre, merci aux "petits" nouveaux (ou moins nouveaux) Gaëtan (t'imagine le nombre de services à définir pour un Terminator ?), Sergio (elle est bientôt prête ton armée d'orcs ?), Pierre (et

son splendide banc de bois), Sylvain (le jujitsu, c'est cool !), Florian, Nico, Yann, Razvan, Pierre, Aurélie, Jérémy, Jeff, Nicolas (encore un !), Et à tous les autres, Nico et ses imprimantes 3D, Ryan, Emma, Myriam, Clément (et son chapeau), Dider, Béatrice (d'ailleurs où en est l'expérimentation "bières" ?), Benjamin Nathalie, Seb, Medhi, Johannes et les autres avec qui j'ai aussi eu l'occasion de passer de bons moments pendant mes phases de débogage.

Merci aux permanents de m'avoir si bien accueilli parmi eux, pour leur gentillesse, leur soutien moral et scientifique. Dans le désordre, Florent (mon super compagnon de découverte de la vie Atibaïanaise !) Patrick (de m'avoir accueilli dans son département, et de tous ses conseils pour parler d'un prob... heuu d'un "truc" à son chef :p), Charles (et ses petits coup vicieux au volley), Michel (et ses sauvegardes automatiques qui m'ont rendu bien des services !), Jean-François (vive les Petri nets Procoséens !), Jean-Loup (vive les ontologies !), Cédric, Gérard (et en particulier de m'avoir fait découvrir que les CSP c'est cool aussi !), Guillaume, Patrick, Guillaume, Frank, Eric, Mathieu, Vincent, Xavier, Clément (mon super compagnon d'escalade, avec Thibault Volpert, que j'ai lâchement abandonné à ma troisième année !), Clément et Christelle (qui m'ont indirectement permis de me nourrir de "vrais" légumes pendant ma thèse), Christelle, et merci aussi à Willy (pour les excellents légumes en question !) Merci à Catherine, tant pour le côté recherche qu'administratif (et à bientôt côté opérateur humain, ce grand absent du manuscrit !) Et également un grand merci à Serge, Françoise, Valérie, et Monique Perron pour leur gentillesse à tous, leur amabilité et toutes les complications administratives qu'ils m'ont évitées ! Merci à l'équipe hélico, Alain, Vincent, Henri, Pierre, Yoko, Alex et Paul (By the way, Lost in Time c'est fort sympathique, bonne recommandation !) et la bonne ambiance des expés Caylus, qui ont osé laisser leur joujou suivre les directives d'une machine pas toujours déterministe.

Merci à Olivier et Romain (mes compagnons secouristes :)) et à Thomas, Jérémy et Xavier (mes PC-men favoris !) Merci à Charles (qui préfère partager équitablement sa cave plutôt que de se siffler ses bouteilles, où va le monde :/), Angela, et mes autres compagnons vacataires. Un grand merci également à Christophe (Garion) de m'avoir accepté comme vacataire POO et fait confiance pendant 3 ans, et qui m'a permis de découvrir que l'enseignement, c'est cool !

Merci aussi à l'équipe Volley qui m'a permis de me dépenser entre deux articles. Merci mon comité de mi-thèse, avec en plus de ceux que j'ai déjà cités, Félix et Pierre. Vous m'avez fourni un bon nombre de remarques constructives (et qui m'ont permis d'entamer ma troisième année avec encore plus de boulot, mais ça en valait la peine !)

Merci à Bruno, mon compagnon favori de parapente, avec qui j'espère revoler bientôt (quel temps pourri sérieux... !) Merci aussi à l'équipe du poste de garde pour leur gentillesse (et Sorus, le 6 rax c'est vraiment bourrin quand même...)

Merci à l'équipe LAAS Action-men, Simon, Arturo (et merci encore pour la transaction brésilienne !), Redouan, Arnaud, Hung, et les deux Cyrils. Merci à l'équipe du CE, Sylvie, Miloud, Christiane et Anthony. Merci à l'Equipe Edsys / Isae Supaéro, et en particulier à Caroline, Hélène et Annie. Merci à Christiane (Bourdiol) pour son efficacité et à Claude (Doriac) pour sa gentillesse extraordinaire, sa patience avec tous ces thésards qui veulent toujours tout au dernier moment, et ses belles photos. Merci aussi à Daniel (Miasnik) pour le soutien logistique. Merci à Arnaud, Etienne et Serge pour mon niveau 1 de plongée (mais ça reste froid le Cap d'Agde...) qui m'ont permis de me relaxer entre deux papiers. Merci à Jean-Christophe Buisson qui a été le premier à me dire que faire une thèse, c'était cool et de m'avoir soutenu dans ce projet. Merci aussi à tous les autres, et à ceux que j'aurais oubliés et qui ont de près ou de loin contribué à cette thèse !

Merci enfin à plein de choses qui ne liront jamais ces lignes, au RMax/Ressac (sans lequel je ne serais peut être pas venu), au Projet-Program d'Etude-z-Amont Action (sans lequel je n'aurais

pas eu de sous), aux Simpsons (qui m'ont permis d'atteindre le seuil de conscience minimal, dixit Calvin, quand je ne pouvais plus écrire).

Enfin, merci à toute ma famille, qui a supporté mon humeur massacrante lors de ma rédaction. Merci à mes parents : vous m'avez toujours soutenu dans mes projets, toujours été là quand j'avais besoin de vous, et j'ai vraiment beaucoup de chance de vous avoir. Il est rare de pouvoir remercier ainsi les gens qu'on aime, alors j'en profite : papa, maman, merci sincèrement pour tout. Merci aussi à toi sœurette de me supporter, de m'accueillir outre-atlantique ou plus localement, de râler sur les robots, et pour toute la gentillesse dont tu fais preuve sans forcement t'en apercevoir, d'aimer le cacao, de partir dans des zones perdues qui sont grâves cools à visiter (ou à éviter :p), et tout plein de bonnes choses. Donc je te le dis ici, c'est une bonne occasion, merci aussi pour tout, même si dans ce "tout" je serais bien incapable d'énumérer ce pourquoi je te suis reconnaissant ! Merci aussi à mes grand-parents, à mes papy-mamy Gateau, pour leur gentillesse, soutien et compréhension, à ma mamy Anne de venir par surprise assister à ma soutenance, et à mon papy Anne qui aurait surement trouvé toute cette thèse bien formidable.

Et merci surtout à mon Iepu, sans qui cette thèse n'aurait purement et simplement jamais abouti ! Au cours de ces trois années de thèses communes, tu m'as tellement soutenu, aidé, relu, supporté qu'au risque de me répéter, je ne pourrais jamais exprimer toute la reconnaissance que j'ai pour toi. Merci pour les événements exceptionnels passés à tes côtés (les superbes voyages, allant de notre terrasse aux Amériques) mais aussi pour le quotidien, les phases de rédaction épiques, d'articles et de mémoire. Ce n'est pas facile de supporter un compagnon thésard (je peux en témoigner :p) donc merci beaucoup à toi que j'aime.

Pour conclure, et pour tout le monde, sincèrement, MERCI !

Finalement, écrire des remerciements, ce n'est pas si terrible (en fait si, c'est horrible mais une fois qu'on s'y met, ça va, c'est un peu comme le pot de thèse...). Et si je ne t'ai pas cité cher lecteur, merci d'avoir lu tout cela, tu peux rajouter ton nom ici pour ne pas te sentir vexé, _____, et filer à la page 105 où on aperçoit les trucs les plus cools de ces pages, avant de caler ton écran avec ce manuscrit : ils sont beaux ces petits robots quand même !

Table des matières

Table des figures

Liste des tableaux

Glossaire

AAV	Autonomous Aerial Vehicle (véhicule aérien autonome)
AGV	Autonomous Ground Vehicle (véhicule terrestre autonome)
ALFUS	Autonomy Levels For Unmanned Systems
ASV	Autonomous Surface Vehicle (véhicule de surface autonome)
AUV	Autonomous Underwater Vehicle (véhicule sous-marin autonome)
AXV	Véhicule autonome (AAV, AUV, ASV ou AGV)
BDI	Belief-Desire-Intention
BEAR	BErkeley AeRobot
CORBA	Common Object Request Broker Architecture
DGA	Direction Générale de l'Armement
CNRS	Centre National de la Recherche Scientifique
CSP	Constraint Satisfaction Problem (Problème de satisfaction de contrainte)
ESA	European Space Agency (agence spatiale européenne)
GPS	Global Positioning System
HARPIC	Hybrid Architecture based on Representations, Perception and Intelligent Control
HFSM	Hierarchical Finite State Machines (automates finis hiérarchisés)
HiDDeN	High-level Distributed DecisioN layer
HSST	Hierarchical Structure of Sub-Teams
HTN	Hierarchical Task Network
HTNi	Plan HTN instancié
IA	Intelligence Artificielle
IDEA	Intelligent Distributed Execution Architecture
IPC	International Planning Competition
Koper	Knowledge base for planning, execution and repair
LAAS	Laboratoire d'Analyse et d'Architecture des Systèmes du CNRS
LAN	Local Area Network (réseau local)
Lidar	Light Detection And Ranging (télédétection par laser)
NASA	National Aeronautics and Space Administration
MORSE	Modular OpenRobots Simulation Engine

MTBF	Mean Time Between Failures
ONR	Office of Naval Research (Office de la Recherche Navale Américaine)
Orocos	Open Robot Control Software
OWL	Web Ontology Language
P2P	Peer-to-peer (pair à pair)
PC	Personal Computer (ordinateur personnel)
PDA	Personal Digital Assistant (assistant numérique personnel)
PDDL	Planning Domain Definition Language
PEA	Programme d'Etudes Amont
POO	Programmation Orientée Objet
POSIX	Portable Operating System Interface
ProCoSA®	Programmation et Conduite de Systèmes à forte Autonomie
REA	Rapid Environmental Assessment
ReSSAC	Recherche sur les Systèmes Sûrs Autonomes et Coopérants
SLAM	Simultaneous Localization And Mapping
SNLE	Sous-marin Nucléaire Lanceur d'Engins
SMA	Système Multi Agents
UAV	Unmanned Aerial Vehicle (véhicule aérien sans pilote)
UML	Unified Modeling Language
W3C	World Wide Web Consortium
WGS 84	World Geodetic System 1984 (système géodésique mondial, révision de 1984)
WLAN	Wireless Local Area Network (réseau local sans fil)
XML	eXtensible Markup Language
YARP	Yet Another Robot Platform

Introduction

L'utilisation de véhicules sans pilote n'a cessé de s'accroître durant ces vingt dernières années. De simples objets télécommandés à l'origine, ils sont désormais assez complexes pour accomplir seuls nombre de tâches autrefois télé-opérées. Le niveau de maturité de leur autonomie décisionnelle amène désormais à s'interroger sur leur capacité à collaborer non pas uniquement avec un opérateur humain, mais au sein d'une équipe de véhicules autonomes qui disposeraient de capacités de déplacement, de perception, de communication et de calculs, complémentaires et redondantes.

Les architectures logicielles actuellement embarquées sont capables de faire réaliser à un robot individuel des tâches complexes : suivi de cible, balayage de zones à la recherche d'un point d'intérêt, mission d'observation suivant un itinéraire, etc. Certains robots sont également capables d'interagir avec d'autres robots et d'effectuer des tâches conjointement. Il serait intéressant de pouvoir réutiliser les capacités opérationnelles déjà développées afin de permettre à une équipe de robots, individuellement déjà autonomes, de mener à bien une mission plus complexe. En effet, un travail collaboratif peut présenter certains intérêts indéniables, comme l'accomplissement d'actions en parallèle, l'augmentation potentielle de la robustesse du système et/ou de son efficacité en offrant plus de redondance de par la présence de véhicules supplémentaires, l'exploitation de la complémentarité de plusieurs véhicules, du point de vue capacités ou milieux d'intervention, etc.

De plus, bien des architectures sont capables de réagir lorsque l'exécution du plan de mission ne se déroule pas de façon nominale. En effet, en conditions réelles, il n'est pas rare de rencontrer des *aléas*. Les véhicules autonomes doivent donc être capables de faire face, dans la mesure du possible, à ces imprévus pour atteindre leurs objectifs de mission. Par exemple, sur un processus de suivi d'itinéraire, un robot sait, dans certaines situations, replanifier seul un chemin lorsqu'il rencontre un obstacle sur son trajet initialement défini. Les architectures embarquées n'ont cependant pas toujours individuellement "l'ouverture d'esprit" suffisante pour répondre à l'aléa et atteindre les objectifs globaux de mission : des prises de décision à l'échelle de l'équipe sont nécessaires. Autant replanifier pour un seul robot peut ne pas poser trop de difficultés, autant modifier un plan qui concerne plusieurs robots est loin d'être un problème trivial : les réparations du plan de mission effectuées au sein d'un des véhicules peuvent notamment rendre la suite du plan incohérente pour les autres véhicules. Lors de l'occurrence d'un aléa sur un véhicule, il est nécessaire de prendre en compte son impact sur les coéquipiers de ce véhicule. Une communication minimale entre les

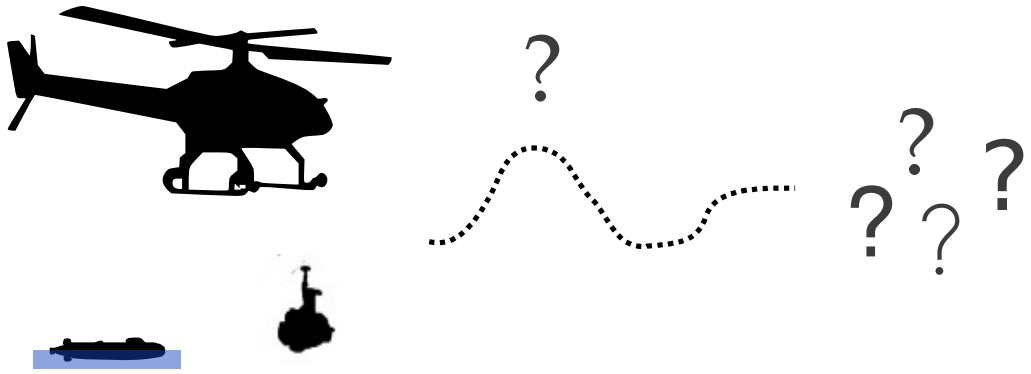
engins s'avère nécessaire pour cette coopération dans l'équipe. Cependant, l'environnement au sein duquel évolue l'équipe ne permet pas à chaque véhicule de pouvoir communiquer avec les autres à tout instant. La prise en considération des communications est donc un point critique, notamment pour mettre à jour les connaissances des véhicules suite à un rendez-vous planifié, mais aussi pour permettre à l'équipe de gérer les aléas.

L'objectif de cette thèse est de mettre en place un système de supervision de haut niveau. Il doit être capable de gérer l'exécution d'une mission par une équipe de véhicules autonomes hétérogènes, dans laquelle le rôle de l'opérateur humain est réduit au minimum, et dans un environnement réel et dynamique, où les communications sont incertaines.

Dans un premier temps, après avoir précisé le contexte, nous synthétisons un état de l'art sur l'autonomie décisionnelle, les architectures de contrôle embarquées, les problèmes de communication récurrents mais aussi sur les stratégies de réparation en ligne de plan de mission. Ensuite, nous présentons les spécifications auxquelles doit répondre le système de supervision d'équipe dans notre cas d'étude, et une première ébauche de l'architecture distribuée associée. Nous évoquons le formalisme que nous utilisons pour définir le plan de la mission, mais aussi la manière dont s'interfacent système de supervision et architectures individuelles des véhicules. Nous nous intéressons à la formalisation des connaissances au sein de l'équipe à travers Koper, la base de connaissance que nous avons développé pour notre système de supervision. Puis nous nous focalisons sur le fonctionnement de notre système de supervision pendant le déroulement de la mission. Nous introduisons alors la problématique liée aux aléas, et à leur détection. Nous décrivons ensuite les stratégies de réparation de plans que nous proposons, basées sur la nature hiérarchique des plans, mais également sur celle de la structure d'équipe. Celles-ci nous permettent notamment de fournir des solutions au regard des contraintes imposées par l'incertitude des communications. Enfin, nous présentons une évaluation des différentes stratégies précédemment citées au travers de simulations sur différents scénarios, académiques ou issus du projet Action, pour finir par un retour sur les expérimentations réelles auxquelles nous avons pu participer. Finalement, nous conclurons en donnant les perspectives envisagées pour notre travail.

CHAPITRE 1

Contexte



Ce chapitre permet de situer le cadre de ces travaux. Nous précisons ce que nous désignons par "supervision d'équipe", "mission" et "autonomie". Nous explicitons aussi à quels types de véhicules et de scénarios nous nous référons, ainsi que les objectifs généraux de ce mémoire.

Sommaire

1.1 Supervision d'équipe

Superviser, c'est "contrôler la réalisation d'un travail accompli par d'autres" [Petit Larousse, 2011]. Dans ce travail, la **supervision** est le contrôle de la réalisation d'une mission tout au long de son exécution, ainsi que de la partie décisionnelle lorsque la mission ne se déroule pas comme prévue.

Nous désignons par "**équipe**" l'ensemble des véhicules autonomes responsables de l'accomplissement de la mission.

Une **mission** est définie comme un ensemble d'objectifs à atteindre, assignés à une équipe de véhicules autonomes. Les véhicules, grâce à leurs capacités individuelles, complémentaires et/ou conjointes, sont alors capables d'interagir avec leur environnement, et ainsi parvenir (lorque c'est réalisable) à accomplir la mission. Le *superviseur* confère à l'équipe son autonomie décisionnelle haut niveau, dans le cadre d'une exécution nominale, mais aussi en cas d'aléas.

Les véhicules que nous considérons sont de nature hétérogène. Ils sont *a priori* aériens à voilure fixe ou tournante, terrestres, sous-marins, de surface, mais rien n'empêcherait de considérer des véhicules spatiaux (satellites, sondes, véhicules de rentrée atmosphérique...) Ils sont dépourvus d'équipage. Un drone [1] est communément considéré comme un aéronef sans pilote humain à bord, capable d'emporter une charge utile, et qui est destiné à des missions civiles ou militaires, généralement de surveillance, de renseignement, voire de combat.

Le lecteur pourra notamment se référer à [Lefevre, 2008] pour un historique sur le sujet. En français, le terme de *drone* est considéré en pratique suffisamment générique pour désigner l'ensemble des véhicules qui nous intéressent. Cependant, rien ne spécifie s'il s'agit de véhicules télé-opérés ou autonomes. Pour la suite, nous préférerons donc le terme de *véhicule autonome* et les acronymes américains AAV (Autonomous Aerial Vehicle), AGV (Autonomous Ground Vehicle), ASV (Autonomous Surface Vehicle) et AUV (Autonomous Underwater Vehicle).

Formant une équipe, les véhicules autonomes sont amenés à interagir. Les interactions au sein de systèmes multirobots sont variées. L. Parker [Parker, 2008] propose de catégoriser les systèmes d'intelligence distribuée suivant le type d'interactions entre véhicules (cf. tableau 1.1), soulignant par ailleurs les différences qui existent au sein des systèmes distribués robotiques. Ces systèmes peuvent ainsi être :

- *collectifs* : les entités ne sont pas conscientes de la présence des autres entités, mais elles partagent des buts et leurs actions contribuent aux actions des autres entités ; en multirobot, il s'agit de la robotique en essaim (swarms), le déplacement en formation...
- *coordonnés* : les entités sont conscientes de la présence des autres entités, elles n'ont pas de but commun et leurs actions ne contribuent pas à la satisfaction des buts des autres entités ; les robots doivent coordonner leurs actions pour éviter les interférences, en planifiant leur trajectoire, en appliquant des règles de contrôle du trafic ;
- *collaboratifs* : les entités sont conscientes de la présence des autres entités, elles ont des buts individuels et leurs actions contribuent à la satisfaction des buts des autres entités ; par exemple, des robots partagent leurs capacités de perception afin que chaque robot rejoigne sa propre position objectif ; la formation de coalition [Parker and Tang, 2006, Vig and Adams, 2006] est un moyen de réaliser cette collaboration ;
- *coopératifs* : les entités sont conscientes de la présence des autres entités, elles partagent des buts et leurs actions contribuent aux actions des autres entités ; par exemple, plusieurs robots

1. En anglais "faux bourdon", terme qui proviendrait des DH.82 Tiger Moth automatisés et utilisés alors pour servir d'avions-cibles.

travaillent ensemble et raisonnent pour réaliser un but commun : pousser une boîte, nettoyer un site, réaliser une mission de recherche et sauvetage...

TABLEAU 1.1 – Classement suivant les types d'interactions dans un système distribué multirobot [Parker, 2008].

Qualification de l'interaction	Connaissance des autres	Buts partagés	Dépendance des actions des autres pour la réalisation d'un but
Collective	non	oui	oui
Coordonnée	oui	non	non
Collaborative	oui	non	oui
Coopérative	oui	oui	oui

Dans ce travail, nous supposons que les véhicules qui réalisent la mission "ne sont pas intentionnellement adversaires et ne fournissent pas volontairement d'informations erronées" (une des hypothèses des travaux de L. Parker [Parker, 1998]), donc non compétitifs. Leur hétérogénéité et leur complexité comportementale potentielle dépassent largement le cadre de l'essaim, dans lequel les entités ne requièrent pas d'interactions complexes et sont toutes interchangeables. Les véhicules ont connaissance de l'existence de leurs coéquipiers (*awarness of others*, [Parker, 2008]) et dans une certaine mesure, des objectifs de ces coéquipiers.

Suivant les types de mission, une simple coordination suffirait parfois pour accomplir les objectifs. Dans d'autres cas, seule une coopération totale entre tous les véhicules permet de les atteindre. Nous nous placerons donc ici dans le cadre *coopératif* : les missions prises en exemple concernent une équipe de véhicules hétérogènes, dotés de capacités complémentaires, et chargés d'exécuter un certain nombre de tâches plus ou moins complexes. Certaines tâches étant réalisées conjointement, ils ont des buts sont partagés. De plus, tous les véhicules ont un objectif en commun, le succès de la mission.

1.2 Autonomie décisionnelle

Nous entendons par **autonomie décisionnelle** la capacité pour un système à prendre une décision sans intervention humaine. Ce facteur dépend de nombreuses caractéristiques intrinsèques des véhicules, de la complexité de l'environnement dans lequel la mission se déroule et des types d'interactions avec l'opérateur humain, comme le précise la contribution ALFUS (Autonomy Levels For Unmanned Systems) [Huang, 2007]. En effet, pour qu'un véhicule inhabité puisse disposer d'une véritable autonomie décisionnelle, un système logiciel embarqué doit remplacer le rôle qu'aurait traditionnellement joué un opérateur humain qui l'aurait piloté (directement à bord, ou de façon télé-opérative) : celui de la prise de décision des actions à faire exécuter à ce véhicule. Il s'agit de l'architecture de contrôle du véhicule, qui, de façon plus ou moins complexe, interagissant avec les capteurs et les actionneurs du véhicule, lui permet de réaliser les tâches pour lesquelles le véhicule a été conçu.

Certains auteurs ont défini des "niveaux" d'autonomie d'un engin inhabité. Nous citerons, à titre d'exemple, le classement de l'Office de la Recherche Navale Américaine (ONR) (tableau 1.2). Si nous nous restreignons à ce classement, nous pouvons considérer que nous nous intéressons exclusivement à des véhicules autonomes disposant déjà d'une *autonomie totale*. Le but est de faire

TABLEAU 1.2 – Classement de l'ONR, traduit par [Chanthery, 2005]

Niveau	Nom	Description.
0	Engin télé-opéré	Toute l'activité du système est le résultat d'entrées/commandes humaines. Le système ne prend aucune décision.
1	Engin humainement assisté	Le système peut agir en parallèle avec des entrées opérateur, il améliore la réussite de l'opérateur sur l'activité qu'il effectue mais n'a pas la capacité d'agir sans ordre de l'opérateur. Par exemple, une voiture à boîte automatique.
2	Délégation du contrôle	Le système peut avoir un contrôle de son activité sur la base de la délégation. Par exemple un pilote automatique pouvant être activé ou non par un pilote.
3	Supervision humaine	Le système peut effectuer un large panel d'activités avec la permission ou le contrôle d'un humain. Le système renvoie suffisamment d'informations sur ses opérations internes et son comportement pour qu'un humain puisse le superviser et le rediriger si nécessaire. Le système n'a pas la capacité de prendre des initiatives sur des comportements qui ne seraient pas en rapport avec les tâches qui lui sont assignées au départ.
4	Initiative partagée	L'humain et le système peuvent tous deux prendre des initiatives basées sur les informations disponibles. Le système peut coordonner son comportement avec celui de l'humain tant explicitement qu'implicitement. Des règles régissent le partage de l'autorité.
5	Autonomie totale	Le système n'a besoin d'aucune intervention humaine pour effectuer sa mission et s'adapte à son environnement.

évoluer des engins de ce type vers une véritable autonomie d'équipe, où l'ensemble de l'équipe ne dépendrait pas directement d'un opérateur humain.

La réalité opérationnelle est cependant plus complexe. Certaines tâches considérées comme critiques imposent généralement l'intervention d'un opérateur humain dans la boucle (responsabilité particulière engagée, ou tâche de perception encore trop complexe pour être automatisée par exemple...) L'autonomie n'est pas un terme aisé à définir et certains auteurs [Mercier, 2010] tentent de s'abstraire de la notion des niveaux d'autonomie fréquemment utilisée dans la littérature. Nous supposerons tout de même que les engins sont *a priori* capables d'assurer seuls la mission qui leur est confiée. L'objectif de nos travaux est donc de rendre l'équipe capable d'accomplir en totale autonomie une mission donnée, et de lui procurer une autonomie décisionnelle suffisante pour lui permettre de réagir seule face à l'occurrence d'aléas.

1.3 Véhicules autonomes et architectures monovéhicules

Nombre de robots font actuellement preuve d'une autonomie décisionnelle suffisante pour accomplir des tâches de plus en plus complexes, sans intervention humaine. Ils sont employés dans de nombreux domaines, pour des applications parfois très différentes. Cette autonomie décisionnelle repose sur des architectures logicielles embarquées spécifiques, qui gèrent l'exécution des tâches pour lesquelles les véhicules ont été conçus, et ce, en conditions réelles. En voici quelques exemples :

- le RQ-8A Fire Scout [2] a démontré sa capacité à décoller et apponter sur le pont d'un navire ;
- les drones ReSSAC - figure 1.1(a) [Teichteil-Königsbuch et al., 2011] - ou Raven [3] - figure 1.2(a) peuvent décoller, suivre une route constituée de waypoints et atterrir en un lieu précis ;
- l'AUV Daurade, figure 1.1(b), [Barbier et al., 2006] peut réaliser un balayage complet d'une zone pour effectuer des relevés hydrographiques et océanographiques ;
- le drone ReSSAC peut également suivre une cible (*tracking*) [Watanabe et al., 2010] sans intervention humaine.

On peut également noter que certains satellites sont plutôt prometteurs en terme d'autonomie décisionnelle (*Earth Observing One* [4], figure 1.1(d), [Chien et al., 2004]), que des sondes spatiales ont également montré une autonomie remarquable, tout comme les rovers d'exploration (Dala, figure 1.1(c), [Basu et al., 2008]). Nombre de véhicules aériens sans pilote (ou UAV) déjà commercialisés et déployés sur différents théâtres d'opération sont désormais capables d'effectuer des missions en autonomie complète, même s'ils semblent rester majoritairement télé-opérés (figure 1.2).

Dans une optique de déploiement d'une équipe de véhicules, le nombre et l'hétérogénéité des architectures rendent les développements complexes. Cette hétérogénéité est renforcée par le fait que leur déploiement peut être progressif et s'étaler sur plusieurs dizaines d'années comme c'est souvent le cas dans de domaine spatial par exemple.

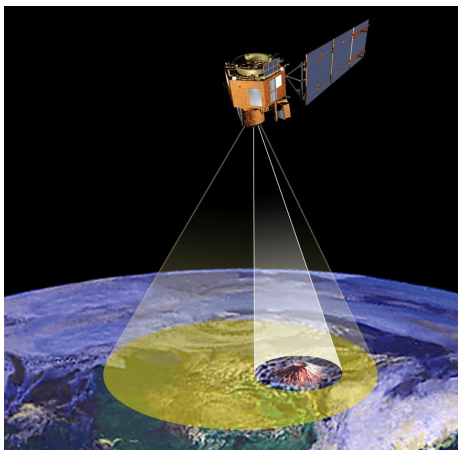
Le tableau 1.3, traduit du site `wiki.robot-standards.org` [5], analyse brièvement quelques architectures développées, en soulignant leurs différences. Ces architectures sont à l'origine conçues dans une optique monorobot. Cependant, il est à noter que certaines de ces architectures ont déjà été utilisées dans des contextes multiagents. Nous en reparlerons dans le chapitre 2.

2. `http://www.navy.mil/search/display.asp?story_id=22038` - Septembre 2012
3. `http://www.globalsecurity.org/intell/systems/raven.htm` - Septembre 2012
4. `http://eo1.gsfc.nasa.gov/` - Septembre 2012
5. `http://wiki.robot-standards.org/index.php/Comparison_and_Evaluation_of_Middleware_and_Architecture` - Mai 2011

(a) Le drone à voilure tournante RMax/ReSSAC de l'Onera.

(b) L'AUV Daurade de la DGA.

(c) Le Dala du LAAS.



(d) Earth Observing One, de la NASA.

FIGURE 1.1 – Exemples de véhicules autonomes.

(a) Le Raven, d'AeroVironment, capable d'accomplir une mission en complète autonomie, dont la navigation est basée sur des points GPS.

(b) Le Harfang, d'EADS. En cas de perte de communication avec les opérateurs, il retourne à son point de départ.

(c) Le MQ-9 Reaper de General Atomics, capable de suivre un itinéraire préprogrammé.

(d) Le RQ-4 Global Hawk de Northrop Grumman dont la navigation peut être autonome, reposant sur GPS.

(e) Le Drone de Reconnaissance Au Contact (DRAC) d'EADS capable de vols et d'atterrissages en complète autonomie.

FIGURE 1.2 – Exemples d'UAV opérationnels déjà dotés d'une certaine autonomie décisionnelle.

	Architecture			Notions de dépendances	
	Modèle du système	Modèle de contrôle	Modèle de décomposition	Tolérance aux fautes	Robustesse
LAAS	3 couches, données décentralisées, client-serveur	centralisé, événementiel	fonctionnel orienté composants	R2C, génération de code automatisé (GenOM,ExoGen), diagnostic de composants	couple planificateur/superviseur contrôle d'exécution (OpenPRS)
CLARAty	2 couches, données décentralisées, client-serveur	centralisé, événementiel	fonctionnel orienté composants	vérification de ressources, estimation d'états, classes de simulation	planificateur/scheduler contrôle d'exécution (TDL) estimation d'état
ORCCAD	similaire à une approche multicouches	applications développées pour être à la fois centralisé et événementiel	considéré comme orienté composants	processus V&V, analyse hardware, génération de code, ressource mapping	récupération de 3 types d'exception
3T	3 couches, données décentralisées, TCP/IP	centralisé, événementiel	fonctionnel orienté composants sur les capacités	gestion de capacités, génération de code automatisé, processus V&V	planificateur, moteur d'exécution, scheduler, suivi d'anomalies
RAX /Livington	3 couches, données décentralisées, système de BUS	centralisé, événementiel	peu clair, mais semble fonctionnel, orienté pipeline	estimation d'états, diagnostics, reconfiguration	planificateur, moteur d'exécution, scheduler
Player	peu de contraintes, données décentralisées, client-serveur	non applicable	composants individuels	mécanismes de blocage de thread	non explicite
ORCA	peu de contraintes, données décentralisées, P2P	non applicable	orienté composants	reconfiguration dynamique de certains composants	non explicite
CoolBot	peu de contraintes, données décentralisées, P2P, client-serveur	centralisé	orienté composants	récupération de différentes exceptions, adaptation	non explicite
Miro	3 couches, données décentralisées, client-serveur	événementiel	orienté composants	détection des erreurs pour les services	BAP framework, reconfigurations dynamiques, gardes logiques
IDEA	peu de contraintes, données décentralisées, P2P	centralisé, événementiel	orienté composants (si on considère un agent comme un composant)	supervision des variables, basée timelines, machine virtuelle pour chaque agent pour simplifier les résolutions de conflits	différents niveaux de planification, gestion des durées d'exécution

TABLEAU 1.3 – Analyse de quelques architectures monorobots, issue de `http://wiki.robot-standards.org`, section middleware.

Plus généralement, les missions robotiques futures pourraient de plus en plus s'appuyer sur du multivéhicule. D'après Parker [Parker, 2008], 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 requiert 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. Au contraire, l'utilisation d'équipes multirobots peut potentiellement [Parker, 2000] :

- accroître la robustesse et la tolérance aux fautes par redondance ;
- diminuer le temps nécessaire à l'accomplissement de la mission via le parallélisme ;
- décroître la complexité de conception individuelle d'un robot grâce à l'hétérogénéité multivéhicule ;
- élargir l'éventail des applications possibles aux missions intrinsèquement distribuées.

Cependant, les systèmes multirobots sont plus complexes à mettre en œuvre et de nouvelles difficultés apparaissent. Les robots doivent interagir et donc communiquer, peuvent se gêner, induisent des problèmes de partage en cas de ressources limitées. La complexité de la phase préparatoire de la mission est accrue de par le nombre de véhicules qui composent l'équipe. Des problèmes de cohérence des données sont à prévoir. Enfin, la décomposition et l'allocation des objectifs au sein de l'équipe est une étape supplémentaire de résolution à effectuer.

L'interaction entre véhicules autonomes est en elle-même un point clé. "On peut raisonnablement considérer que les collaborations [et la coopération] entre robots dans les années à venir seront de plus en plus nombreuses, dans l'intérêt de la recherche" [Visentin, 2007].

1.4 Contexte opérationnel : le PEA Action

Ce travail se déroule dans le cadre du Programme d'Etudes Amont Action, ou PEA Action. L'objectif du PEA Action est d'étudier les moyens disponibles et de préparer les technologies futures en vue de renforcer les performances de la fonction localisation dans un réseau d'entités hétérogènes constituées de vecteurs autonomes. Les technologies proposées seront évaluées dans le cadre de démonstrations scientifiques s'appuyant sur un ensemble de scénarios représentatifs consolidés avec les opérationnels [6].

Ces scénarios, dont nous détaillons dans la suite ceux qui servent de support expérimental à ce mémoire, illustrent le contexte dans lequel nous nous sommes placés. Les véhicules de l'équipe y sont tous supposés déjà autonomes individuellement. Ils sont hétérogènes, tant sur le plan de leur milieu opérationnel (aérien, terrestre, maritime de surface, sous-marin) qu'au niveau de leur architecture embarquée. L'environnement des scénarios est dynamique, et les communications y sont incertaines, voire impossibles lors du déroulement de certaines phases (par exemple, lorsque un véhicule sous-marin est en plongée, il ne peut pas communiquer).

Le scénario I (figure 1.4) est un scénario aéroterrestre. Deux véhicules, un terrestre (AGV) et un aérien (AAV) à voilure tournante, explorent une zone rurbaine à la recherche d'un intrus. Celui-ci doit être localisé, puis chassé de la zone (l'hypothèse d'un comportement évasif de l'intrus est faite).

Le scénario II (figure 1.5) est un scénario aéromaritime. L'objectif pour l'équipe constituée d'un AAV et d'un sous-marin (AUV) est de "blanchir" un chenal, c'est-à-dire de garantir qu'une zone ne contient ni bâtiment de surface, ni mine, pour permettre par exemple à un sous-marin de quitter son port en toute sécurité.

6. Pour plus de détails, voir `http://action.onera.fr/`.

Le scénario VI (figure 1.6) est encore un scénario aéroterrestre (les quatre scénarios aéroterrestres sont de complexité croissante). Il s'agit cette fois de gérer une flottille de véhicules aériens et terrestres autonomes qui vont devoir contrôler simultanément plusieurs zones rurbaines. Le terrain à contrôler est composé de bâtiments, de végétation plus ou moins traversable et de chemins. Si un intrus y est localisé, il est chassé par des véhicules autonomes de l'équipe hors de la zone.

Un AGV est capable de détecter une cible et de trouver seul son chemin entre deux points, s'il dispose d'une carte de l'environnement et qu'un trajet y est possible pour lui. Il peut néanmoins se perdre (erreur de localisation, dérive des centrales inertielles...) ou rester dans un état "bloqué" lorsqu'un obstacle lui barre la route qu'il avait et qu'il ne trouve pas de solution pour l'éviter.

Un AAV est capable d'effectuer des déplacements aériens de waypoint en waypoint. Il peut effectuer des balayages de zones de plusieurs façon différentes. Il dispose de moyens pour détecter et localiser avec précision des entités telles que des AGV, des bâtiments de surface, des AUV en surface, des nappes de polluant, ou des intrus. Il peut construire une carte de terrain sur une zone définie. Enfin, il est capable de suivre une cible. Pour éviter les risques de collision, l'AAV vol au-dessus des obstacles possibles (bâtiments, arbres...)

Un AUV est capable de balayer une zone à la recherche de mine ou d'épave en complète autonomie. Il peut aller d'un waypoint à un autre, plonger ou faire surface. Il ne peut rester immobile mais peut réaliser des hypodromes.

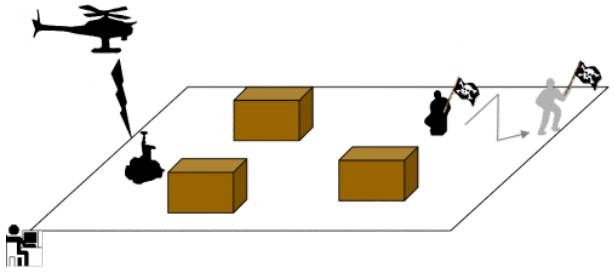


FIGURE 1.4 – Scénario I.

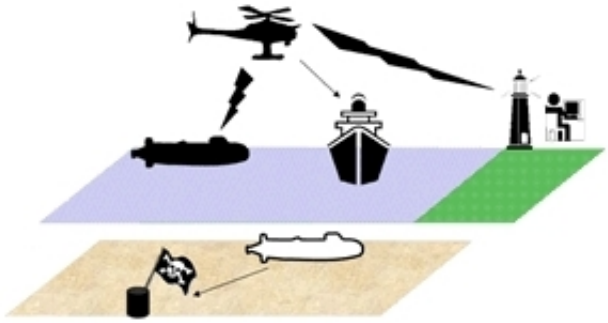


FIGURE 1.5 – Scénario II.

La nature même de ces missions opérationnelles explique l'utilisation de différents véhicules dotés de fonctionnalités complémentaires. L'environnement est dynamique, ce qui implique la nécessité de prendre en compte l'occurrence d'aléas durant le déroulement d'une mission.

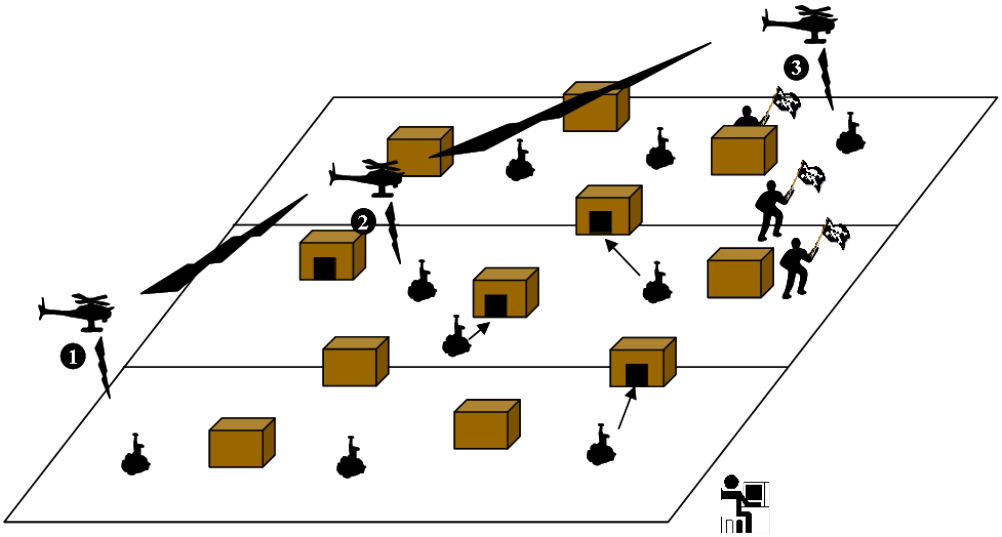


FIGURE 1.6 – Scénario VI.

1.5 Problématique

Notre travail consiste ainsi à concevoir et à mettre en œuvre une architecture décisionnelle capable de superviser l'exécution d'une mission par une équipe de véhicules autonomes hétérogènes, avec comme application le cadre des scénarios du PEA Action.

Le superviseur, objet de cette thèse, doit être capable, non seulement de contrôler l'exécution d'un plan prédéfini pour des véhicules hétérogènes de par leur nature et leur architecture logicielle, mais aussi d'intégrer des mécanismes de gestion des aléas, puisque l'environnement dans lequel les véhicules autonomes hétérogènes sont amenés à évoluer est dynamique. Les véhicules autonomes actuellement opérationnels pourraient-ils être directement intégrés dans des équipes ? Sinon, existe-t-il déjà des **architectures décisionnelles multivéhicules** capables de superviser cette exécution ? Une approche centralisée est-elle possible, ou un système distribué est-il mieux adapté ?

La communication entre véhicules est également une contrainte à ne pas négliger. Dans un monde réel, les véhicules peuvent rencontrer des situations dans lesquelles ils ne peuvent plus, temporairement, communiquer avec leurs coéquipiers. Ils peuvent être également restreints à ne pouvoir communiquer qu'avec un nombre limité d'autres véhicules, ou avec un débit réduit. Le superviseur doit prendre en compte ces facteurs. Quel comportement doit adopter un véhicule qui se retrouve isolé en terme de communication ? Doit-il chercher à la rétablir à tout prix, ou peut-il s'en passer temporairement ? La question concernant la détection de la perte d'un véhicule peut également s'avérer intéressante : comment distinguer si le véhicule ne communiquant pas est simplement dans une zone où la communication n'est pas disponible, est victime d'une panne de son système de communication, ou est hors service ?

En robotique d'exploration spatiale, il n'est pas aisé de coordonner finement un ensemble de véhicules depuis un centre d'opération basé sur Terre, principalement à cause des contraintes de communication. Une majeure partie de cette coordination doit donc avoir lieu sur place, menée par les agents robotiques eux-mêmes. En conséquence, les véhicules doivent non seulement être dotés de moyens pour communiquer entre eux, mais également d'une autonomie décisionnelle suffisante pour interagir sans intervention d'opérateurs distants.

Pour de telles missions, impliquant diverses tâches et véhicules, structurer à la fois les actions

à effectuer et l'équipe en charge de leur exécution semble être une solution naturelle. Le cadre opérationnel du PEA Action nous fournit un **plan de mission** qu'il serait intéressant d'exploiter, ne serait-ce que pour fournir aux véhicules les objectifs à accomplir. Quelle serait la forme de plan la plus adaptée pour être exploitable par la supervision ? Quels outils serviront à sa génération concrète et ceux-ci pourront-ils être employés en-ligne ?

Les véhicules doivent-ils être organisés suivant une **structure d'équipe** particulière ? La communication entre véhicules est-elle un facteur contraignant quant à la forme organisationnelle que prendrait l'équipe ?

Enfin, l'aspect dynamique de l'environnement dans lequel évoluent les véhicules ne peut être ignoré. Il faut partir d'un postulat certain : dans un cadre robotique opérationnel, une mission a une probabilité très faible de se dérouler exactement comme prévu initialement. Il est donc impératif de prendre en compte l'occurrence d'aléas. Lorsqu'un aléa survient, comment le système en est-il informé et comment doit-il réagir ? Dans quelle mesure est-il possible de limiter son impact sur le déroulement de la mission ? L'ensemble du plan doit-il être remis en cause ou des modifications locales du plan - des **réparations** - sont-elles envisageables ? L'ensemble des véhicules est-il concerné par cet aléa ?

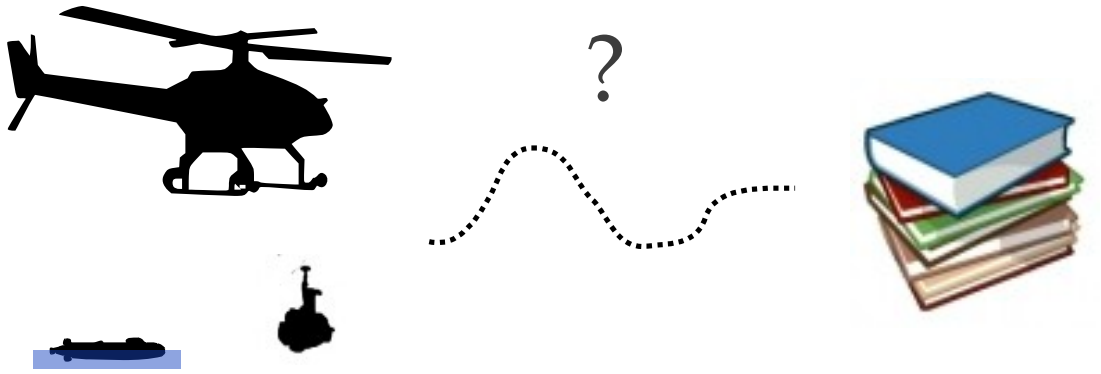
Notre problèmatique est donc la "supervision de mission pour une équipe de véhicules autonomes hétérogènes" :

- qui disposent déjà d'une architecture décisionnelle individuelle,
- basée sur l'existence d'un plan de mission,
- en prenant en compte les contraintes de communication d'un environnement dynamique opérationnel,
- robuste face à l'apparition d'aléas pendant le déroulement de la phase d'exécution,
- pouvant exploiter un ou plusieurs planificateurs,
- avec comme application des scénarios réalistes,
- et pouvant se dérouler potentiellement sans intervention extérieure à l'équipe (rôle de l'opérateur humain limité à l'initialisation).

Pour tenter de trouver des solutions face à ces interrogations, nous étudions dans un premier temps les travaux existants qui s'y réfèrent.

CHAPITRE 2

Etat de l'Art



Pour répondre à notre problématique, nous avons analysé l'état de l'art en nous concentrant sur les architectures logicielles multivéhicules actuelles, les stratégies de réponse à la présence d'imprévus lors du déroulement de la mission d'équipe, ainsi que sur la manière dont les incertitudes sur les communications sont prises en compte.

Sommaire

2.1 Introduction

Dans un système multivéhicule autonome, les véhicules vont devoir non seulement gérer leur autonomie individuelle mais également le fait qu'ils ne sont plus seuls à être en charge de la mission : des besoins supplémentaires d'une coordination régulière apparaissent tout au long de la mission.

Comme pour le monovéhicule (cf. section 1.2 p.6), il n'est une fois de plus pas aisé de "classifier" le type d'autonomie de tels systèmes. A titre indicatif, le tableau 2.1 propose un classement d'un système multivéhicule suivant les "niveaux" d'autonomie des véhicules.

TABLEAU 2.1 – "Niveau" d'autonomie d'équipe, source OSD UAV Roadmap 2002-2027, traduction [Chanthery, 2005]

1	Guidage à distance
2	Diagnostic d'état en temps réel
3	Adaptation aux pannes et aux conditions de vol
4	Replanification embarquée
5	Coordination de groupe
6	Replanification tactique de groupe
7	Buts tactiques de groupe
8	Contrôle distribué
9	Buts stratégiques de groupe
10	Groupe totalement autonome

Notre travail se place au "niveau" d'autonomie le plus élevé. Un certain nombre d'architectures multivéhicules ont été proposées, cette section en présente les principales. Comme nous faisons référence à des travaux issus de la communauté des SMA (Systèmes Multi Agents) et pour éviter toute confusion, nous utiliserons la définition la plus générale d'un **agent** : "une entité physique ou informatique plongée dans un environnement, pouvant percevoir tout ou partie de cet environnement grâce à des capteurs fournissant des entrées, et le modifier grâce à des effecteurs délivrant des sorties" [Russel and Norvig, 1995]. Mais nous privilégierons l'utilisation des termes **robots** et **véhicules autonomes** de notre contexte lorsqu'ils sont applicables.

Nous presentons dans un premier temps un état de l'art sur différentes architectures multivéhicules de la section 2.2 à la section 2.6. Nous prêterons plus particulièrement attention à la prise en considération des communications, à la gestion des aléas, à la généricité des architectures (à savoir, si les architectures monovéhicules initiales sont conservées) et enfin quelles en ont été les mises en œuvre concrètes. Puis, après un premier bilan section 2.7 (p.39) la section 2.8 (p.42) résume les travaux multivéhicules spécifiques à la prise en compte des contraintes portées sur les communications.

2.2 Architectures décisionnelles appliquées au multirobot

2.2.1 L'architecture ALLIANCE

L'architecture ALLIANCE [Parker, 1998] 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 co-

hérente à une évolution inattendue de l'environnement" au travers d'un ensemble d'attitudes prédéfinies (Behavior Set). Cette architecture a été conçue pour permettre de traiter des problèmes du monde réel, avec des équipes de robots (possiblement hétérogènes) à la fois tolérantes aux fautes et adaptables. En effet, ces robots ne sont pas infaillibles, sont dotés de capteurs bruités et d'actionneurs imparfaits. Le but est de les amener à pouvoir coopérer.

La réactivité des robots est induite par des *motivations* : chaque robot modélise, *via* l'observation de performances et de statistiques sur la qualité de réalisation (par exemple : durée de réalisation de la tâche), sa *motivation* et la *motivation* des autres robots de l'équipe à réaliser les tâches du système. Ces modèles sont utilisés pour l'affectation des futures tâches à réaliser : un robot en bon état de marche devient *impatient* pour réaliser une tâche dont il constate qu'elle est en échec chez un coéquipier ; un robot défaillant devient *consentant* pour abandonner une tâche qu'il constate ne pas ou ne plus pouvoir faire. Cette approche permet une réactivité importante aux aléas dans l'équipe ou aux modifications dans la mission ou l'environnement. En fait, 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 (issues de capteurs, de messages, d'instructions d'inhibitions fournies en interne), et, en fonction de ces dernières, sélectionnent les actions adaptées à la situation. Sur la figure 2.1, les interrupteurs schématisés indiquent si les sorties des *Behavior Set*, qui correspondent chacun à un module de *Motivational Behavior*, sont transmises ou non aux actuateurs des robots. Les différents *layer* représentent des ensembles de comportements toujours actifs tel que l'évitement d'obstacle. Finalement, la sommation des informations issues des liens actifs représentent le comportement que le robot va adopter.

Le laboratoire ORNL (Oak Ridge National Laboratory) a démontré avec succès la faisabilité de l'architecture en implémentant une équipe de robots nettoyeurs [Parker, 1998], de robots poussant des boites conjointement [Kannan and Parker, 2007], ainsi que divers scénarios qui impliquent des robots hétérogènes [Parker, 2000].

Ce type d'approche est qualifiée de *réactive*, n'ayant ni planification de haut niveau, ni allocation de tâches. Des auteurs [Hartley and Pipitone, 1991] constatent de grosses limitations en terme de passage à l'échelle vers des problèmes "modérément complexes" et en terme de modularité pour de telles approches, notamment au sujet des travaux de Brooks [Brooks, 1990] basés sur des approches réactives.

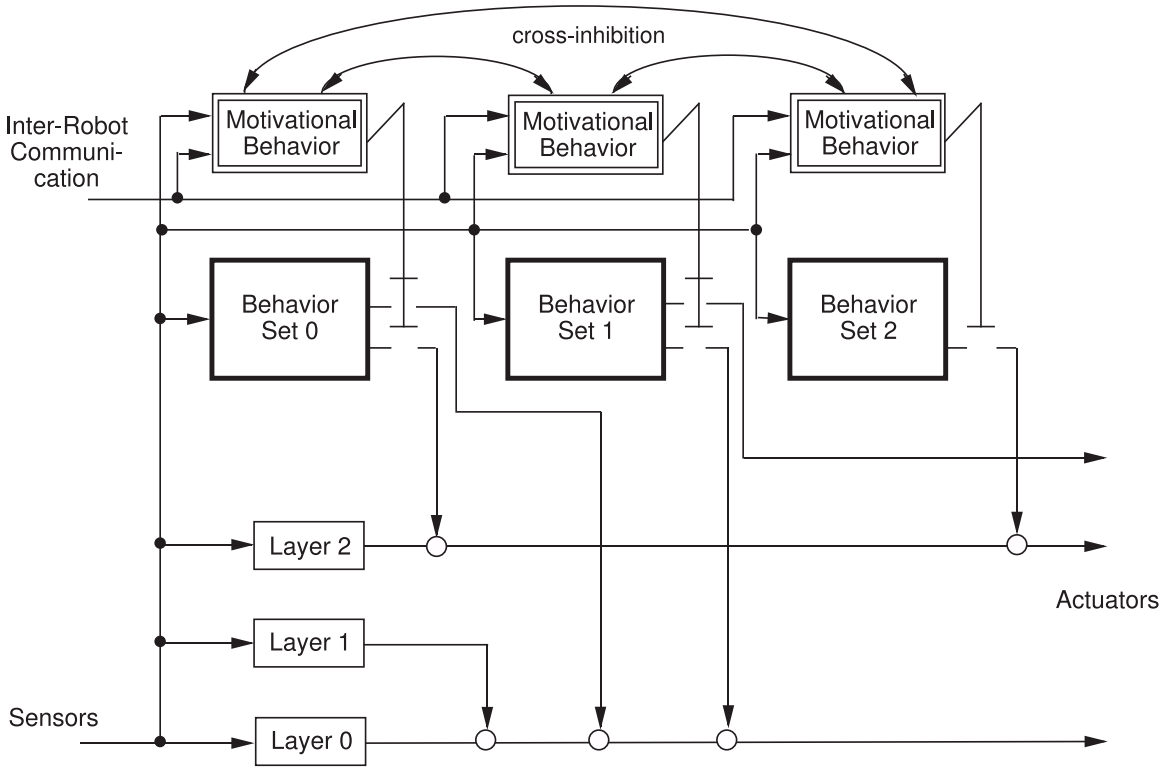


FIGURE 2.1 – L'architecture ALLIANCE, pour un robot (extrait de [Parker, 1998]).

2.2.2 Architecture du projet BEAR

BErkeley AeRobot (BEAR) [1] 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. Ils utilisent pour cela une architecture qui répartit le contrôle de chaque agent sur différentes couches abstraites et hiérarchisées. La figure 2.2 en montre un exemple, les flèches désignant les relations hiérarchiques entre couches abstraites. Mais ils supposent que chaque agent connaît l'état, la position et les observations de tous les autres agents, ce qui n'est pas envisageable pour une implémentation réelle [Vidal et al., 2002].

Ces derniers proposent donc, pour y remédier, une architecture distribuée, hybride et hiérarchique basée sur BEAR. Ils insistent sur l'autonomie de chaque agent et le fait d'avoir une coordination au sein de l'équipe. Ils tiennent compte du fait que l'environnement est dynamique, les agents hétérogènes, les capteurs bruités et que des pannes sont possibles sur les robots. 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. L'architecture est en outre dotée d'une tolérance aux fautes à tous niveaux d'abstraction (du capteur élémentaire au robot manquant).

Malgré l'aspect modulaire, les connexions entre les constituants sont nombreuses, ce qui implique, dans la mise en œuvre concrète du système, un processus d'adaptation à chaque type d'agent complexe. Par exemple, l'architecture gère et prend en charge les communications à la fois entre les modules bas niveau (GPS, altimètre...) et haut niveau (planificateurs, fusion de données...)

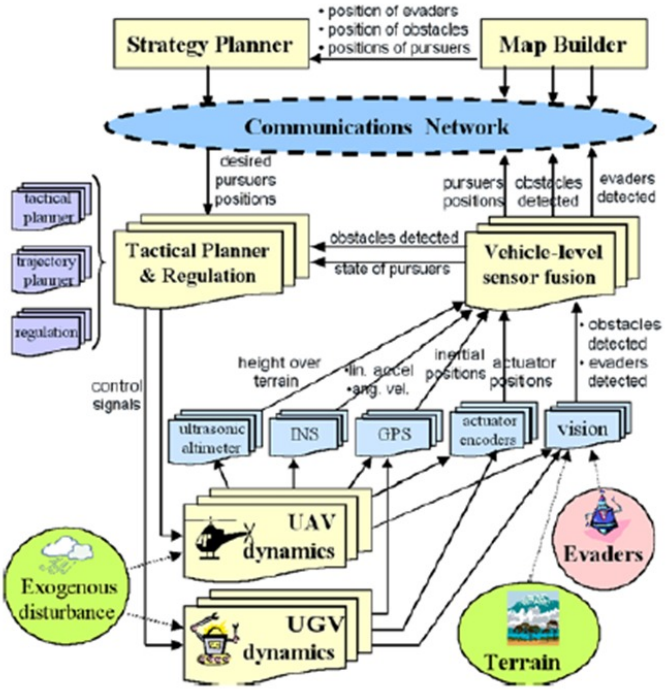


FIGURE 2.2 – L'architecture BEAR : coopération de deux véhicules.

1. http://robotics.eecs.berkeley.edu/bear

2.2.3 Architecture basée sur l'architecture LAAS

Dans sa thèse, Gancet [Gancet, 2005] propose une architecture pour la planification, la supervision et la coopération d'UAV (Unmanned Aerial Vehicles) basée sur l'architecture monovéhicule LAAS [Alami et al., 1998], associée au planificateur SHOP2 [Nau et al., 2003]. L'autonomie des robots y est classifiée en degrés et l'humain est maintenu aux côtés de l'architecture. Cela permet notamment de proposer un large spectre de configuration d'autonomie décisionnelle différente, allant d'un extrême (les UAV dépendent totalement d'une station centrale pour leurs actions) à l'autre (les UAV s'auto-organisent pour accomplir leurs buts). La figure 2.3 schématise l'architecture proposée. L'Exécutif multi-degré est chargé d'exécuter les plans. Il est basé sur Open PRS (Open Procedural Reasoning System) [Ingrand et al., 1992] et s'interface avec les composantes fonctionnelles propriétaires (CPE de la figure). Un couple formé par le planificateur symbolique et les raffineurs spécialisés permet à la planification HTN (Hierarchical Task Network [Erol et al., 1994]) d'exploiter des modèles géométriques de l'environnement et des engins. Le gestionnaire d'interactions est au cœur de la coopération entre véhicules, notamment au niveau des tâches jointes qu'ils doivent exécuter. Le superviseur de la couche délibérative (CD) organise les activités et rythme les flux de données dans la couche délibérative. Les degrés d'autonomie auxquels se situent les flux sont précisés dans la légende.

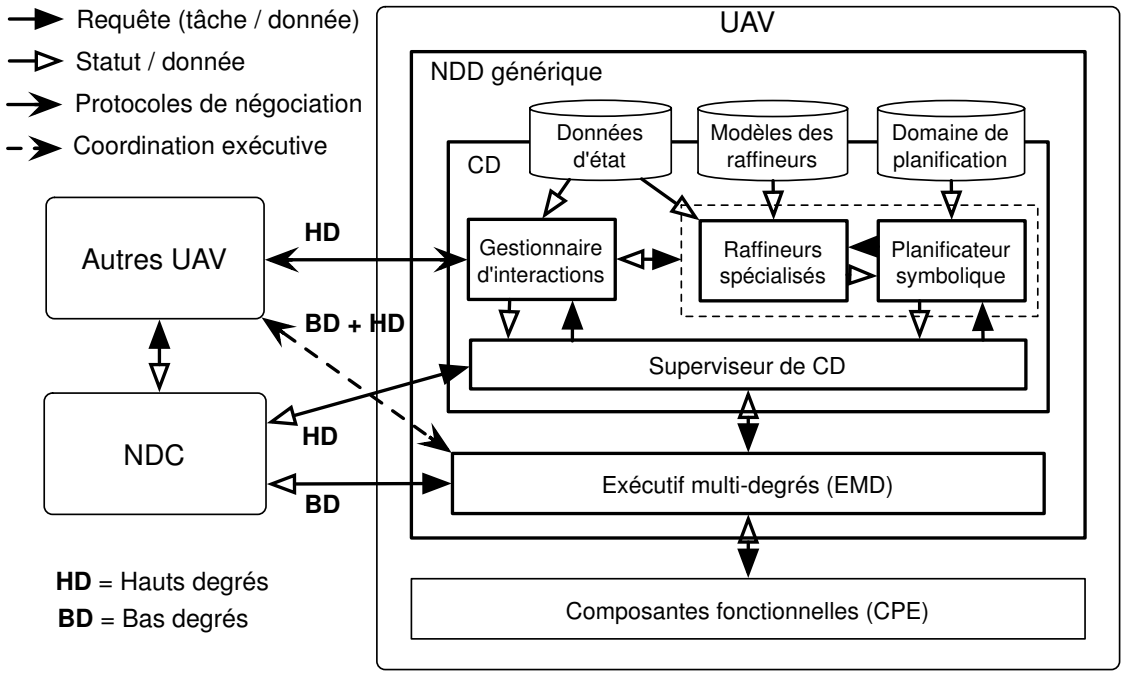


FIGURE 2.3 – Une architecture pour la planification, la supervision et la coopération d'UAV, extrait de [Gancet, 2005] (NDD : Nœud Décisionnel Distribué, NDC : Nœud Décisionnel Centralisé).

2.2.4 Architecture T-Rex

L'architecture T-Rex [McGann et al., 2008] développée au MBARI, permet d'affiner, de décomposer et d'ordonnancer des tâches, tout en prenant en considération leur future exécution. T-Rex est composée de plusieurs réacteurs (figure 2.4). Chaque réacteur contient une base de données (*Database*) où le modèle temporel des tâches est défini, un planificateur (*Planner*) qui utilise un horizon de planification, un synchroniseur (*Synchronizer*) et un répartiteur (*Dispatcher*). Le planificateur utilisé par défaut dans T-Rex est le planificateur Europa [Bresina et al., 2005]. On peut ensuite définir un agent contenant plusieurs réacteurs travaillant avec des latences, des fréquences, et des horizons temporels différents sur des sous-parties du problème général. Même si la version originale de T-Rex ne permet pas l'échange de plans ou de buts entre les robots, cette architecture a tout de même déjà été employée dans un contexte multirobot. Ainsi, dans celle proposée par [Belbachir et al., 2012], chaque robot est un agent T-Rex avec plusieurs réacteurs. L'un de ces réacteurs (CoopReactor) est en charge de communication et d'échange d'informations avec les autres agents. En l'état, ce réacteur est essentiellement utilisé pour partager des données de perception entre les agents, mais on peut imaginer des extensions où ce serait des objectifs ou des plans qui seraient partagés. Il faut toutefois noter qu'ici, la nature même de l'application, l'exploration sous-marine, limite considérablement les possibilités de communication et donc ne permet pas une bande passante ni une réactivité compatible avec des protocoles d'allocation ou de ré-allocation de tâches dont nous parlerons à la section 2.5 (p.34).

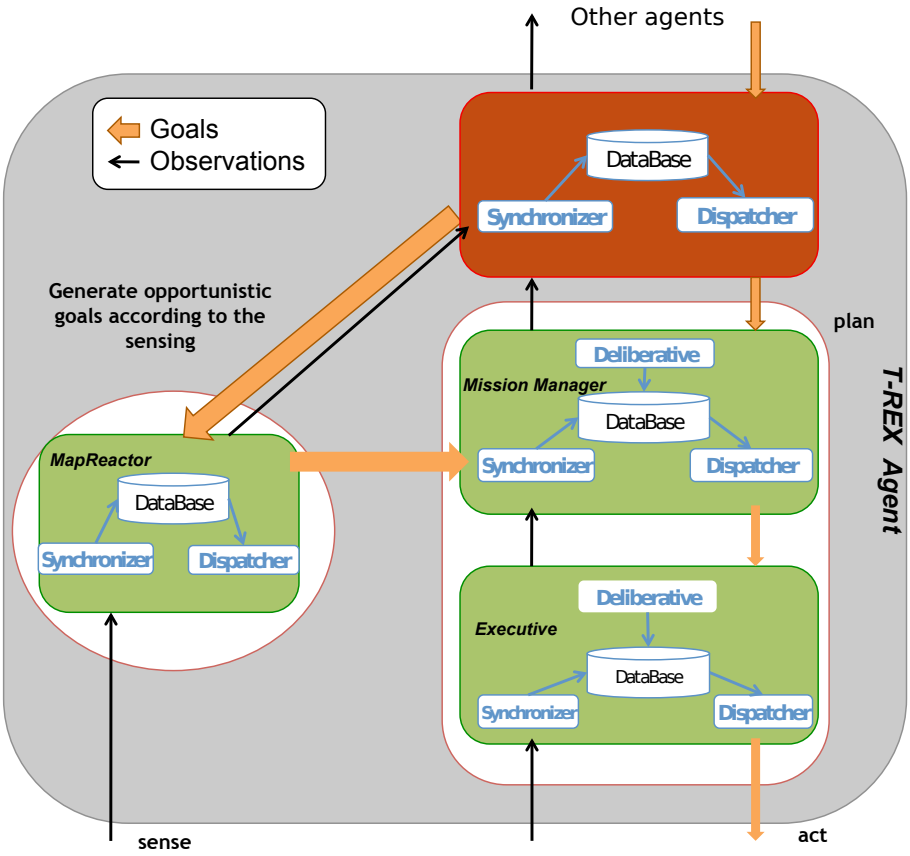


FIGURE 2.4 – L'architecture T-Rex, pour un véhicule.

2.2.5 Architecture basée sur BIIMAPS

Dans le contexte d'une équipe de véhicules autonomes sous-marins (AUV) utilisée dans un scénario de guerre des mines, Sotzing [Sotzing et al., 2008] note que ce genre de mission est actuellement basé sur des scripts de type "IF-THEN-ELSE", et qu'il est donc souvent difficile de prendre en compte des événements non prévus dans le plan pour la supervision de l'équipe de véhicules autonomes. Cette difficulté s'amplifie lorsqu'une coordination est requise dans la mission. Il propose donc une architecture dotée d'un tableau de bord BIIMAPS (*BlackBoard Integrated Implicit Multi-Agent Planning Strategy*) (figure 2.5) qui supervise l'exécution de la mission et facilite la coordination multivéhicule. En cas d'aléas, le *tableau de bord* se replace au dernier état valide et se charge de la coordination des informations entre les agents. Dans un plan BIIMAPS, chaque but contient une condition qui permet de déterminer son accomplissement. La partie planification est basée sur les HTN [Erol et al., 1994]. L'intérêt principal du tableau de bord est de permettre à des buts d'être accomplis en parallèle, et surtout, de corriger d'éventuels conflits qui pourraient apparaître après une période sans communication. Le tableau de bord gère notamment la mise à jour des informations connues par les autres véhicules.

L'architecture a été implémentée et testée. Les auteurs ont notamment utilisé le simulateur DELPHIS [Sotzing, 2007], chargé de coordonner des véhicules dans des environnements aux conditions de communication restreinte, tout en gardant les besoins de replanification de la mission au strict minimum. Chaque véhicule est doté d'une copie complète du plan BIIMAPS. Même si tous les véhicules dans ces travaux sont de même type (des AUV), ils indiquent que leur système est aisément implémentable sur d'autres types de véhicules.

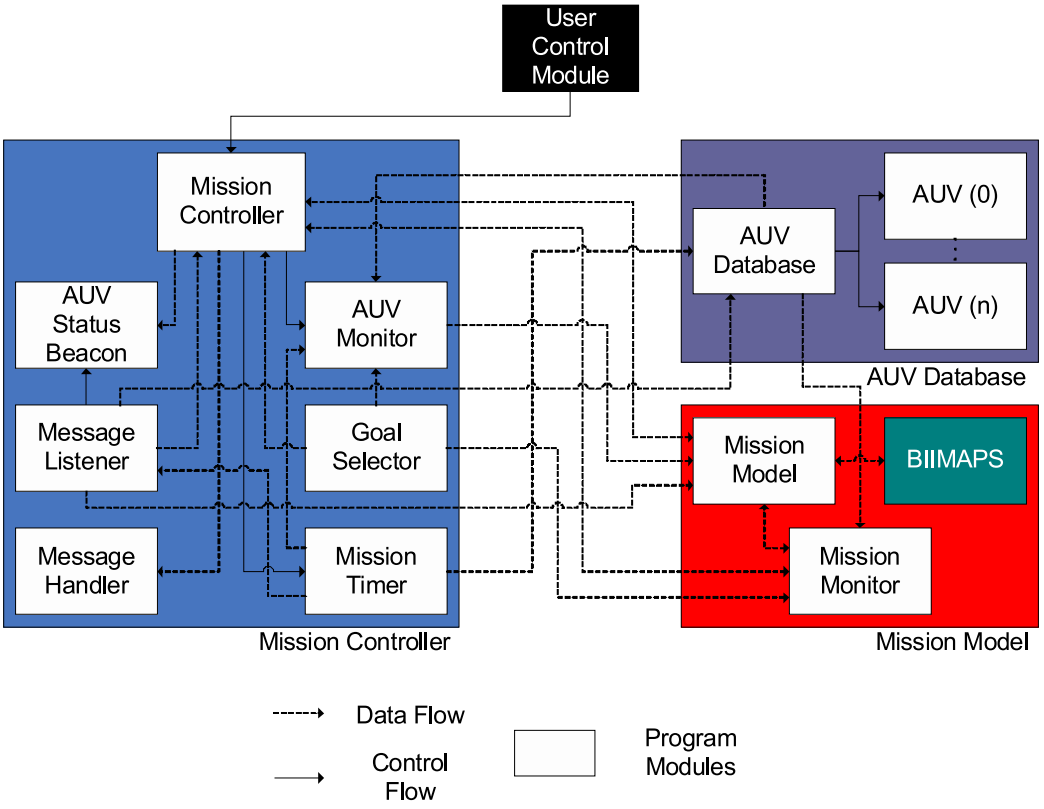


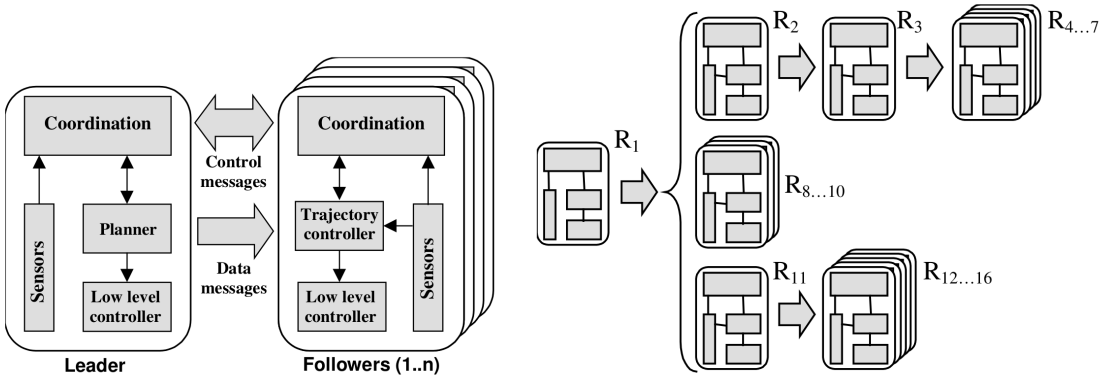
FIGURE 2.5 – L'architecture basée sur BIIMAPS sur chaque AUV, extrait de [Sotzing et al., 2008].

2.2.6 Architecture exploitant la logique temporelle

[Doherty et al., 2009] travaillent sur des scénarios qui impliquent des équipes de véhicules aériens (AAV) 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. Un moteur d'exécution est d'ailleurs chargé de les contrôler. L'objectif est d'utiliser directement des informations collectées durant le déroulement de la mission pour vérifier la validité du plan fourni, et replanifier lorsque c'est nécessaire. Pour cela, le middleware *DyKnow* [Heintz and Doherty, 2004] est chargé de recueillir ces informations sur l'état courant du véhicule. Leur architecture se décompose ainsi en trois couches : une couche de contrôle bas niveau, une couche réactive et une couche délibérative. Ils utilisent des machines à état hiérarchisées pour le cœur du contrôleur. Les échanges entre différents services sont assurés via le protocole CORBA. L'interaction entre les véhicules semble implicite, ou la communication totale.

2.3 Architectures fortement couplées

Une architecture fortement couplée est une architecture dans laquelle tous les acteurs ne sont pas seulement liés entre eux, mais ils sont aussi dépendants les uns des autres. Dans le cadre de mission bien spécifiques, il existe aussi des architectures fortement couplées [Chaimowicz et al., 2001,Parker et al., 2004] pour de la coopération de robots hétérogènes. Dans ces missions, plusieurs classes de robots se distinguent. Certaines sont riches en termes de capteurs, d'actionneurs et de systèmes décisionnels embarqués, les *leaders*, d'autres, les *followers*, n'ont que des capacités limitées (par exemple, ni évitement d'obstacle, ni localisation...), et dépendent donc des leaders précédents pour recevoir des instructions et contribuer à la mission (figure 2.6(a)). L'équipe de robots s'organise logiquement suivant une structure flexible *leader-followers* (figure 2.6(b)) pour réaliser les tâches dont elle a la charge. Les architectures robotiques peuvent être indépendantes du donneur d'ordre. Par contre, l'état est en général critique si la communication est rompue entre leaders et followers.



(a) Architectures fortement couplées.

(b) Un leader a des *followers*, qui peuvent eux-même être leaders d'autres *followers*...

FIGURE 2.6 – Architecture fortement couplée, image extraite de [Chaimowicz et al., 2001].

2.3.1 L'architecture ASyMTRe

Dans le cas d'ASyMTRe [Parker and Tang, 2006], la connaissance de chaque robot est organisée en briques de base, appelées "schémas", qui représentent les capacités élémentaires du robot (capteurs de l'environnement, schémas perceptuels, moteurs, communication). Les schémas se connectent entre eux (à l'intérieur d'un robot et entre robots) pour la réalisation de tâches, selon le type des informations requises (par les entrées des schémas) et générées (par les sorties des schémas) (figure 2.7). La détermination d'un flux d'information valide permet de construire des coalitions dans l'équipe de robots pour effectuer les tâches.

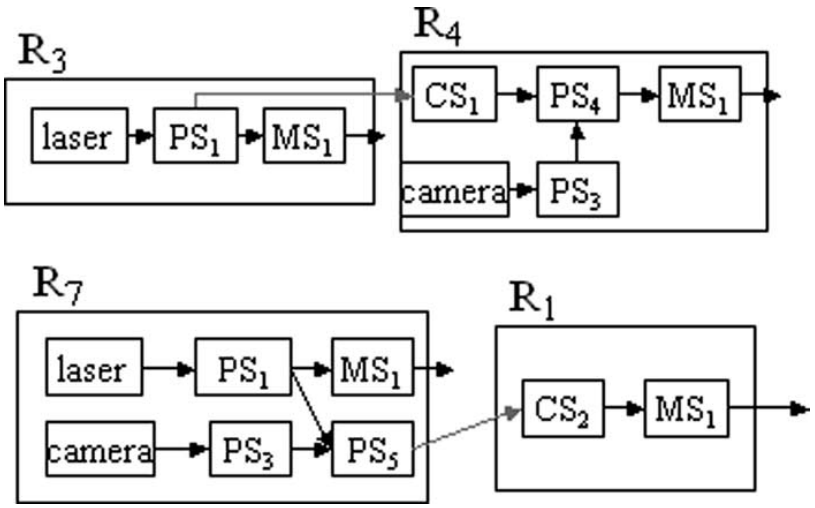


FIGURE 2.7 – Connexion des schémas dans ASyMTRe, extrait de [Parker and Tang, 2006] (R_i : Robot, PS : Perceptual Schema, MS : Motor Schema, CS : Communication Schema).

2.3.2 L'architecture GRAMMPS

Les concepteurs de GRAMMPS [Brummit and Stentz, 1998] soutiennent que pour qu'un système multirobot coopératif soit pleinement utile dans un environnement réel, il doit pouvoir exécuter efficacement un large panel de tâches complexes, dans des environnements potentiellement inconnus ou non structurés. GRAMMPS aborde le problème en couplant une grammaire à la fois facile à interpréter pour une machine et très générale pour définir des tâches, à des techniques de planification dynamiques. GRAMMPS est une architecture à deux niveaux qui couvre ainsi une large classe de domaines sur des systèmes de navigation locale et des groupes de robots hétérogènes, en fournissant des exécutions de missions "optimales" compte tenu de la connaissance courante du monde. Un robot "leader" se charge de la partie décisionnelle (le robot A, figure 2.8) et fournit les buts à accomplir au reste des robots. Des ruptures de communications sont prises en compte. En revanche, si le leader est hors service, ou incapable de communiquer avec ses coéquipiers, le système ne fonctionne plus.

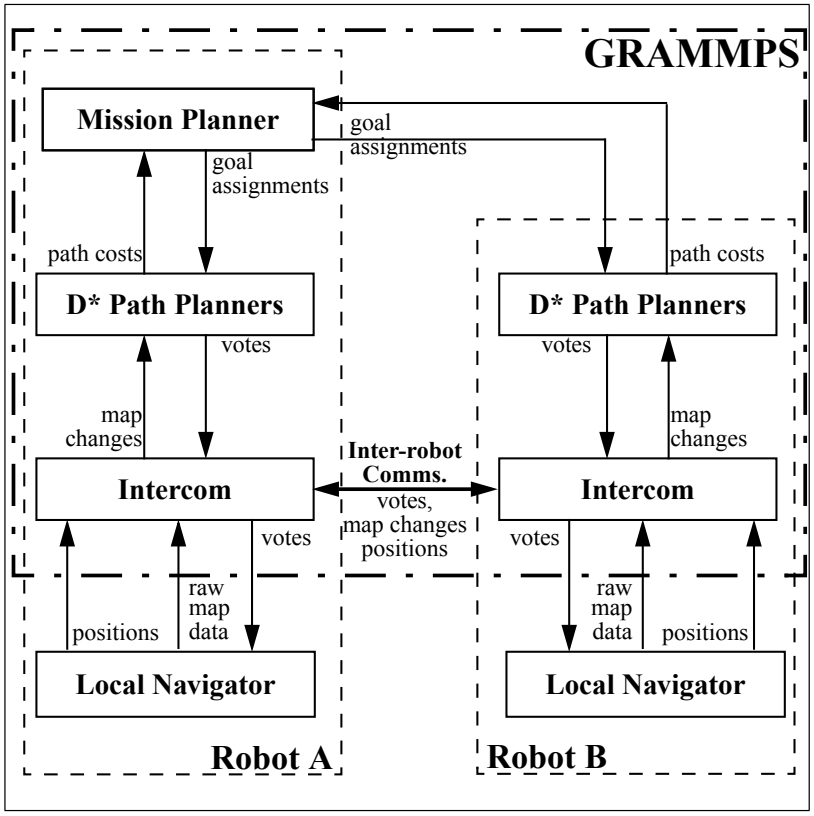


FIGURE 2.8 – L'architecture GRAMMPS, interaction entre robots, extrait de [Brummit and Stentz, 1998].

2.4 Architectures BDI

2.4.1 Cadre BDI

Le cadre de travail BDI [Rao and Georgeff, 1995] est régulièrement cité dans les SMA : il s'agit d'une implémentation logicielle du modèle "Belief-Desire-Intention" de Michael Bratman [Bratman et al., 1988] pour programmer des "agents intelligents".

- Les croyances (**Beliefs**) correspondent à la représentation qu'a l'agent du monde dans lequel il évolue, de lui-même ainsi que des autres agents. Ces croyances peuvent être exactes ou erronnées.
- Les désirs (**Desires**) représentent les motivations de l'agent, c'est-à-dire les objectifs qu'il aimerait atteindre.
- Les engagements (**Intentions**) indiquent ce que l'agent a décidé de faire. C'est le désir particulier auquel l'agent va consacrer ses actions.

L'agent BDI suit le processus en boucle suivant : à partir d'une perception de son environnement, il génère ses croyances. Celles-ci vont conduire à définir ses désirs. Parmi eux, il choisira ceux sur lesquels il va effectivement s'engager, pour agir sur son environnement, qu'il devra à nouveau percevoir, etc.

2.4.2 STEAM

Pour [Tambe, 1997], les incertitudes des domaines complexes et réels, s'opposent à un travail d'équipe lucide. Une coordination hautement flexible et la communication sont la clé pour remédier au problème. Dans sa logique, une équipe doit être capable de surveiller l'exécution de sa mission et de se réorganiser de façon flexible pour répondre à toute éventualité. Il considère que, puisque la coordination au sein d'un plan est fortement dépendante du domaine de planification, la réutilisation de cette coordination sur d'autres domaines est très restreinte. En fait, la coordination est bien souvent redéfinie à chaque nouveau domaine.

C'est pour cette raison qu'il propose l'architecture *TeamCore*, et le système de simulation STEAM (Shell for TEAM work) s'appuyant sur la *joint intention theory* et la *shared plan theory*. Dans la première théorie, l'idée est qu'une équipe est engagée conjointement dans une tâche si chacun de ses membres est engagé dans cette tâche et croit qu'il exécute la tâche. Pour la seconde, les membres de l'équipe construisent une structure complexe d'intentions communes, d'intentions individuelles et de croyances au sujet des intentions des autres membres de l'équipe.

Dans STEAM, la collaboration d'agents hétérogènes passe par l'utilisation de hiérarchies d'*opérateurs*. Un opérateur instancie une intention, qu'elle soit commune ou individuelle, et représente donc une activité. Les opérateurs principaux sont les opérateurs d'équipe, qui sont décomposés en sous-opérateurs jusqu'aux opérateurs individuels élémentaires (figure 2.9).

Le cœur de la plate-forme se veut basé sur le travail d'équipe. La mission est donc construite autour d'elle. STEAM a été conçu pour des domaines complexes et tirés d'un monde réaliste. Trois scénarios ont été implémentés et testés avec STEAM. Chaque scénario définit pour l'équipe une structure hiérarchique complexe de sous-équipes. Ce travail reste cependant basé sur le cadre BDI, ce qui implique des problèmes de cohérence des données à prendre en compte, la non représentation explicite des buts, et la recherche de plan optimaux difficile.

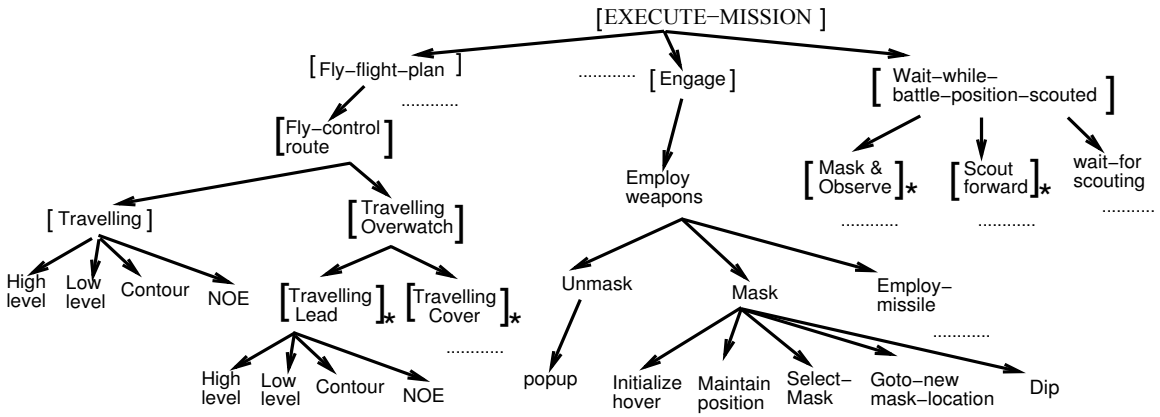


FIGURE 2.9 – Exemple de hiérarchie d'opérateurs du domaine "Attack", dans STEAM, extrait de [Tambe, 1997]. Les opérateurs d'équipe marqués par un '*' sont exécutés par des sous-équipes de ce domaine.

2.4.3 Retsina

L'architecture Retsina proposée par [Paolucci et al., 2000] utilise un module de planification qui permet d'entrelacer planification et exécution : HITaP (HIerarchical Task Network planner), basé sur des structures HTN.

D'après les auteurs, il est impossible pour un système multiagent de suivre un schéma « classique » dans lequel un plan complet est construit puis exécuté. Le risque d'aléa est en effet trop grand en environnement dynamique. Ils proposent donc l'idée suivante : un agent lance une phase de planification où il construit son propre plan, jusqu'à ce qu'il s'aperçoive qu'il a besoin d'informations, d'aide d'un autre agent, ou qu'il a besoin de se synchroniser avec d'autres agents. La phase de planification est alors suspendue, et l'agent exécute la partie du plan planifiée, puis exécute ensuite des tâches qui lui permettent de pallier les manques qui avaient stoppé le processus de planification. Le processus de planification établit une liste dynamique d'objectifs stockés sous la forme d'une queue à priorités. L'architecture se décompose en quatre modules autonomes qui s'exécutent en parallèle : un "communicator", un "planner", un "scheduler" et un "executor monitor" qui communiquent en KQML [2] (figure 2.10). Les requêtes d'un agent sont transformées par le "communicator" en "buts" (au sens de la planification) pour le "planner". Les objectifs de l'agent seront stockés dans une base de données "Objective DB". A ces objectifs sont associés des priorités. Les objectifs les plus prioritaires sont choisis par le planificateur qui construit un plan permettant d'atteindre le but. Pour construire ce plan, le planificateur utilise une "Task Schema Library" (type de tâche pouvant être accomplie par l'agent) et une "Task Reduction Library" (décomposition des tâches complexes en tâches élémentaires - figure 2.11). Le "planner" a accès à la "BeliefDB" (Base de données de stockage de la connaissance du domaine, des croyances qui peuvent changer au fur et à mesure que les actions sont exécutées).

Le plan construit est stocké dans une autre base de données (TaskDB), interface entre le "planner" et le "scheduler". Les tâches ajoutées par le planner, quand elles sont prêtes à être exécutées, sont "prises" et effacées par le "scheduler" qui choisit à quel moment elles doivent être exécutées. Le résultat du scheduling est directement utilisé par l'exécution.

2. Knowledge Query and Manipulation Language - `http://www.csee.umbc.edu/csee/research/kqml/` (Septembre 2012)

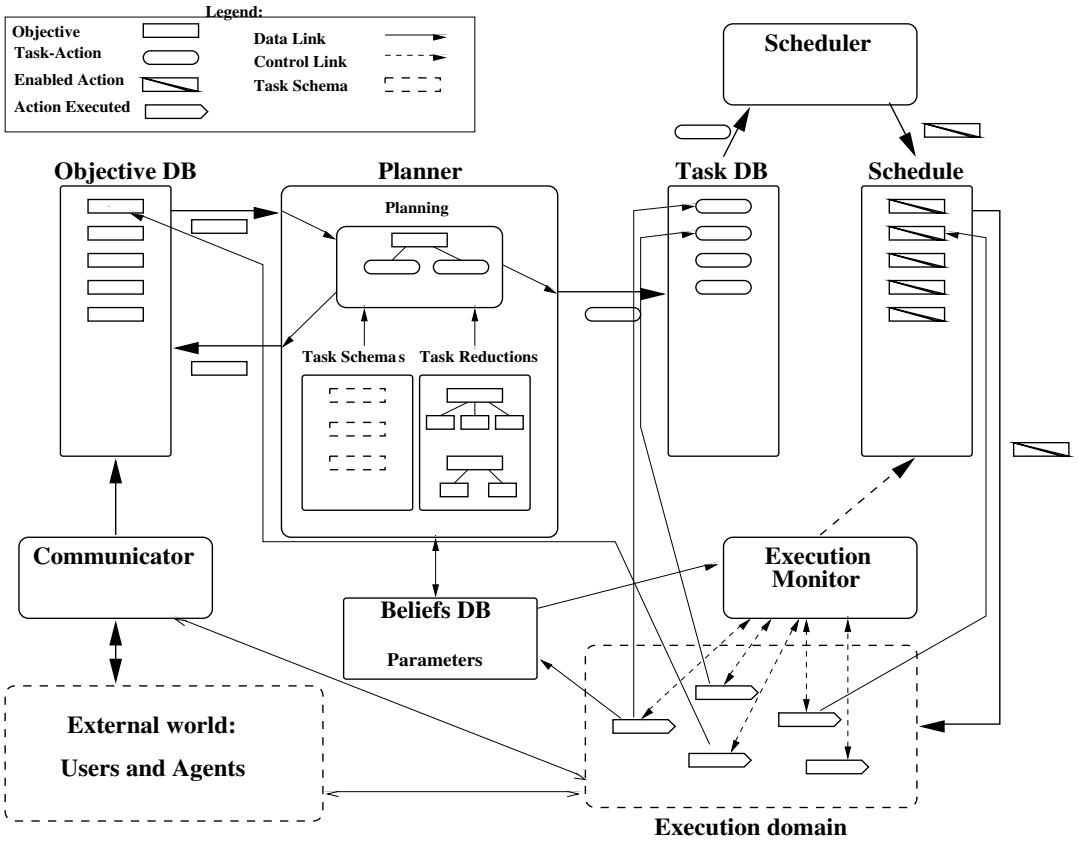


FIGURE 2.10 – L'architecture Retsina [Paolucci et al., 2000].

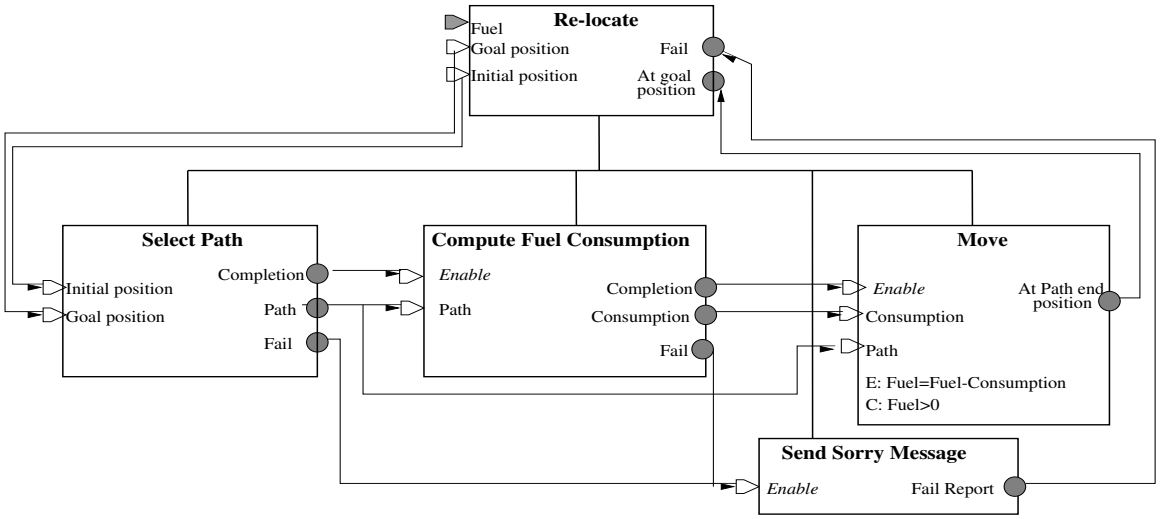


FIGURE 2.11 – Schéma de décomposition d'une tâche "Re-locate" [Paolucci et al., 2000].

Cette architecture semblerait n'avoir été testée qu'en simulation. Elle est basée sur le cadre BDI : les agents construisent des plans partagés, et des négociations ont lieu entre eux pour se mettre d'accord sur leurs objectifs respectifs. Il n'y a pas explicitement de plan global d'équipe, et donc pas de notion de plan optimal.

2.4.4 IDEA

L'architecture IDEA (Intelligent Distributed Execution Architecture) [Muscettola et al., 2002] lie planification et exécution, et est composée d'un ensemble d'agents qui s'appuient sur un même formalisme. Elle a été conçue pour gérer finement des problèmes liés à la multiplicité des modèles, rencontrés notamment dans des architectures monovéhicules. A chaque intervalle de temps d'exécution élémentaire, dénommé *tokens*, les agents IDEA exécutent une *procédure* comprenant des arguments d'entrée, de mode, de sortie et qui retourne une valeur de *statut*. L'exécution de la *procédure* se termine lorsque une valeur de *statut* est retournée, ou que l'agent décide de l'interrompre (par exemple, déclenchement d'un *time-out*). La figure 2.12 montre un agent IDEA. La communication entre deux agents IDEA est assurée via le *IDEA Comm Wrapper*, et sert à envoyer des messages d'initialisation d'exécution de procédures sur des agents distants, et à recevoir des buts sous forme de *tokens*. Le format de ces communications est décrit au sein du *Model* : celui-ci précise les interfaces des *procédures* des agents (i.e. les arguments et les valeurs de *statut* des procédures). Un plan centralisé (figure 2.13) et régulièrement mis à jour (*Plan Database*) précise les *tokens* à exécuter, mais garde également en mémoire ce qui a été exécuté par le passé, et les *tokens* en cours d'exécution. Le *Plan Runner* est activé de façon asynchrone lorsqu'un message en provenance d'un autre agent a été reçu, ou lorsqu'un timer s'est terminé. Une fois l'origine de cette activation analysée, il fait appel à un *Reactive Planner* chargé de renvoyer un plan localement exécutable, que le *Plan Runner* va être chargé de faire exécuter. L'approche par plan centralisé présente des avantages indéniables. En revanche, une rupture de communication entre agents IDEA rend l'accès à cette information impossible.

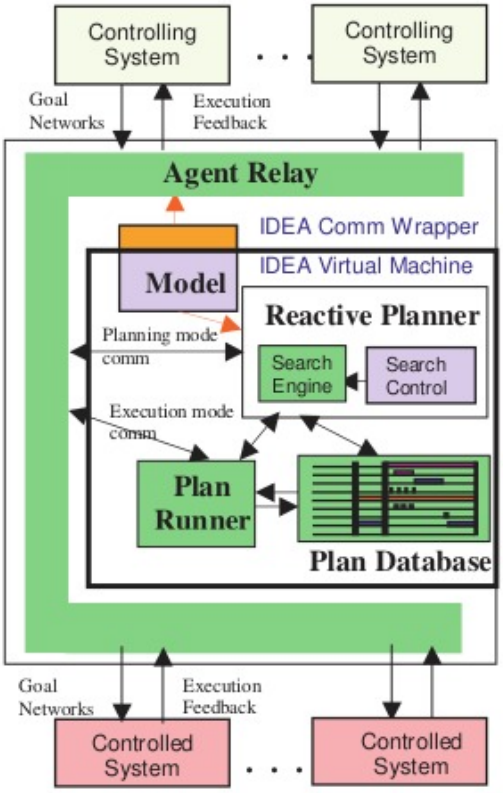


FIGURE 2.12 – Un agent IDEA [Muscettola et al., 2002].

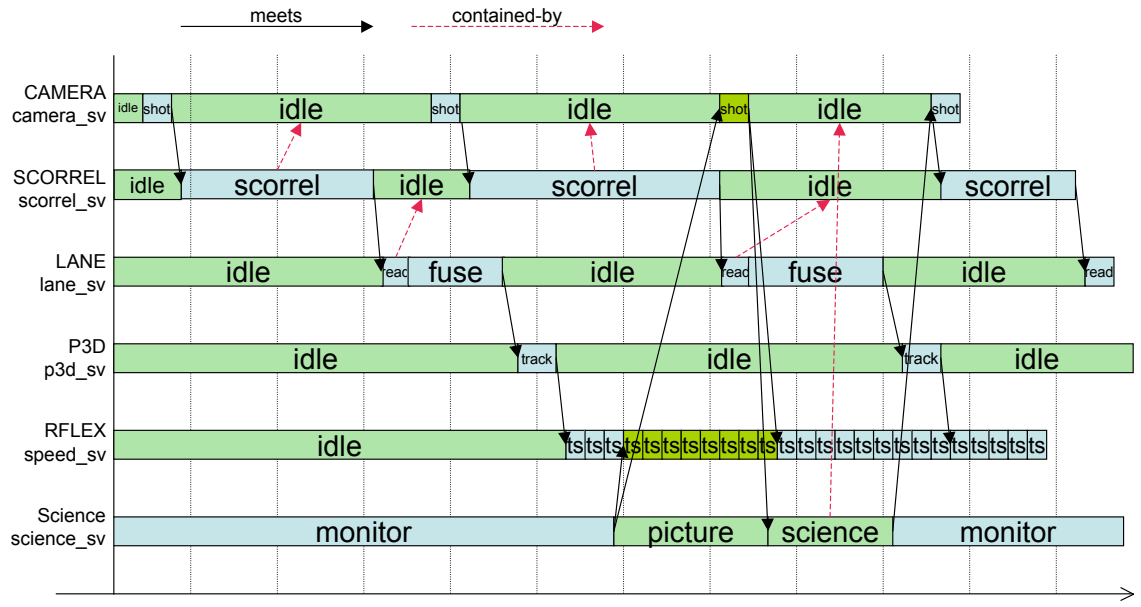


FIGURE 2.13 – Exemple de timelines IDEA [Muscettola et al., 2002].

2.4.5 HARPIC

HARPIC (*Hybrid Architecture based on Representations, Perception and Intelligent Control*) [Luzeaux and Dalgalarrondo, 2001] est un système multiagent qui gère les ressources matérielles et logicielles d'un ou plusieurs robots en lien avec une ou plusieurs interfaces homme-machine. A bord d'un seul véhicule, les agents de type ressource ont chacun un rôle particulier : perception, action, attention, sélection de comportement et d'interface avec des matériels ou des opérateurs humains (figure 2.14). Les capteurs du robot envoient des informations à l'agent de perception, qui créé des *représentations* de l'environnement, sous contrôle de l'agent d'attention. L'agent de sélection d'action choisit le comportement du robot en fonction d'objectifs prédéfinis, de l'action courante, des représentations et de l'indice de confiance de ces dernières. Les actionneurs du robots sont alors sollicités, tout en maintenant une sorte de boucle fermée avec l'agent de perception.

La communauté des agents qui constitue l'architecture est distribuée sur des PC ou PDA embarqués sur les robots ou à la disposition des opérateurs. Au sein d'un même robot, les agents sont représentés par des Threads POSIX et communiquent par messages. On peut noter la présence d'un agent de communication (non représenté sur la figure) qui permet d'échanger des données avec d'autres robots.

HARPIC illustre le concept d'autonomie ajustable (le robot seul peut modifier son autonomie vis-à-vis de l'opérateur [Mercier, 2010]) et démontre ce que pourraient être des moyens robotisés à la disposition d'un groupe de combat pour des missions de reconnaissance. Par exemple, la plate-forme Pioneer (un robot d'expérimentation terrestre) est capable de réaliser une carte de son environnement dont l'opérateur se sert à la fois pour commander le robot et pour reconnaître cet environnement. L'architecture semble cependant plus portée sur des problématique d'interaction entre opérateurs et véhicules autonomes.

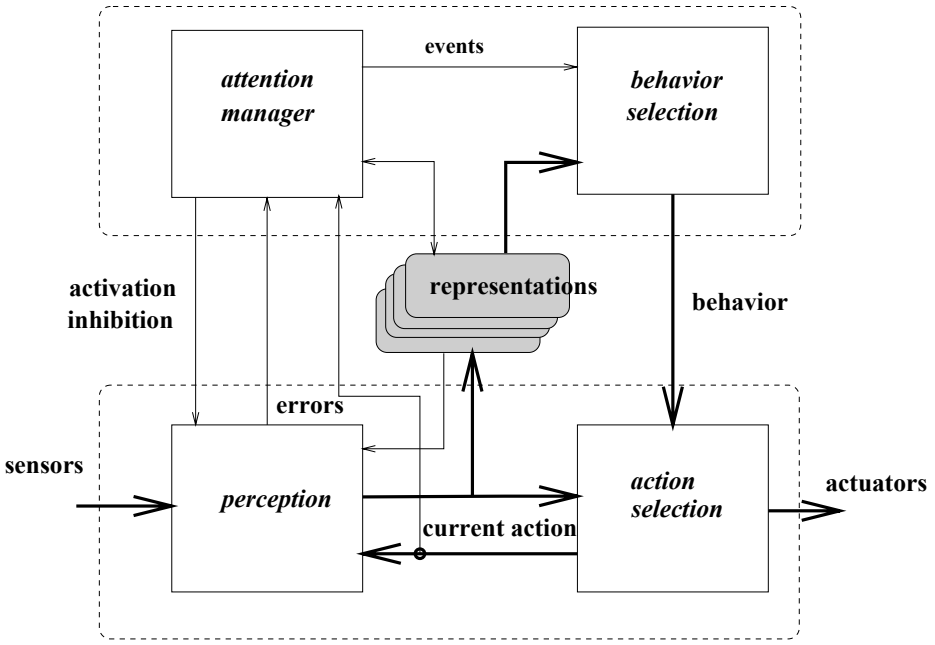


FIGURE 2.14 – Diagramme fonctionnel de l'architecture HARPIC pour un véhicule [Dalgalarrondo, 2001].

2.4.6 Plans partiels dans un cadre BDI

Certains auteurs [Boella and Damiano, 2008] abordent le problème de la replanification sous un autre angle, plus dans l'idée d'une "modification" du plan pour optimiser ce dernier et en améliorer la qualité. En effet, malgré le fait que certains problèmes de replanification soient plus durs que la planification elle-même [Nebel and Koehler, 1993], ils parviennent à obtenir empiriquement de meilleurs résultats en utilisant justement de la replanification locale. Ils exploitent pour cela des plans *partiels*, i.e des plans dont toutes les variables ne sont pas instanciées. 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. L'algorithme de replanification consiste à se "rétracter" d'un choix de raffinement qui s'est déroulé durant la phase initiale de planification, et à relancer le processus de raffinement sur le plan partiel ainsi déterminé. Ils se placent cependant dans un cadre BDI, et même si l'intérêt théorique est certain, l'application à des plans multirobots en environnement réel semble moins évident.

2.5 Architectures basées sur l'allocation de tâches

Plusieurs auteurs ont travaillé sur de l'allocation de tâches parmi un ensemble d'agents, hors-ligne, et pendant la phase d'exécution. On peut notamment citer l'approche par CNP (Contract Net Protocol), proposée par [Smith, 1980] ; dans [Zlot, 2005], un plan HTN [Erol et al., 1994] est utilisé pour décrire des tâches à exécuter, couplé à une approche par Contract Net Protocol.

Un autre exemple d'architecture par allocation de tâches est proposée par Miyata [Miyata et al., 2002] qui s'intéresse au transport coopératif d'objets en environnement inconnu. Cependant, les robots pris en charge sont tous du même type, et la problématique des communications n'est pas traitée.

2.5.1 L'architecture Cobos

Dans Cobos (**Co**operative **b**ack**o**ff adaptive **s**cheme) [Fua and Ge, 2005], chaque robot a et maintient à jour un modèle des capacités (les siennes et celles qu'il connaît des autres par la communication), des différentes façons de réaliser les tâches, et des dépendances entre tâches. Il maintient également une matrice de « pertinence » (*suitability matrix*), la pertinence (un nombre entre 0 et 1) étant la compatibilité entre les capacités intrinsèques du robot et la nature de la tâche (si la tâche est une tâche complexe, la pertinence est l'agrégation des pertinences correspondant aux tâches élémentaires qui la composent - les pertinences élémentaires sont spécifiées au départ). Cette approche permet d'accroître la flexibilité de l'équipe de robots, chacun d'eux adaptant ses actions en fonction de son expérience. La portée des communications de chaque robot est limitée.

2.5.2 L'architecture DEMiR-CF

DEMiR-CF (Distributed and Efficient MultiRobot-Cooperation Framework) [Sariel-Talay et al., 2009] est un système conçu pour de la coopération multirobots sur des missions de type multi-voyageurs de commerce en environnement réel (cf. figure 2.15). Des aléas de communications sont explicitement pris en compte. Il est basé sur de l'allocation de tâches, implémentée par des CNP. L'état du monde n'est pas supposé exactement connu à chaque instant.

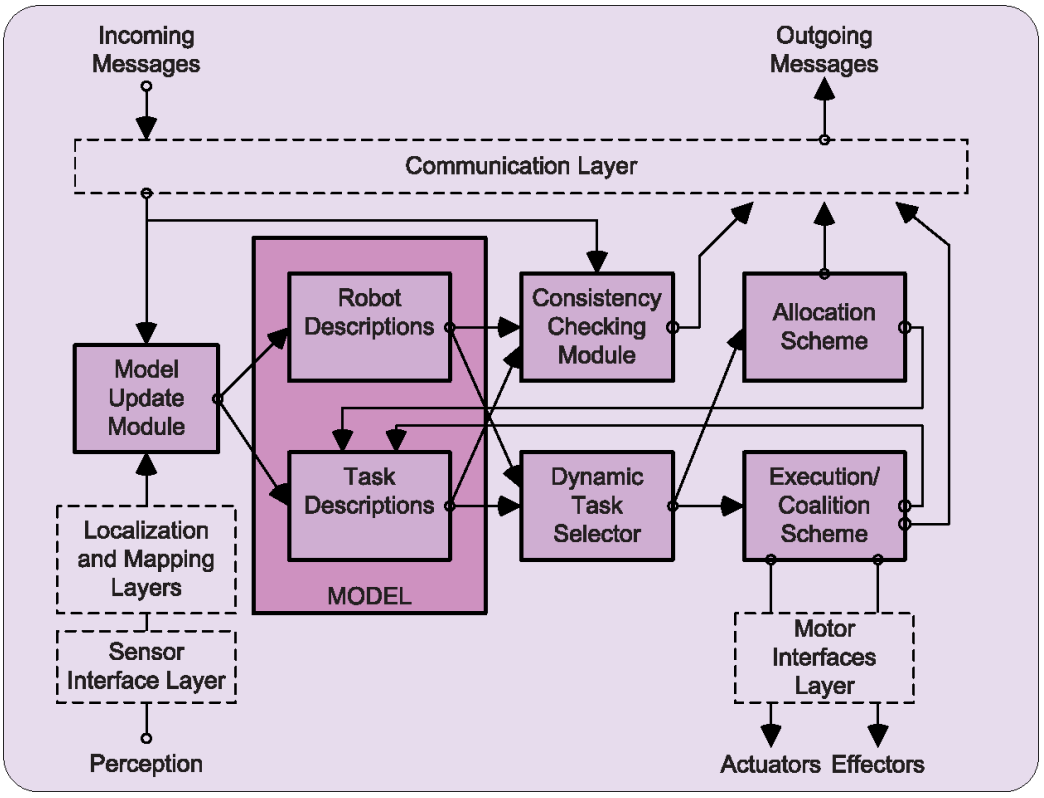


FIGURE 2.15 – Architecture DEMiR-CF pour un véhicule, extrait de [Sariel-Talay et al., 2009].

2.5.3 Limites des approches par allocation de tâches

Les approches d'allocation de tâches et d'ordonnancement (scheduling) ne semblent cependant pas toujours très adaptées aux contextes multivéhicules. En effet, comme le souligne d'ailleurs [Parker, 2008], "prises indépendamment, ces approches ne sont pas directement applicables à des missions multirobots, puisqu'elles ne prennent pas en compte les performances multirobots en un environnement dynamique qui impliquerait une hétérogénéité entre les robots, des incertitudes sur les senseurs et le non-déterminisme des actions des robots". De plus, la négociation sous-entend la nécessité de communiquer entre entités négociantes.

2.6 Replanification et réparation de plans pour les architectures multirobots

Nous considérerons dans la suite que le terme "replanification" est associé à une remise en cause du plan complet de mission, opposé à une "réparation", qui ne se contente de modifier que localement le plan de mission. Concrètement, une réparation peut concerner l'ensemble du plan (par exemple, si la tâche racine est remise en cause dans un plan hiérarchique) et inversement, une replanification peut ne consister pour un planificateur à ne modifier qu'une partie du plan existant en exploitant le travail de planification effectué pour fournir le plan de mission initial.

2.6.1 Réparation de plans hiérarchiques, agenticité et hiérarchie d'équipe

Des travaux sur les SMA ont cherché à organiser une structure pour l'équipe en charge de la réalisation d'une mission. Une étude approfondie de la structure dynamique de sous-équipes est proposée par [Bonnet-Torrès and Tessier, 2005]. Ils définissent en particulier la notion d'*agenticité* (la profondeur dans la hiérarchie d'une sous-équipe) qui amène à définir une équipe comme une *hiérarchie d'agenticité*. Les plans de mission sont représentés par des réseaux de Petri hiérarchiques. Des structures particulières dans le réseau de Petri qui représente le plan de mission de l'équipe d'agents modélisent la dynamique des sous-équipes au sein de l'équipe : arrivée d'un agent dans une sous-équipe, départ d'un agent d'une sous-équipe, transfert d'un agent d'une sous-équipe à une autre, scission, réunion de sous-équipes. Un agent peut appartenir simultanément à plusieurs sous-équipes en contribuant à différentes tâches *via* les différents services qu'il peut offrir (par exemple : photographier une cible et relayer des informations), du moment que les contraintes de ressources et d'espace sont respectées. La réduction (au sens des réseaux de Petri) successive des différentes structures permet de hiérarchiser le plan selon les différentes profondeurs de sous-équipes (figure 2.16) : la supervision du plan pourra donc être effectuée au niveau de sous-équipe pertinent selon la phase de mission.

En cas d'aléa, une réparation aussi locale que possible est tentée. Il est possible de remonter dans la hiérarchie des sous-équipes si nécessaire, successivement à partir des heuristiques de substitution d'agent, de substitution de recette, de substitution de sous-équipe (figure 2.17). La réparation de plan partiel est donc explicitement étudiée. Cependant, la question de la communication n'est pas réellement abordée, et les formalismes des phases de planification et d'exécution diffèrent [3]. Ce principe de réparation locale est plutôt adapté à des équipes dont les sous-équipes sont faiblement couplées (cas par exemple de sous-équipes évoluant dans des sous-zones distinctes). Disposer d'une équipe dynamique sous-entend que les divers co-équipiers peuvent communiquer.

3. Les résultats de la planification ne sont pas directement exploitables par le moteur d'exécution.

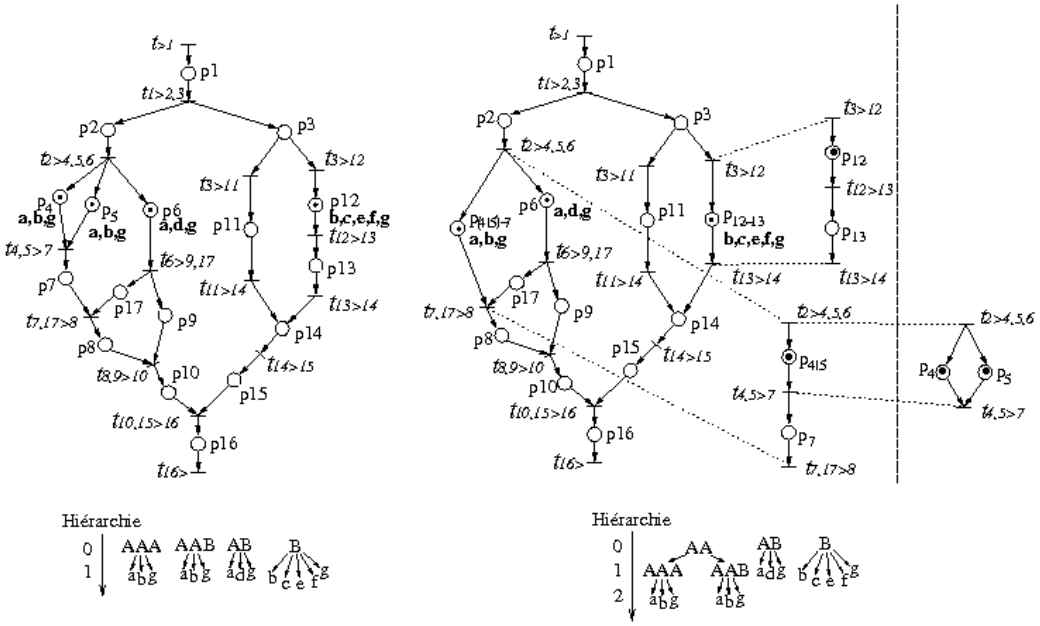


FIGURE 2.16 – Hiérarchisation du plan : à gauche, le plan est « à plat », le marquage courant fait apparaître que les agents a, b et g réalisent en parallèle les tâches correspondant aux places p_4 et p_5, les agents a, d et g réalisent la tâche correspondant à la place p_6 et les agents b, c, e, f et g réalisent la tâche correspondant à la place p_{12} ; la réduction de la structure de parallélisme entre p_4 et p_5 donne la place $p_{4|5}$, qui correspond à la tâche réalisée par la sous-équipe AA, constituée des agents a, b et g.

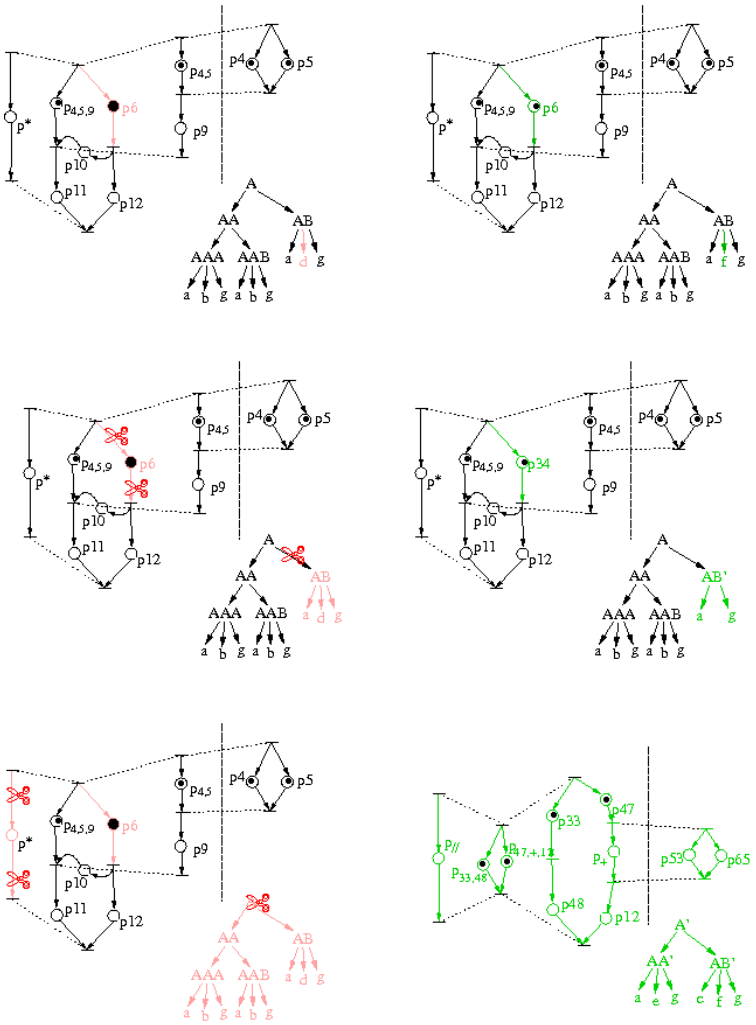


FIGURE 2.17 – Réparation du plan suite à la défaillance de l'agent d, de la plus locale à la moins locale : substitution d'*agent* (en haut : l'agent d défaillant est remplacé par l'agent f), de *recette* (au milieu : la place p_6 est remplacée par la place p_{34} qui correspond à une autre recette ou à un autre sous-but impliquant la sous-équipe $A'B'$, constituée seulement des agents a et g), de *sous-équipe* (en bas : toute la sous-équipe A correspondant à la place réduite $p*$) est remplacée par la sous-équipe A' correspondant à la place réduite $p_{//}$, qui implique la sous-équipe AA' formée des agents a,e et g, et la sous-équipe AB' formée des agents c, f et g.

2.6.2 HOTRiDE

Des idées de réparation ont déjà été proposées dans des contextes monoagents. Avec HOTRiDE (Hierarchical Ordered Task Replanning in Dynamic Environments), [Fazil Ayan et al., 2007] proposent d'exécuter un plan et de la réparer en ligne sans notion d'équipe. L'idée consiste à entrelacer génération, exécution et réparation d'un plan structuré en HTN, en s'aidant d'un planificateur SHOP [Nau et al., 1999]. L'agent dispose *a priori* d'un plan HTN. Un graphe de dépendance entre les tâches d'un HTN 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 (figure 2.18). En revanche, ce n'est pas dans le cadre d'un système multiagent. De plus, lorsque l'on traite des tâches "haut niveau", et si une des tâches principales pour la mission échoue, toutes celles qui suivent dans le plan sont *a priori* compromises.

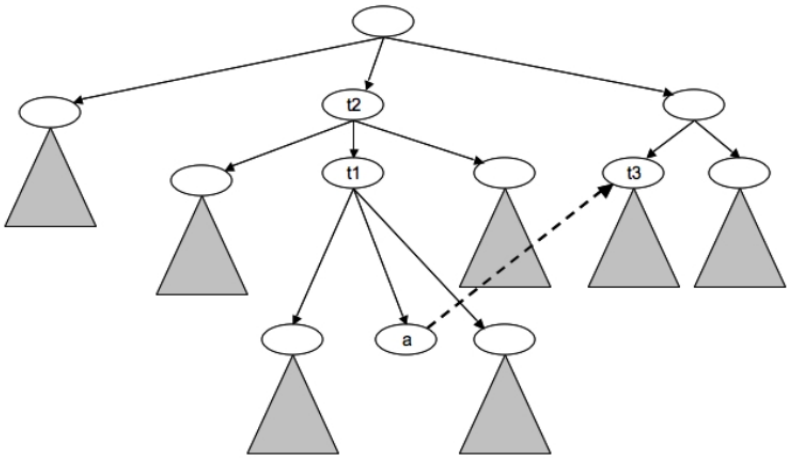


FIGURE 2.18 – Exemple d'un graphe de dépendance entre tâches HTN.

2.6.3 Domaines spécifiques

On peut noter que des architectures très spécifiques à certaines applications existent aussi. Par exemple, celle proposée par Frederik Heger se concentre spécifiquement sur des tâches d'assemblage de structures [Heger and Singh, 2002], et dispose d'un planificateur dédié, et d'un système pour exécuter les plans produits. Le problème de réparation / re-planification n'est abordé et traité que dans ce domaine très particulier, ce qui facilite sa formalisation et son traitement.

2.7 Un premier bilan

A l'exception de l'approche inspirée de BEAR [Vidal et al., 2002] qui concerne explicitement des équipes mêlant UAV et AGV, ou de celles basées sur ALLIANCE [Kannan and Parker, 2007] qui ont permis des implémentations concrètes sur des robots différents, l'hétérogénéité des véhicules ne semble pas être une préoccupation majeure des travaux précédemment cités. Les approches multiagents/multirobots peuvent impliquer des agents/robots hétérogènes, mais ceux-ci semblent dotés *a minima* d'interfaces standardisées au sein de l'équipe. Aucune architecture ne semble proposer

de solution idéale pour la gestion des communications, même si certaines cherchent plutôt à les limiter [Vidal et al., 2002], notamment lorsque le milieu opérationnel les contraint fortement [Sotzing et al., 2008, Belbachir et al., 2012]. Lynne Parker [Parker, 1998] pose des hypothèses de base totalement valables dans nos travaux, à savoir :

1. les robots ne sont pas intentionnellement adversaires, et ne "mentent" pas,
2. les moyens de communication ne sont pas toujours disponibles,
3. les capteurs et actionneurs des robots ne sont pas parfaits,
4. les sous-systèmes composant un robot peuvent entrer dans un état d'erreur avec une probabilité non nulle,
5. si un robot est dans un état d'erreur, il ne communiquera pas nécessairement son état d'erreur aux autres robots et
6. un stockage centralisé de l'état complet du monde et des données n'est pas disponible.

En revanche, de par l'impossibilité de s'interfacer directement avec les actionneurs du robot dans notre cadre de travail, nous nous plaçons à un niveau plus délibératif.

Le tableau 2.2 récapitule pour les principales architectures multivéhicules citées dans l'état de l'art quatre points qui nous semblent incontournables pour la supervision que nous souhaitons réaliser, à savoir : la prise en considération des communications, une gestion de leur perte éventuelle, les réponses face à des aléas, l'hétérogénéité des véhicules ou agents et l'existence d'expérimentation.

TABLEAU 2.2 – Récapitulatif des architectures précédentes (en *italique*, informations déduites).

Architecture	Communication	Pertes des communications	Gestion des aléas	Architecture monovéhicule conservée	Expériment-ations
STEAM [Tambe, 1997]	permanente	non abordées	**réorganisation d'équipe**	non précisé	**oui**
ALLIANCE [Parker, 1998]	permanente	**gérées**	**approche réactive**	non	**oui**
GRAMMPS [Brummit and Stentz, 1998]	non abordée	non abordées	**planification dynamique**	non précisé	**oui**
Retsina [Paolucci et al., 2000]	non abordée	non abordées	**approche délibérative**	non applicable	simulations
HARPIC [Luzeaux and Dalgalarrondo, 2001]	permanente	non abordées	non abordée	non	**oui**
BEAR [Vidal et al., 2002]	réduite	**gérées**	**oui, à tous niveaux d'abstraction**	**oui, rendue abstraite**	**oui**
IDEA [Muscettola et al., 2002]	permanente	non abordées	plan centralisé	*oui*	*non*
BIIMAPS [Sotzing et al., 2008]	réduite	gérées	**approche délibérative**	*oui*	**oui**
T-Rex [Belbachir et al., 2012]	réduite	non abordées	**approche délibérative**	*oui*	simulations
COBOS [Fua and Ge, 2005]	permanente	**gérées**	**par allocation de tâches**	*oui*	simulations
ASyMTRe [Parker and Tang, 2006]	permanente	non	non abordées	**oui**	**oui**
DEMIR-CF [Sariel-Talay et al., 2009]	permanente	non abordées	**oui**	non	simulations
[Doherty et al., 2009]	*permanente*	**gérées**	**oui**	*oui*	**oui**
[Gancet, 2005]	*permanente*	**gérées**	**oui**	*oui*	simulations
[Bonnet-Torrès and Tessier, 2005]	*permanente*	non	**réparation de plans hiérarchisés**	non applicable	simulations

2.8 Point clé : la communication

2.8.1 Un problème récurrent

Les contraintes sur les communications semblent être un problème récurrent dès que les véhicules circulent dans un environnement dynamique opérationnel. Elles sont généralement une des composantes clés pour la coordination et le partage d'informations entre véhicules. Nombre de travaux tels que [Dix et al., 2003] 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 [face à des événements imprévus], plus la charge de communication est élevée" [Tambe, 1997]. Et inversement, une réduction de cette charge entraîne une perte en robustesse, d'autant plus lorsque l'environnement est dynamique.

Des travaux comme ceux de Legras [Legras and Tessier, 2003] définissent une sous-équipe en se basant sur les liens de communication.

Baser son architecture sur le cadre BDI implique généralement 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 genre de structure. Dans le travail de [Magnusson et al., 2008], les agents perdent leur capacité de replanification locale en cas d'interruption des communications. En revanche, certains auteurs [Joyeux et al., 2007, Micalizio and Torasso, 2008] 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.

Dans [Sotzing et al., 2008], les actions d'un véhicule distant sont prédites lorsque la communication n'est pas disponible. Lors d'un retour à un état où la communication est possible, le véhicule met à jour ses connaissances.

2.8.2 Maintenir explicitement une connexion entre les agents ?

Un certain nombre de travaux s'intéressent au maintien permanent d'une connectivité entre véhicules autonomes durant le déroulement de la mission. Ils prennent explicitement en compte les contraintes de communication, et garantissent, en organisant des stratégies de routage des informations, que l'ensemble des véhicules autonomes impliqués sur la mission maintiennent un lien de communication, même si celui-ci doit parfois passer par des relais.

A titre d'exemple, nous pouvons citer [Mosteo et al., 2008] qui proposent divers algorithmes sur les stratégies de routage, ou encore [Rekleitis et al., 2004], qui, pour effectuer une couverture de zone, tient compte des contraintes de communication (par exemple, basées sur la visiblité) et organisent des stratégies d'exploration en fonction d'elles. Des auteurs comme [Rooker and Birk, 2007, Atay and Bayazit, 2007] évitent explicitement de perdre la communication entre agents et utilisent une partie de ces derniers comme relais. Le problème des communications est donc divisé en problèmes de résolution de contraintes pour les liens locaux de communications, situation qui conduit cependant à un problème encore plus critique lorsque les communications ne peuvent être garanties pendant le déroulement de la mission. En conséquence, ces méthodes semblent difficiles à appliquer dans notre contexte.

2.9 Conclusion sur l'état de l'art et intuitions

Plusieurs difficultés ressortent de cette étude. Les communications semblent primordiales mais la considération de leur absence non triviale. Le traitement des aléas est fondamental et doit être pris en compte. Mais la cohérence du système en cas de réparation de plan est un autre point dur. Organiser une structure d'équipe particulière semble bénéfique au déroulement de la mission, notamment pour les éventuelles réparations en cas d'aléas. Les notions issues des SMA semblent exploitables, mais leur mise en œuvre concrète délicate. Elles impliquent souvent des charges élevées de communication. Les architectures multivéhicules présentées dans cet état de l'art sont toutes très liées à l'architecture locale des véhicules autonomes, ou requièrent une forte adaptation de l'architecture locale des véhicules, voire leur redéfinition. Nous aimerions analyser si le contraire est possible : avoir une architecture multivéhicule la plus générique possible, qui puisse s'interfacer directement avec des architectures locales déjà fonctionnelles, mais hétérogènes.

En découle un autre point qui semble peu abordé : celui de la formalisation, des données et des interfaces. En effet, un certain nombre d'entités logicielles différentes sembleraient être amenées à s'interfacer et échanger des données. Généralement, un accord commun sur le format de données existe implicitement entre les véhicules autonomes d'une même équipe, mais celui-ci n'est pas formellement décrit. Plusieurs exemples vont dans ce sens. Avec l'architecture inspirée de BEAR, on note une complexité d'adaptation sous-jacente des connexions entre modules. Même si les données d'environnement sont suffisamment détaillées pour être fournies à un planificateur tactique ou stratégique, elles restent très spécifiques à la mission traitée. Dans d'autres architectures, des entités (agents dans Harpic, réacteurs dans T-Rex...) spécialisées peuvent se charger des échanges de données entre agents. IDEA a d'ailleurs été créée pour mieux gérer les problèmes liés à la multiplicité des modèles dans les architectures classiques. Côté formalisation des interfaces, nous avons aussi pu voir que des auteurs utilisaient explicitement des middlewares tels que CORBA pour interfacer leurs différents services [Doherty et al., 2009].

La problématique de décalage entre la planification d'une mission et l'exécution d'un plan est explicitement posée par Dix [Dix et al., 2003]. Les plans ne sont pas toujours directement exécutables, ni triviaux à régénérer pendant le déroulement d'une mission. Un autre auteur [Boddy, 2003] met en relief le fait que PDDL (Planning Domain Definition Language) [McDermott et al., 1998], le langage conçu pour l'IPC (International Planning Competition), reste mal adapté à la représentation de problèmes du monde réel, et ce malgré son extension en PDDL 2.1 [Coles et al., 2009].

Différentes agences spatiales (ESA, NASA...) ont déjà commencé à évoquer les futurs besoins en terme de standards et d'interopérabilité [Tramutola and Martelli, 2010, Chien et al., 2006, Merri et al., 2002]. On peut dans ce sens remarquer l'utilisation de standards pour les communications et de certaines procédures [Kazz and Greenberg, 2002], au niveau matériel. Au sujet des missions multirobots, il est à noter que la formalisation des protocoles d'échanges en général est loin d'être satisfaisante. En premier lieu, au niveau de l'exécution de plan et de leur réparation en-ligne, des interfaces ad-hoc (entre les planificateurs et les véhicules) sont implicitement utilisées pour traduire les plans de mission en un langage qui puisse être interprété par les véhicules autonomes. La réparation de plan est souvent prise du point de vue de problématiques de planification pure, tandis que la question de la génération des données qui doivent être pourtant fournies au planificateur - et provenant de sources variées - semble peu abordée. Ensuite, l'hypothèse que les véhicules autonomes d'une même équipe disposent des mêmes standards pour leurs communications ou leur type de données est souvent implicitement posée. En pratique, ce n'est pas toujours le cas.

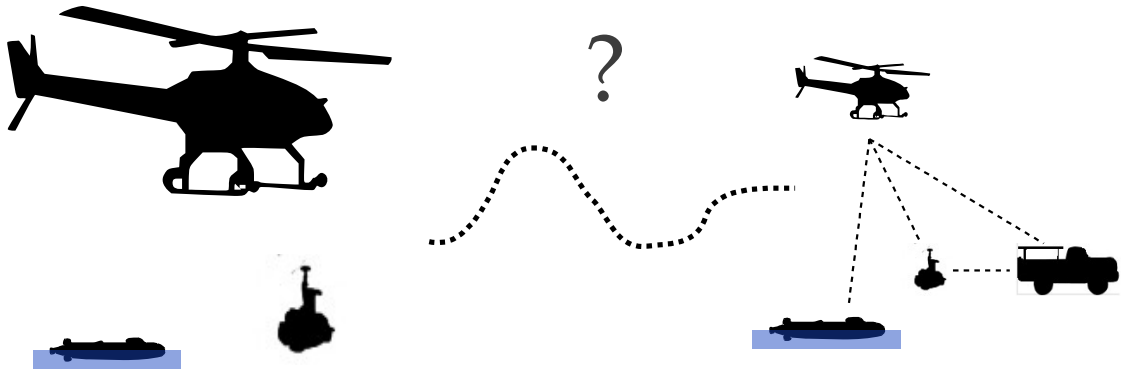
Dans quelle mesure est-il possible de **formaliser** les échanges et les données entre les composants logiciels liés à la supervision ? L'hétérogénéité des véhicules implique certainement une

représentation non standardisée de leurs connaissances, lesquelles doivent être néanmoins partagées s'ils souhaitent coopérer dans l'exécution de leur mission d'équipe. D'ailleurs, ces connaissances peuvent-elle être directement représentées au sein des entités décisionnelles d'équipe ? Comment seront transmises les informations sur la chaîne décisionnelle et jusqu'à l'architecture locale des véhicules autonomes eux-mêmes ? C'est à ces questions que ce mémoire tente d'apporter des réponses.

En nous inspirants des architectures que nous avons pu étudier, nous avons décidé de concevoir notre propre architecture décisionnelle, répondant en particulier aux besoins du PEA Action. Nous en donnons une description au chapitre suivant.

CHAPITRE 3

Architecture de supervision distribuée



Dans ce chapitre, nous définissons les besoins pour la supervision d'une équipe de véhicules autonomes hétérogènes, ce qui nous amène à une description générale de l'architecture distribuée HiDDeN que nous proposons d'utiliser en tant que superviseur haut niveau. Nous fournissons ici une courte description des modules qui la composent, dont le fonctionnement sera approfondi dans les chapitres suivants.

Sommaire

3.1 Spécifications de la composante supervision

3.1.1 Rappel du contexte opérationnel

Nous souhaitons concevoir et mettre en œuvre une architecture de haut niveau, un *superviseur*, capable de contrôler une équipe de véhicules autonomes hétérogènes et lui permettre d'exécuter des missions coopératives suivant un plan de mission défini hors-ligne. Le superviseur doit être capable de prendre en charge des cas où un véhicule ne dispose momentanément pas de lien de communication vers ses coéquipiers et doit néanmoins prendre une décision seul, suite à un aléa par exemple.

Des véhicules hétérogènes

Les membres de l'équipe de robots à superviser sont tous supposés déjà *autonomes* : chacun embarque une *architecture décisionnelle* monovéhicule qui lui confère une autonomie individuelle. Ces architectures de contrôle leur permettent d'accomplir les missions pour lesquelles les robots ont été conçus. Nombre de véhicules autonomes sont actuellement opérationnels, et ont déjà subi un large éventail d'essais et de tests (cf. section 1.3, p. 8). Ces véhicules peuvent également différer de par leur nature (véhicules autonomes aériens, terrestres, sous-marins, de surface, spatiaux...) provenir d'époques (cas de déploiements incrémentaux) et de concepteurs différents. 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.

Un environnement dynamique

L'équipe est plongée dans un environnement dynamique, donc susceptible d'évoluer pendant le déroulement de la mission. Les véhicules autonomes peuvent être confrontés à des états non initialement prévus dans le plan initial, qui remettent en cause ce dernier. Le superviseur doit donc être capable de réagir face à l'occurrence d'aléas, notamment en permettant à l'équipe d'exécuter un plan alternatif, issu de réparation(s) du plan de mission initial ou d'une replanification complète, pour que les objectifs de mission puissent être atteints.

Des communications restreintes et incertaines

La communication est une composante dynamique particulière de l'environnement. Les communications entre véhicules sont possibles mais contraintes et incertaines et la supervision doit le prendre en compte. D'une part, des modèles pour les communications peuvent être disponibles et permettre de connaître les zones à privilégier ou à éviter pour planifier des communications. Par exemple, dans le scénario II du PEA Action, en plongée, l'AUV Daurade ne peut communiquer avec l'AAV ReSSAC. D'autre part, tout message émis à destination d'un robot distant a une probabilité non nulle de ne pas être reçu, même si le modèle prévoyait l'inverse. La cause peut être d'origines diverses : perte du message par les moyens de communication localement perturbés, configuration inattendue de l'environnement, ou plus simplement, aléa lors du déroulement de la mission qui a provoqué une configuration spatiale des véhicules peu adaptée à une communication. A un instant donné, deux véhicules (ou plus) ne sont donc jamais certains de pouvoir communiquer.

Une mission coopérative

Les véhicules autonomes que nous considérons doivent *coopérer* pour accomplir leur mission (cf. section 1.1, p. 5). Ils disposent de fonctionnalités complémentaires et/ou redondantes et peuvent avoir à réaliser des tâches *conjointement* (deux véhicules ou plus sont impliqués sur une même tâche). Ils ne sont pas intentionnellement adversaires et partagent des objectifs globaux, réalisés de par leurs actions locales.

3.1.2 Hypothèses et spécifications

Dans ces travaux, nous supposons donc que :
- les véhicules employés sont dotés d'une architecture monovéhicule qui leur confère une autonomie décisionnelle *suffisante* pour accomplir les tâches dont ils sont capables individuellement ;
- leur architecture monovéhicule est ouverte à un interfaçage avec un logiciel embarqué externe ;
- les véhicules ont la capacité de communiquer entre eux, et mettent à disposition leurs primitives d'envoi et de réception de messages à un logiciel embarqué externe ;
- nous disposons d'un système de planification embarqué (éventuellement distribué), capable de produire des plans de mission initiaux, mais également de répondre à des requêtes de planification lors du déroulement de la mission.

D'un point de vue abstrait, la couche décisionnelle haut niveau qu'est la supervision doit ainsi répondre à plusieurs besoins. Elle doit être capable :
- de s'interfacer avec les architectures monovéhicules des véhicules, se substituant d'une certaine façon à un rôle habituellement dévolu à un opérateur extérieur chargé de surveiller le bon déroulement de la mission ;
- de contrôler l'exécution du plan de mission (composé de tâches "haut niveau") pour l'équipe de véhicules ;
- de disposer de moyens de détection d'aléas qui impliquent un ou plusieurs véhicules et qui surviennent pendant la mission ;
- d'apporter des réponses d'équipe face à ces aléas impactant possiblement l'équipe entière ;
- de prendre en compte le caractère irrégulier et incertain des communications notamment en gérant l'absence de communication immédiate avec les autres membres de l'équipe, ou en exploitant au mieux les liens de communication existants ;
- de gérer la synchronisation entre les véhicules en cas de tâches réalisées conjointement par deux véhicules autonomes ou plus, mais aussi en cas d'aléas.

3.2 L'architecture HiDDeN

3.2.1 Un système distribué pour la supervision

Les contraintes de communication interdisent de fait une approche centralisée, où une entité unique enverrait les consignes aux véhicules autonomes et attendrait leur retour. L'environnement, tout comme l'équipe de robots, est intrinsèquement dynamique. Comme le souligne Parker [Parker, 2008], une architecture distribuée est une solution naturelle à ce genre de problème : un tel système est par nature distribué en temps, en espace et/ou en terme de fonctionnalités. Un système distribué

est avantageux pour gérer parallélisme (des véhicules peuvent effectuer des tâches simultanément, gain de temps notable pour une mission) et redondance (la robustesse du système est améliorée lorsque différents véhicules sont capables d'effectuer la même tâche : si l'un d'entre eux est défaillant, il peut être remplacé). En revanche, un système non centralisé présente aussi l'inconvénient d'être plus complexe au niveau de la gestion globale. Il est nécessaire de disposer d'une coordination fine des communications, ainsi que de veiller à gérer la cohérence des données du système et les interférences possibles entre les entités qui le composent. En systèmes distribués, une entité ne dispose que d'une connaissance locale de l'état courant du système.

Nous avons ainsi choisi de concevoir une couche 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 décisionnelle respective (figure 3.1).

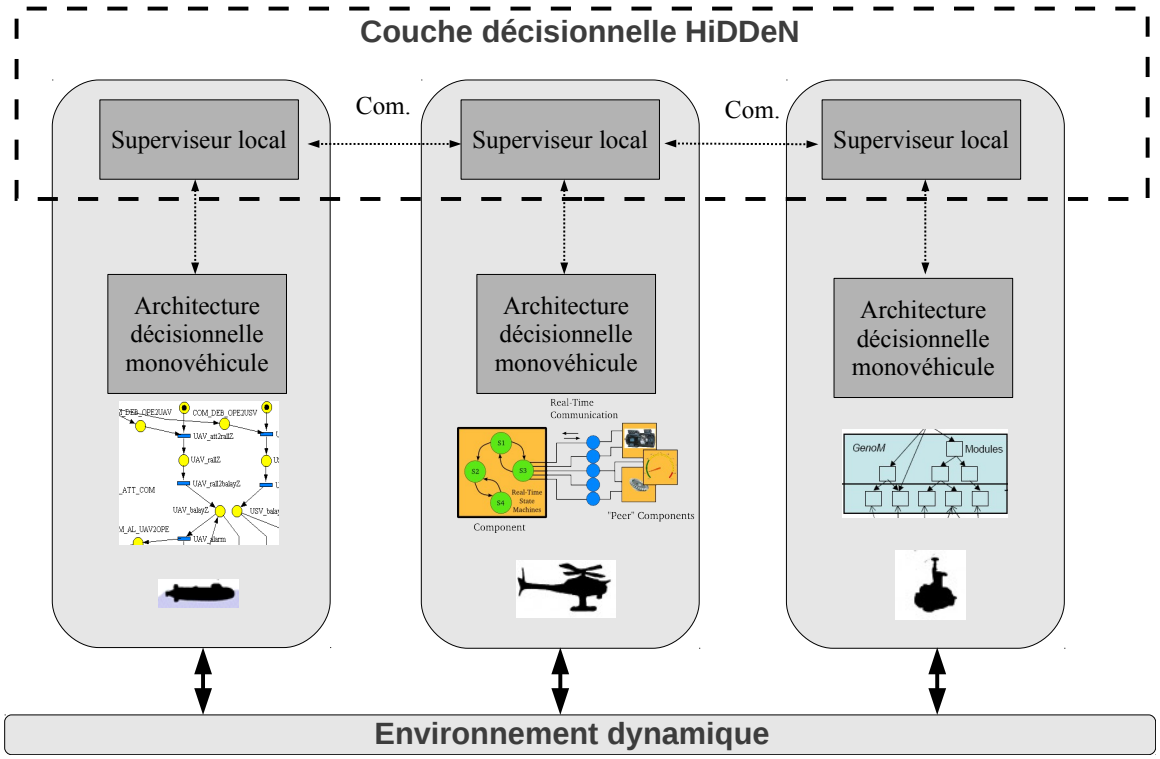


FIGURE 3.1 – L'architecture décisionnelle multivéhicule HiDDeN, supervisant une équipe de véhicules autonomes hétérogènes.

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.

3.2.2 Un système modulaire pour la supervision

La figure 3.2 présente le schéma global de notre architecture du point de vue d'un véhicule autonome. Pour les superviseurs locaux, nous reprenons le côté majoritairement modulaire des architectures de l'état de l'art, qui permettent en général une meilleure généricité. Dans la mesure ou leurs interfaces sont bien définies, le développement de chaque module peut être mené indépendamment.

De plus, la conception de tests unitaires et la ré-utilisabilité de parties de code sont généralement facilitées.

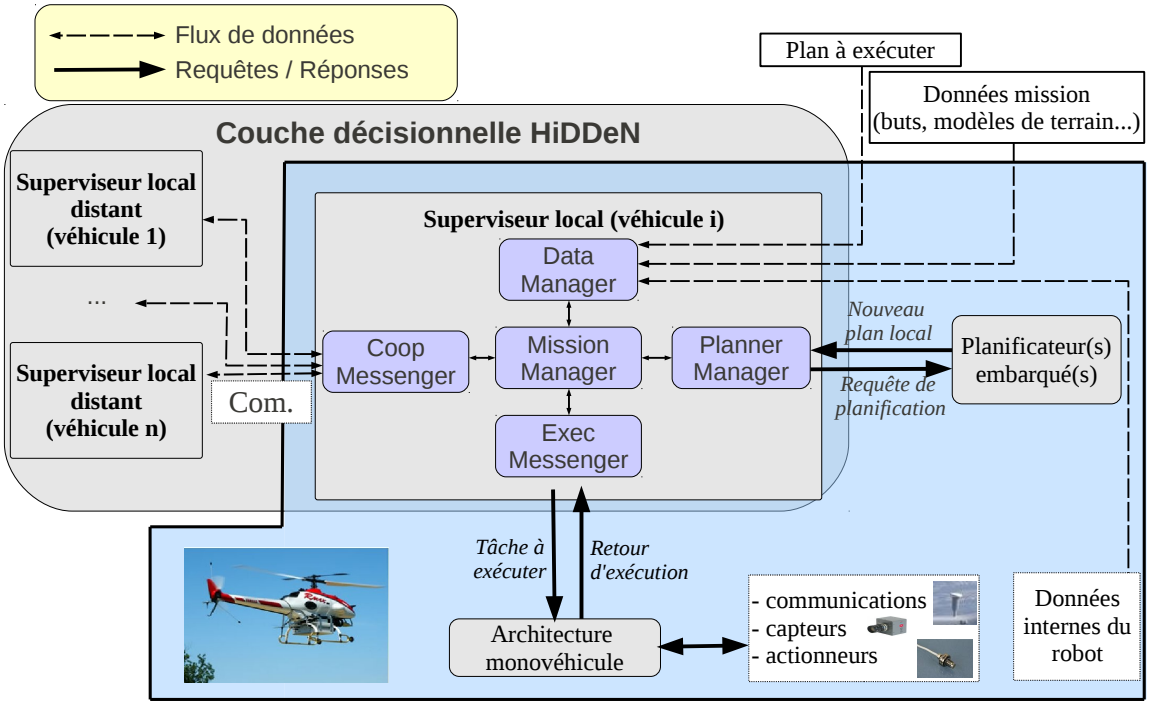


FIGURE 3.2 – Schéma de l'architecture décisionnelle multivéhicule distribuée sur chaque véhicule.

Le superviseur local, destiné à être embarqué à bord du véhicule, est ainsi composé de cinq modules que nous détaillons ci-dessous.

MissionManager : cœur du processus de supervision

Le **MissionManager** constitue l'élément central du superviseur local. C'est l'organe de décision en soi, qui sélectionne, en fonction du plan de mission, de l'état du monde et des objectifs à atteindre, la tâche à accomplir par le véhicule autonome, via son architecture monovéhicule. C'est lui qui est responsable de la coordination des autres modules. Le chapitre 5 (p.67) détaille son fonctionnement dans le cas d'un déroulement nominal de mission, tandis que le chapitre 6 (p.89) traite de la gestion des aléas.

ExecMessenger : interface avec l'architecture monovéhicule

A l'heure actuelle, les architectures monovéhicules des engins autonomes sont en général pourvues d'un protocole de communication permettant à un opérateur humain de dialoguer avec elles, ou du moins d'un système de supervision interne indiquant au véhicule les tâches à accomplir. Chaque architecture monovéhicule dispose de ses propres spécificités. L'**ExecMessenger** s'interface entre le **MissionManager** et l'architecture monovéhicule. Il permet de convertir la tâche à exécuter courante désignée par le **MissionManager** en information intelligible par l'architecture monovéhicule. C'est également lui qui permet la réception et l'interprétation des retours d'exécution de l'architecture

monovéhicule. Ceux-ci peuvent être conformes à un retour espéré, ou renvoyer des comptes rendus d'échec, le déroulement de la tâche n'ayant pas eu lieu correctement, ou un élément étant venu perturber l'exécution ayant été détecté. L'interaction entre **MissionManager** et **ExecMessenger** est décrite au chapitre 4 (p.53).

CoopMessenger : coopération au sein du système distribué

Une hypothèse nécessaire à ces travaux est l'existence d'un lien de communication entre les véhicules, mis à disposition par ces derniers et compatible malgré leur hétérogénéité. Sans lui, il ne peut y avoir de coopération (cf. table 1.1). C'est sur lui que reposent les échanges de message au sein du système de supervision distribué. Cependant, de par les contraintes de communication déjà mentionnées, ce lien peut être temporairement rompu, ou de mauvaise qualité. Le **CoopMessenger** permet de gérer les messages de synchronisation échangés entre les différents superviseurs locaux et indispensables à la cohésion d'équipe. Ce module est comparable au *CoopReactor* de T-Rex [Belbachir et al., 2012], ou à l'*Intercom* de GRAMMPS [Brummit and Stentz, 1998]. En effet, chaque superviseur local doit pouvoir communiquer avec les autres superviseurs locaux, en fonction des moyens de communication mis à sa disposition par le véhicule. Ils peuvent donc s'échanger des informations utiles pendant la phase d'exécution. Si plusieurs moyens de communication sont disponibles (modem acoustique et Wifi pour l'AUV Daurade par exemple), c'est à lui de sélectionner le plus approprié (ou plus simplement, le protocole qui a été implémenté). Nous verrons dans le chapitre 5 (p.67) comment sont générées des tâches de synchronisation destinées à l'émission de ces messages.

DataManager : gestion des connaissances pour la supervision

Pour exécuter sa mission coopérative, l'équipe de véhicules autonomes hétérogènes doit détenir une base de connaissances comprenant des informations sur les objectifs de mission, et les moyens dont les véhicules disposent pour les atteindre. Il peut aussi s'avérer utile pour un organe de supervision de posséder une représentation de l'état courant du monde, même partiel, qui puisse être mise à jour tout au long du déroulement de la mission, notamment dans une optique de replanification en ligne. Le **DataManager** repose sur une base de connaissances qui est décrite au chapitre 4 (p.53). Nous verrons alors que le fonctionnement de l'ensemble des modules de HiDDeN repose sur son existence.

PlannerManager : interface avec le système de planification

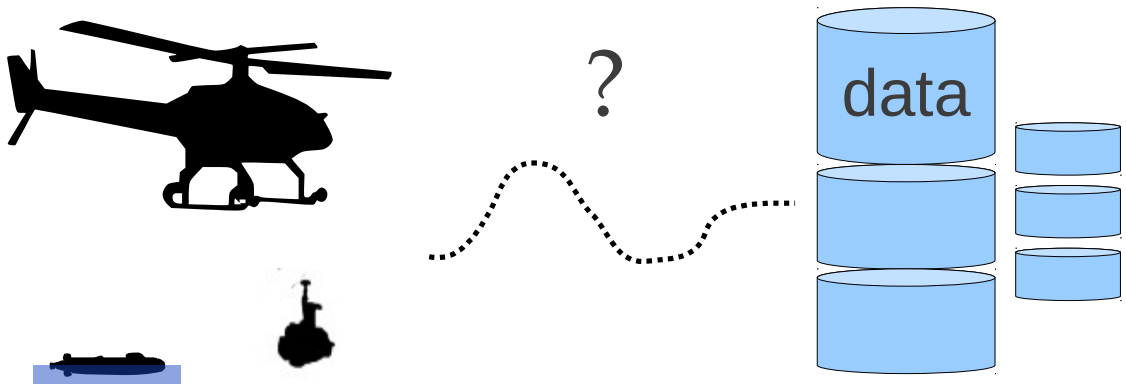
La présence d'au moins un planificateur embarqué est nécessaire pour effectuer des réparations ou des replanifications en ligne durant le déroulement de la mission. Des requêtes de planification peuvent lui être envoyées, et il doit en retour renvoyer des plans ou parties de plan associés à ces requêtes. C'est au **PlannerManager** de gérer l'interaction avec le ou les planificateurs embarqués. Il sélectionne le plus approprié suivant l'aléa rencontré et permet de mettre à jour le plan courant de la mission. Les réparations et les mises à jour s'effectuent sous le contrôle du **MissionManager** qui veille à la cohérence des données au sein de l'équipe via **DataManager**. Ces réparations sont abordées au chapitre 6 (p.89).

3.3 Synthèse

Finalement, HiDDeN semble répondre convenablement aux attentes que nous avons spécifiées à la section 3.1.2. L'**ExecMessenger** fournit l'interface nécessaire pour communiquer avec l'architecture monovéhicule. Dans un souci de généricité, le développement d'un tel module requiert tout de même un travail de formalisation à mener vis-à-vis des architectures monovéhicules, afin d'éviter de devoir développer un code *ad-hoc* : à partir d'une description d'un véhicule autonome, il serait intéressant de pouvoir extraire automatiquement les primitives nécessaires à cet interfaçage, pour éviter d'avoir à adapter le code de l'**ExecMessenger** à chaque nouvelle architecture rencontrée. Nous décrivons dans le chapitre suivant (chapitre 4, p.53) comment nous formalisons cette description, et quelle est la portée de son utilité. Dans le chapitre 5 (p.67), nous explicitons le contrôle d'exécution du plan de mission par HiDDeN à travers les procédures du **MissionManager**. Dans ce même chapitre, nous proposons des moyens concrets de détection d'aléas dont le module a également la charge. Le chapitre 6 (p.89) est quant à lui dédié aux réponses à apporter face aux aléas rencontrés, notamment grâce au **PlannerManager**, responsable des planifications. Le caractère incertain et irrégulier des communications ainsi que la synchronisation de l'équipe font l'objet d'une attention particulière tout au long de ces chapitres, et sont pris en compte par une association entre le **MissionManager** et le **CoopMessenger**. Enfin, les derniers chapitres sont consacrés à diverses évaluations et aux expérimentations de l'architecture HiDDeN.

CHAPITRE 4

Formalisation des connaissances



Ce chapitre est consacré à Koper, base de connaissances pour la planification, l'exécution et la replanification de mission pour une équipe de véhicules hétérogènes. Nous avons choisi l'ontologie comme modèle de connaissance. Grâce à elle, nous disposons d'un outil qui peut fournir une description formelle des véhicules de notre équipe, et en particulier de leurs interfaces, point crucial pour l'automatisation des transferts d'informations entre HiDDeN et les architectures monovéhicules. Nous la détaillons dans un premier temps, puis nous explicitons l'usage potentiel qui peut en être fait, que ce soit au niveau des aspects planification ou à celui de la gestion des connaissances de l'équipe.

Sommaire

4.1 Une ontologie pour formaliser les connaissances

4.1.1 Spécification des besoins propres à la supervision

Dans l'architecture HiDDeN, le module *ExecMessenger* s'interface entre le module *MissionManager* (l'organe de décision) et l'architecture monovéhicule. Chaque requête en provenance du *MissionManager* doit donc être formatée dans un langage interprétable par l'architecture monovéhicule. Par exemple, si le superviseur local souhaite qu'un véhicule exécute un "goto" vers un waypoint "wptA", comment adresser cette requête à l'architecture monovéhicule qui contrôle le robot ? Comment savoir si le véhicule est seulement capable d'accomplir cette action ? La procédure d'exécution d'une action portant le même nom dans un plan d'équipe peut différer suivant les véhicules, surtout si ceux-ci sont explicitement hétérogènes. Et il en est de même pour le type des données utilisées : comment la variable "wptA" doit-elle être interprétée ? On peut difficilement supposer que le robot dispose d'une table de correspondance entre nom de variable abstraite, et valeur de variable à utiliser. Il faut donc pouvoir transmettre la valeur de cette variable dans un langage que chaque architecture monovéhicule pourra interpréter. Un AAV pourrait par exemple exiger une représentation en trois dimensions, sous la forme de coordonnées GPS, tandis qu'un AGV utiliserait une représentation en 2D, avec deux coordonnées de distance par rapport à un point de référence.

Pour que l'*ExecMessenger* du superviseur local et les architectures monovéhicules puissent dialoguer sans devoir systématiquement développer une interface *ad-hoc* à chaque nouvelle architecture rencontrée, nous avons besoin d'une description formelle des services fournis par l'architecture monovéhicule. C'est dans cette optique que nous avons choisi de développer une base de connaissances pour la supervision. L'objectif est de faire en sorte que si les services d'une architecture monovéhicule sont décrits dans cette base de connaissances, le véhicule équipé de cette architecture puisse directement s'insérer au sein de l'équipe pour en devenir un membre à part entière.

Une description formelle des tâches qu'un véhicule est capable d'accomplir, ainsi que les protocoles permettant de dialoguer avec lui pour les lui faire exécuter, aurait un intérêt certain en terme de génération de code automatique : la couche décisionnelle de l'équipe saurait quel robot est capable d'accomplir quelle tâche, et pourrait lui donner l'instruction de la réaliser. Un véhicule autonome est généralement doté d'un ensemble de connaissances (position, état de ses capteurs, modèles d'environnement...) Disposer d'une description formelle de celles-ci pourrait également être bénéfique pour l'équipe en terme d'automatisation, puisque le superviseur pourrait s'abstraire du type spécifique des données utilisées par chacun de ses membres.

Nous partons de l'hypothèse qu'une telle base de connaissances serait définie hors ligne, et son contenu amené à évoluer lors de la phase d'exécution, de par la nature dynamique de l'environnement et de par les actions des véhicules autonomes hétérogènes. Elle permettrait de s'abstraire du caractère hétérogène des véhicules composant l'équipe, qui, disposant alors d'un langage commun, verrait leur interopérabilité accrue.

4.1.2 Choix d'une ontologie

Si on regarde l'état de l'art, la problématique de formalisation semble récurrente dans la supervision de mission. Elle semble sous-entendue pour l'architecture modulaire de [Vidal et al., 2002], ou lors des échanges entre services pour [Doherty et al., 2009]. Des auteurs comme [Dix et al., 2003] en soulignent la présence, notamment au niveau du décalage qui existe entre la planification d'une mission et son exécution. Elle pourrait s'avérer nécessaire dans une implémentation en situation réelle du système proposé par [Bonnet-Torrès and Tessier, 2005].

Bon nombre d'outils et de procédés existent dans la littérature pour représenter et organiser des connaissances (taxonomie, thésaurus, table de base de données...) Notre choix s'est porté sur une formalisation sous forme d'ontologie, sur laquelle se basera notre modèle de connaissances. Une ontologie est une forme de représentation classique des connaissances de domaines particuliers, qui autorise différentes personnes à dialoguer entre elles pour y définir et partager des *concepts*, [Baclawski and Simeqi, 2001]. Pour ce faire, des relations riches sont introduites entre ces concepts.

Dans les domaines de la robotique, les ontologies sont régulièrement employées pour permettre à une communauté de s'accorder sur des spécifications et la conceptualisation de connaissances. Une description formelle est alors utilisée, présentant l'intérêt d'être commune au groupe de travail, directement interprétable par une machine [Sellami et al., 2011], et procurant la possibilité d'y inférer de nouvelles informations [Schlenoff and Messina, 2005]. En SMA, les ontologies peuvent notamment être utilisées pour le partage d'informations entre agents, même si elles ne donnent pas de garantie sur la compréhension de leur contenu [Deplanques et al., 1996]. Cependant, il faut veiller à bien distinguer ontologie (la spécification) et base de connaissances (qui contient concrètement les données) [Lortal et al., 2011]. Même si les ontologies sont vouées à être génériques et réutilisables, elles restent en pratique très liées au point de vue de leurs auteurs [Deplanques et al., 1996].

4.1.3 Réutiliser une ontologie existante ?

Divers projets utilisent déjà des ontologies. Dans le domaine des *Web Services*, un système développé par Evren Sirin [Sirin et al., 2003] permet l'exécution d'un plan composé de requêtes vers des Web Services. Ce plan est obtenu via un planificateur HTN [Nau et al., 2003]. Des informations en provenance de Web Services sont d'ailleurs utilisées pendant la phase de planification. Il est à noter que les auteurs fournissent un algorithme robuste et complet qui permet de transcrire les descriptions de services écrites sous OWL-S [1] en un domaine adapté à SHOP2.

Le projet européen Robot Earth [Tenorth et al., 2012] vise à construire une base de données, avec laquelle des robots du monde entier pourraient partager des informations concernant leurs expériences, tout en s'abstrayant de leurs spécifications matérielles. Le projet est récent et s'oriente plus sur la représentation et sur le partage de la connaissance de l'environnement que sur des thèmes de coopération de robots.

Dans le projet Proteus [Lortal et al., 2011], une ontologie de plus de quatre-cent classes est utilisée pour effectuer un transfert de connaissances scientifiques entre différentes communautés robotiques. Cependant, l'ontologie développée ne peut être directement exploitée, notamment dans le cadre de génération automatique de code : les auteurs doivent passer par une phase de transformation semi-automatique de l'ontologie en une représentation UML (Unified Modeling Language). L'ontologie utilisée est également spécifique à leurs scénarios et problématiques.

Dans le domaine spatial, le projet NASA SWAMO [Witt et al., 2008] se sert d'une ontologie pour définir un standard entre les interfaces qui permettent d'accéder aux différentes ressources de la mission (capteurs, capacités des agents...) De façon similaire, l'ontologie A3ME [Herzog et al., 2008] donne une description d'un ensemble d'appareils hétérogènes afin de disposer d'une interopérabilité en terme de communication. D'autres auteurs [Schlenoff and Messina, 2005] ont également travaillé sur la représentation des capacités des robots, en particulier dans un contexte de missions de recherche et sauvetage.

L'intérêt de telles approches est indéniable et pourraient constituer un bon point de départ à notre travail. Cependant, les ontologies proposées ici sont maintenues à un niveau de représentation

1. http://www.w3.org/Submission/OWL-S/

des connaissances "bas niveau". Les auteurs se concentrent plus sur la description des capacités d'agents mobiles très concrètes (comment vont s'enclencher tels capteurs, quel ampérage est nécessaire au fonctionnement nominal...) que sur la description haut niveau des services des agents autonomes dont nous désirons disposer. Certaines ontologies citées sont très riches, mais en contrepartie difficiles à exploiter en terme de génération automatique de code. Pour ces raisons, nous avons décidé de définir notre propre ontologie.

4.2 La base de connaissances Koper

4.2.1 Description de l'ontologie

Les connaissances que nous souhaitons modéliser dans l'ontologie correspondent à une description des véhicules impliqués dans la mission, les informations qu'ils sont susceptibles de fournir et de recevoir, ainsi que leurs capacités que nous qualifierons de *services*. L'intérêt est d'avoir d'une part la description de la façon dont les services d'un véhicule autonome sont appelés, et d'autre part, le type de retour qu'ils peuvent renvoyer. En effet, cela permettrait d'automatiser l'interfaçage avec toute architecture monovéhicule.

Différents langages sont utilisés pour décrire des ontologies dans des formats exploitables par une machine. Notre choix s'est naturellement porté sur celui qui semble être actuellement le plus employé en IA : OWL (Web Ontology Language), défini par le W3C [2]. OWL est un langage basé XML [3] spécialement développé pour décrire des ontologies en logique descriptive (ou DL, Description Logic). Des outils variés ont été conçus pour manipuler ce langage, dont Protégé [4], avec lequel nous avons décrit notre ontologie (figure 4.1). C'est sur les spécifications de l'ontologie définie que nous avons conçu Koper (**Kno**wledge base for **p**lanning, **e**xecution and **r**epair), la base de connaissances que nous utiliserons.

Robot et services

Une façon classique de conceptualiser un robot consiste à définir les services qu'il peut fournir, ainsi que la manière selon laquelle des agents extérieurs peuvent s'interfacer avec eux [Brugali and Scandurra, 2009]. De nombreuses ontologies utilisent déjà le concept de service dans le domaine des *Web Services*, avec notamment OWL-S [5] , ou WSDL [6] (Web Services Description Language). Nous nous en sommes également inspiré, même si nous nous appuyons surtout sur le paradigme de la programmation par contrat vis-à-vis des services qu'un véhicule autonome peut fournir [Brugali and Scandurra, 2009].

Un `Service` est une tâche ou une activité qu'un robot (ou un agent logiciel) peut accomplir dans l'environnement dans lequel il est plongé (y compris ce que les capacités de ses capteurs et calculs internes peuvent fournir, par exemple de la détection de cible, une analyse d'échantillon rocheux, etc.) Dans notre représentation (figure 4.1), un service doit respecter un `contract` (`SerContract`). Un ensemble de paramètres d'entrée peuvent lui être nécessaires pour pouvoir

2. World Wide Web Consortium, `http://www.w3.org/standards/techs/owl`.
3. eXtensible Markup Language `http://www.w3.org/standards/techs/xml#w3c_all`.
4. `http://protege.stanford.edu/`
5. `http://www.daml.org/services/owl-s/` et `http://www.w3.org/Submission/OWL-S/`
6. `http://www.w3.org/TR/wsdl`

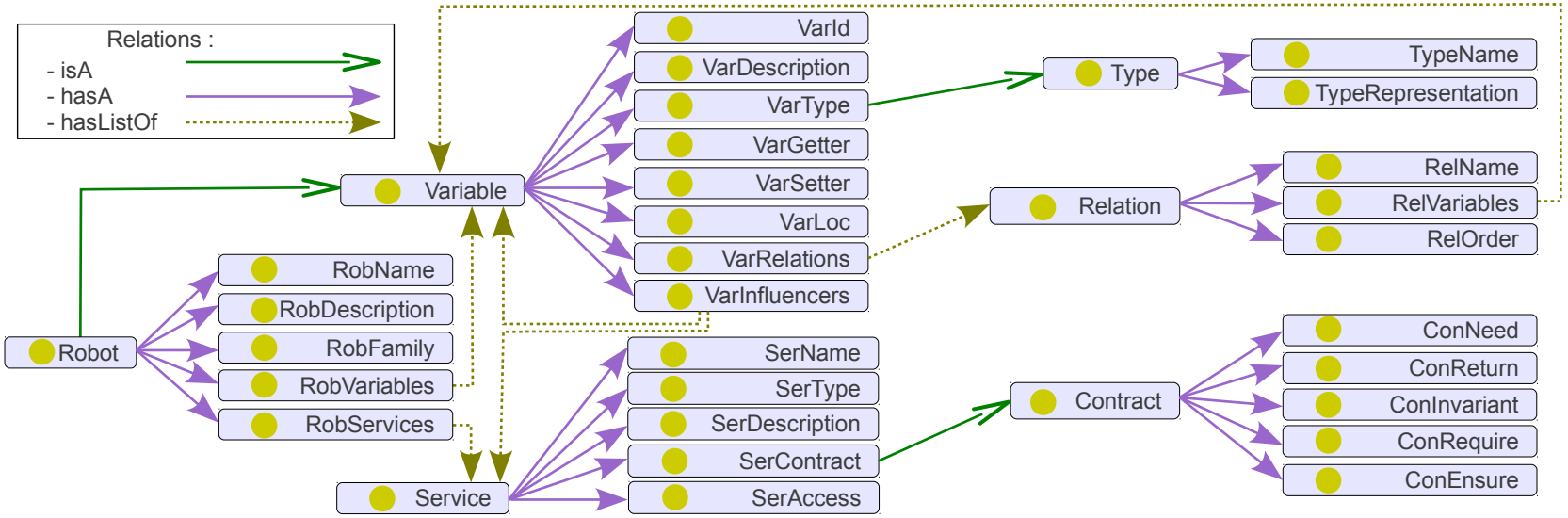


FIGURE 4.1 – Représentation simplifiée de l'ontologie définie. Schéma librement adapté du visualiseur de Protégé, Ontograf.

l'utiliser (ConNeed), et il peut renvoyer un ensemble de paramètres en sortie (ConReturn). Il peut être associé à certaines spécifications, telles que des préconditions (ConRequire), des post-conditions (ConEnsure) ou des invariants (ConInvariant). Cette description est suffisante dans le cadre de nos travaux, même si ces trois derniers concepts pourraient être décrits plus en profondeur, par exemple en séparant les préconditions et les postconditions qui dépendent du contexte de celles qui n'en dépendent pas [Huang et al., 2009].

Un Service est défini par son nom (SerName) et une description (SerDescription). Son interface avec le monde extérieur, SerAccess, permet de décrire comment une entité distante (un autre robot, un superviseur d'équipe...) peut faire appel à ce service pour que le robot puisse l'exécuter.

Nous distinguons les champs de Service en différents types (SerType). Lorsqu'ils sont *synchrones*, ils peuvent être directement appelés (action de mouvement, prise d'image, calcul de données, etc.) et un résultat d'exécution est généralement attendu. Sinon, lorsqu'ils sont *asynchrones*, nous les qualifierons de services *dæmons* : ils tournent en "tâche de fond" et peuvent renvoyer à tout moment des événements particuliers (détection d'une cible, détection d'un lien de communication rompu...) Il est à noter qu'ils peuvent demander une activation particulière, cette dernière pouvant correspondre à une tâche synchrone "interrupteur" (par exemple, "enclenchement de la détection de cible", avec le retour "ok, détection de cible enclenchée").

Variables internes des robots

Une description de la connaissance manipulée par le robot semble indispensable : cela se fait grâce au concept de Variable. Par exemple, si un robot dispose d'un GPS, il peut être intéressant de disposer d'une variable $RobotPositionGPS$ qui fournisse sa position. Dans notre ontologie, une variable est classiquement reliée aux concepts d'identifiant (VarId), de type (VarType) et optionnellement d'une description textuelle informelle (VarDescription).

Les concepts de VarGetter et de VarSetter sont comparables aux accesseurs classiques du paradigme de la POO (Programmation Orientée Objet). C'est un moyen d'accéder aux valeurs de la variable, et un moyen d'en modifier la valeur. Mais à l'instar de la POO, ceux-ci ne sont pas toujours disponibles. La localisation (VarLoc) de la variable (l'endroit où elle est stockée) est indirectement liée aux entités (véhicules distants, opérateurs ou autres variables...) pouvant avoir une influence sur cette variable (VarInfluencers). En effet, lors de la phase d'exécution, il peut être intéressant d'observer les éléments susceptibles de faire évoluer la valeur d'une variable.

Par exemple, un robot $r1$ peut contrôler sa propre position. Un autre robot $r2$ peut disposer d'un accès vers cette valeur, mais ne pourra la modifier... sauf dans le cas où $r2$ est capable de déplacer $r1$, ou plus simplement, capable de le relocaliser avec une meilleure précision par exemple.

Le concept de Relation, visible sur la figure 4.1, sera explicité dans la section 4.3.2. Il enrichie l'ontologie originale sur laquelle est basée Koper.

4.2.2 Exemple d'instanciation pour Koper

Plusieurs versions de Koper ont été instanciées pour répondre aux besoins des différents scénarios du PEA Action. Voici un exemple de description dans Koper d'un véhicule AAV ReSSAC : le listing 4.1 présente des exemples de services et de variables internes que l'architecture mono-véhicule propose. L'AAV ReSSAC est ainsi capable d'effectuer des déplacements, décoller, atterrir, localiser avec précision une cible dans son champ de vision, la suivre, envoyer sa position, balayer une zone, se ravitailler et enfin faire un signe à un bateau. On notera la présence d'une tâche dæmon,

"TargetDetectionOn" qui modélise le fait que l'architecture monovéhicule est en permanence à la recherche de cibles. Le véhicule dispose de sa position (GPS), de son état (au sol, en déplacement, stationnaire...), de la quantité de carburant qui lui reste, d'une liste de waypoints autorisés et d'un modèle de sa consommation pour chacun de ses services.

Un exemple détaillé du premier service, le service *goto*, est fourni dans le listing 4.2. Les listings sont simplifiés pour alléger les écritures.

```
<Robot>
 <RobName>aav1</RobName>
 <RobDescription>L'AAV ReSSAC de l'Onera </RobDescription>
 <RobType>AAV</RobType>
 <RobServices>
  <Service><SerName>goto</SerName>...</Service>
  <Service><SerName>takeOff</SerName>...</Service>
  <Service><SerName>land</SerName>...</Service>
  <Service><SerName>localizeTarget</SerName>...</Service>
  <Service><SerName>track</SerName>...</Service>
  <Service><SerName>getGPSPoint</SerName>...</Service>
  <Service><SerName>carto</SerName>...</Service>
  <Service><SerName>balayage</SerName>...</Service>
  <Service><SerName>repBalayage</SerName>...</Service>
  <Service><SerName>refuel</SerName>...</Service>
  <Service><SerName>signeBoat</SerName>...</Service>
  <Service><SerName>stationnaireCOM</SerName>...</Service>
  <Service><SerName>TargetDetectionOn</SerName>...</Service>
 </RobServices>
 <RobVariables>
  <Variable><VarId>pos</VarId>...</Var>
  <Variable><VarId>state</VarId>...</Variable>
  <Variable><VarId>fuel</VarId>...</Variable>
  <Variable><VarId>listWpts</VarId>...</Variable>
  <Variable><VarId>modelActionDuration</VarId>...</Variable>
 </RobVariables>
 </RobVariables>
</Robot>
```

Listing 4.1 – Exemple de description de l'AAV ReSSAC (extrait d'une instance destinée au scénario II du PEA Action).

4.3 Utilisations de Koper

4.3.1 Exploitation de Koper par l'*ExecMessenger*

Exécuter un plan de mission consiste à faire exécuter chacune des actions décrites dans le plan en suivant un ordre précis par les robots concernés. Les véhicules autonomes disposent de protocoles spécifiques qui décrivent la façon dont un agent distant peut interagir avec eux. Les actions du plan doivent donc être transcrites suivant ce protocole pour être interprétées et réalisées par les véhicules autonomes. Koper permet d'automatiser ce processus. Chaque fois qu'un *ExecMessenger* envoie une action vers un véhicule autonome, le format de celle-ci est transformé pour respecter ce protocole de communication, en particulier grâce au champ `SerAccess`. En effet, ce dernier précise le type de communication (mémoire partagée, socket, ...), la configuration associée (par

```
<Service>
 <SerName>goto</SerName>
 <SerType>DirectTask</SerType>
 <SerDescription>
   The vehicle moves to the ?wptTo location
 </SerDescription>
 <SerContract>
  <ConNeed>
   <Arg>
    <ArgName>?wptFrom</ArgName>
    <ArgType>Waypoint</ArgType>
   </Arg>
   <Arg>
    <ArgName>?wptTo</ArgName>
    <ArgType>Waypoint</ArgType>
   </Arg>
  </ConNeed>
  <ConReturn>
   <Arg>
    <ArgName>?msgReturn</ArgName>
    <ArgType>SimpleReturn</ArgType>
   </Arg>
  </ConReturn>
  <ConRequire>
    <Cond>(this.pos == ?wptFrom)</Cond>
    <Cond>(this.state != this.onGround)</Cond>
  </ConRequire>
  <ConEnsure>
    <Cond>(this.pos == ?wptTo)</Cond>
    <Cond>(this.pos != ?wptFrom)</Cond>
  </ConEnsure>
  <ConInvariant>
   <Cond>(this.enoughFuel)</Cond>
  </ConInvariant>
 </SerContract>
 <SerAccess
  ComBase="Socket"
  OutPort="60100"
  ParamUsed="?wptTo"
  OutMsgFormat="SocketMsg('GOTO'+Waypoint)"
  InPort="60101"
  InMsgFormat="SocketMsg(SimpleReturn)"/>
</Service>
```

Listing 4.2 – Exemple du contrat d'un service "goto" d'un AAV, représenté sous format XML ; le mot-clé *this* fait référence au propriétaire du service.

exemple, adresses et ports pour le cas des sockets) ainsi que le format du contenu qui doit être envoyé. Quand un robot exécute une action, un retour qui précise si la tâche a pu être accomplie correctement ou non est généralement attendu. Une fois de plus, la forme et la manière de ce retour sont décrites dans Koper, à travers les champs `ConReturn` et `SerAccess`.

Par exemple, prenons une action "goto wpt1 wpt2 aav1" d'un plan fourni par un planificateur PDDL classique : l'AAV *aav1* doit aller du waypoint *wpt1* au waypoint *wpt2*. L'action est transmise du *MissionManager* vers l'*ExecMessenger*. L'*ExecMessenger* cherche dans Koper le service

associé au `SerName` "goto" (listing 4.2). Il est informé que la communication avec l'architecture monovéhicule est basée sur des sockets (ligne 37 du listing 4.2), via un port précis, et également du format du message sous lequel envoyer l'ordre au robot. Le paramètre "Waypoint" (du champ `ConNeed`) est envoyé au robot, en ayant pris le soin de récupérer au préalable sa valeur à travers une fonction `VarGetter`. Finalement, le processus d'exécution de plan est capable d'envoyer un message par socket, formaté en un paquet intelligible par l'architecture monovéhicule du robot. Une fois le message envoyé, l'*ExecMessenger* se met en attente de réception d'un message sur un autre socket de type "SimpleReturn", qui correspond à un compte rendu sous forme de booléen (`true` si le point $wpt2$ est atteint, `false` dans tous les autres cas).

Koper répond convenablement à notre besoin de formalisation des interfaces des véhicules. L'*ExecMessenger* dispose de toutes les informations requises pour dialoguer avec l'architecture locale : les protocoles d'échange mais aussi les correspondances entre actions et services, les correspondances entre variables abstraite de la couche décisionnelle et variables concrétement exploitable par l'architecture monovéhicule et enfin les différents types utilisés entre toutes les entités. En l'enrichissant quelque peu, nous avons également réussi à en tirer parti pour d'autres besoins de la mission, notamment dans le cadre :

1. de la génération de modèles de planification hors ligne lors de la phase d'exécution ;
2. du maintien de la cohérence des données entre les superviseurs locaux ;
3. de la détection de fautes lors du déroulement des actions.

Ces trois capacités sont décrites dans les paragraphes suivants.

4.3.2 Génération de modèles de planification

La génération d'un modèle de planification issu d'une base de connaissances externe et sans intervention d'un expert humain pour une mission réelle est loin d'être évidente. Sauf si la mission est déjà entièrement décrite dans un langage de planification. En revanche, fournir une correspondance entre deux modèles de données lorsqu'ils sont formellement décrits est plus aisé. C'est ce que nous proposons de faire en nous basant sur le modèle formel sur lequel repose Koper pour effectuer de la génération automatique de code pour le modèle de planification. Cette approche présente aussi l'avantage d'être généralisable à tout langage de planification, dans la mesure où celui-ci serait suffisamment riche pour exprimer ce que l'on souhaite modéliser.

A partir de l'ontologie précédente, il est aisé d'ajouter à la description des services des robots et de leurs variables internes une description de l'environnement et une description des objectifs de la mission. Dès lors, on dispose d'une base d'information suffisante pour écrire un modèle de planification. En effet, les variables d'environnement (zones, objets d'intérêt, ...) et les objectifs de missions peuvent être modélisés en s'aidant du champ `variable`. De plus, nous exploitons un type de variable particulier, les variables de type $Model$ à qui sont associés des modèles robotiques. Cela permet de représenter des données issues de modèles externes, telles que des cartes de traversabilité, des estimations de la disponibilité des communications, les modèles de consommation en carburant des robots... et qu'un planificateur exploite généralement.

Le nouveau concept de `Relation` (figure 4.1) permet quant à lui d'interconnecter des variables entre elles, et de leur ajouter des caractéristiques (par exemple, une zone $z1$ peut être sous-zone de la zone $z2$), sans avoir besoin de créer de nouveaux types pour ces variables.

Pour illustrer cette génération, nous utilisons ici le langage PDDL (Planning Domain Definition Language). En PPDL, un modèle de planification est composé d'un domaine (qui décrit les actions

utilisables) et d'un problème (qui contient les instances des variables du domaine à l'état initial, et l'état but à atteindre).

Domaine PDDL : Un domaine PDDL est constitué d'un ensemble d'$actions$. Une action PDDL est décrite par son nom, des paramètres, une précondition et un effet. Il est possible d'employer des transformations de modèle à modèle (ou M2M - Model to Model) pour générer une action PDDL à partir de chaque `Service` de Koper. `SerName` correspond au nom de l'action PDDL. `ConNeed` fournit les informations relatives aux paramètres de l'action PDDL. `ConRequire` correspond aux préconditions de l'action, alors que `ConEnsures` et `ConInvariant` précisent les effets de l'action. Certains champs `Variable` de Koper peuvent intervenir dans la génération du domaine, étant donné qu'elles peuvent être "influencées" par des services.

L'action PDDL qui correspond au service *goto* précédent (décrit dans le listing 4.2) apparaît dans le listing 4.3, dans lequel la variable $?v$ correspond à l'AAV. La variable $?duration$ provient de la `Variable` d'identifiant *modelActionDuration* du listing 4.1.

```
(:action goto
 :parameters (?wptFrom ?wptTo - Waypoint  ?v - Vehicle)
 :duration (= ?duration (function_gotoDuration ?wptFrom ?wptTo ?v))
 :condition ((enoughFuel ?v)(atPosition ?wptFrom ?v)(not (onGround ?v)))
 :effect (and((not(atPosition ?wptFrom ?v))(atPosition ?wptTo ?v))))
```

Listing 4.3 – Une action "goto" en PDDL.

Problème PDDL : Un problème PDDL est un ensemble d'objets, un état initial et un état but. Nous utilisons à nouveau une transformation de modèle à modèle pour générer le problème PDDL, en s'appuyant sur les instances de `Variable` et de `Relation`. L'ensemble des objets PDDL proviennent du `VarId` et du `VarType` de chaque `Variable`. La description de l'état initial nécessite une couverture exhaustive des valeurs des instances de `Variable`, accessibles via les informations issues de `VarGetter` et de `VarLoc`. La valeur de planification de la `Variable` est formatée suivant la documentation de sa `VarType`. Les entités de `Relation` sont ensuite interprétées, puisque des variables déjà écrites peuvent être en inter-dépendantes. Enfin, le but est généré similairement, étant donné que dans Koper, il est représenté également par une `Variable`.

Le listing 4.4 montre la définition de variables dans Koper et de quelques relations, et le listing 4.5 le problème PDDL associé.

4.3.3 Regénération de modèles et maintient de la cohérence

L'approche proposée avec Koper autorise aussi une regénération de ces modèles pendant la phase d'exécution, lorsque l'ensemble de connaissances a évolué et que l'on souhaite envoyer de nouvelles requêtes de planification. Générer un modèle de planification en ligne suppose que les données sources soient valides. Or dans les systèmes distribués, la connaissance exacte de l'état global du système à un instant donné n'est pas possible. Koper apporte des éléments digne d'intérêt en ce qui concerne la mise à jour de variables distantes et la récupération de leurs valeurs. En effet, le champ `VarInfluencers` permet de spécifier par qui une `Variable` peut être "influencée" et donc mise à jour. Par exemple, la position d'un robot r évoluera dès lors que des appels à des services de mouvement vers ce robot seront effectués. La localisation physique de la variable `VarLoc`

```
<Variable>
 <VarId>aav1Position</VarId>
 <VarType>Waypoint</VarType>
 <VarGetter>Service::getGPSPoint</VarGetter>
 <VarSetter>none</VarSetter>
 <VarLoc>local</VarLoc>
 <RelName>aav1PositionRelation</RelName>
 <VarInfluencers>
  <SerName>aav1::goto</SerName>
 </VarInfluencers>
</Variable>
<Variable><VarId>wpt1</VarId>...</Variable>
<Variable><VarId>wpt2</VarId>...</Variable>
<Variable><VarId>goalVariable</VarId>...</Variable>
<Relation>
 <RelName>aav1PositionRelation</RelName>
 <RelVariables>
  <VarId>aav1</VarId>
  <VarId>aav1Position</VarId>
 </RelVariables>
 <RelOrder>simple</RelOrder>
</Relation>
```

Listing 4.4 – Exemple d'une représentation en XML de variables et de relations pour la description de l'environnement.

```
(:objects
 aav1 - Vehicle
 aav1Position - Waypoint
 wpt1 - Waypoint
 wpt2 - Waypoint)
(:init
 (isAAV aav1)
 (atPosition aav1Position aav1)
 (onFlight aav1)
 (enoughFuel aav1))
(:goal
 (atPosition wpt2 aav1))
```

Listing 4.5 – Vue partielle d'un problème PDDL, généré à partir des listings 1.1, 1.2 et 1.4 : $aav1Position$ est décrite dans le listing 1.4. Les variables $wpt1$ et $wpt2$ proviennent d'une description similaire, tout comme la variable *goalVariable*, qui précise que $aav1$ doit atteindre le waypoint $wpt2$; $aav1$ est le véhicule instancié dans Koper décrit par le listing 1.1 ; une relation existe entre $aav1$ et $aav1Position$ (le champ `RelVariables` du listing 1.4) qui conduit à la génération de la deuxième ligne de l'état initial.

est aussi un aspect exploitable. Si celle-ci doit être mise à jour, le *MissionManager* sait où elle se trouve. Sa valeur peut être ensuite récupérée *via* les champs `VarGetter`. Ceci est exploité lors de la synchronisation des bases de connaissances entre superviseurs locaux, notamment lors des phases de réparation de plan que nous aborderons au chapitre 6 (p.89).

Lorsqu'un superviseur local est en communication avec tous superviseurs locaux distants, il peut donc avoir une bonne estimation de l'état global du système, modulo le temps de transfert des

données. Si la communication est rompue, il peut tout de même disposer de l'état des superviseurs locaux au moment de la dernière communication.

4.3.4 Détection d'aléas à l'exécution

Le contrat du service (`SerContract`) permet d'effectuer de détecter des aléas lors de la phase d'exécution. Le *MissionManager* dispose ainsi de moyens pour tester que : (a) les préconditions de l'action (`ConRequire`) sont satisfaites avant de lancer l'exécution, (b) les effets de l'action sont cohérents par rapport au modèle de l'action (`ConEnsure`) et (c) les invariants de l'action restent valides durant le déroulement de l'action. Nous reviendrons sur la détection des aléas en fin de chapitre 5 (section 5.5, p.87).

4.4 Conclusion

Finalement, grâce à cette ontologie, nous disposons d'une description de ce que les véhicules sont capables d'accomplir, de la manière dont il est possible de dialoguer avec eux, de l'environnement dans lequel ils sont plongés et des objectifs qu'ils doivent atteindre. En fait, cette ontologie est capable de regrouper à la fois des informations qui concernent les aspects décisionnels (comme la planification), et des informations plus spécifiques à la phase d'exécution.

La connaissance manipulée par les véhicules est directement impliquée dans le processus de planification, mais sous une forme en général plus abstraite. Comme nous l'avons déjà noté dans l'état de l'art, il existe un certain décalage entre la planification d'une mission et l'exécution d'un plan, problème explicitement posé par Dix [Dix et al., 2003]. Koper y apporte des solutions, et fournit notamment une correspondance entre les variables "concrètes" d'exécution et les variables "abstraites" de planification.

HiDDeN et Koper sont donc étroitement liés (figure 4.2), le second permettant au premier de disposer de toutes les données nécessaires pour faire exécuter les actions du plan aux architectures monovéhicules, de pouvoir en gérer les retours, et d'être capable de générer des domaines et des problèmes de planification en ligne, lorsque une réparation est nécessaire.

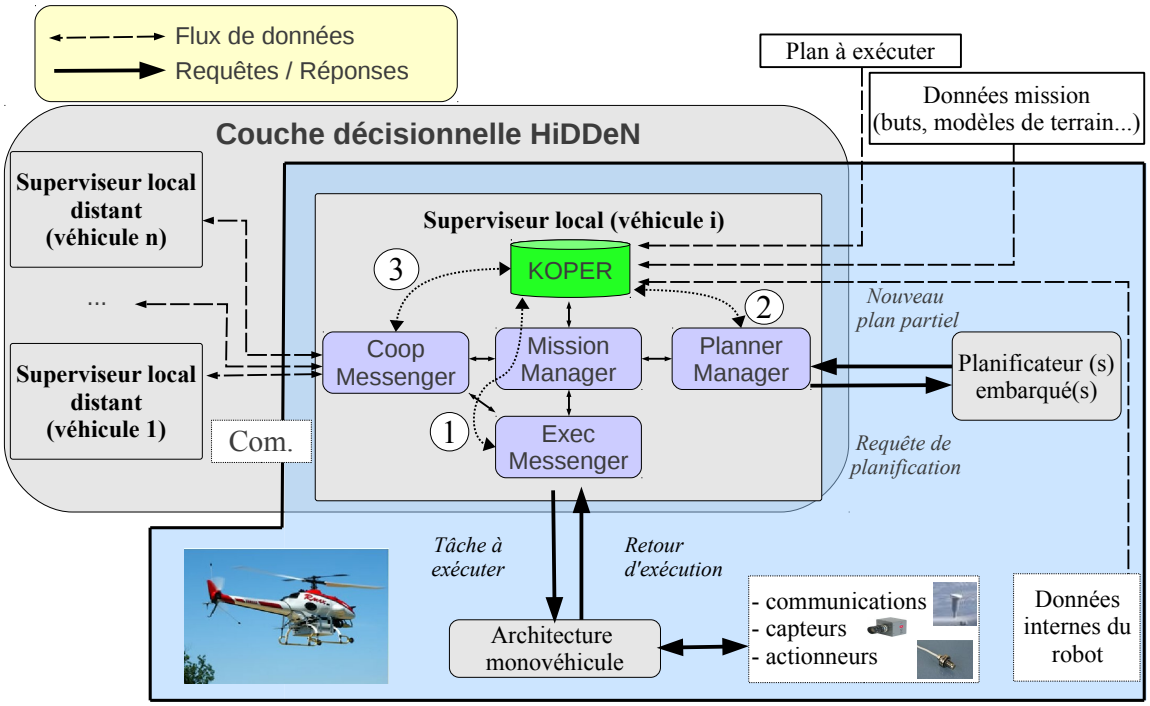
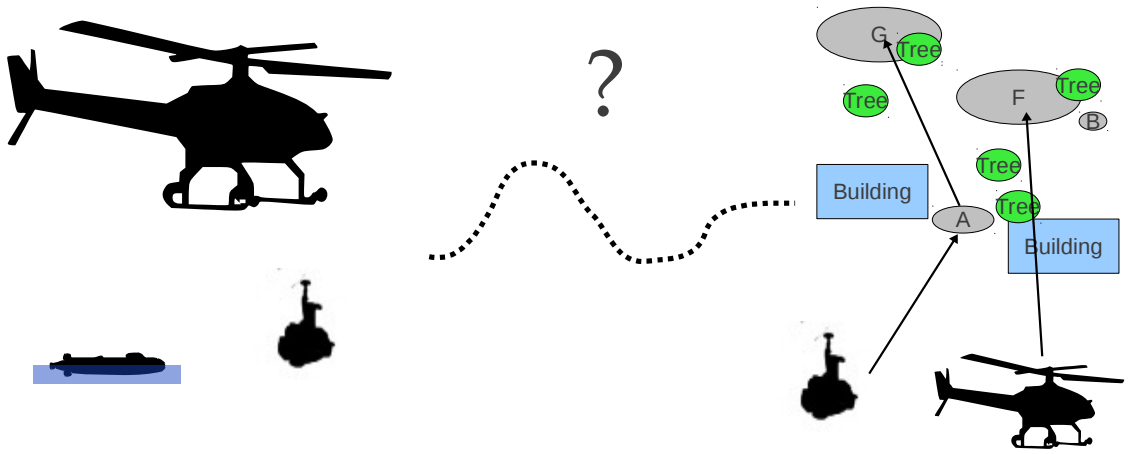


FIGURE 4.2 – Interactions entre la couche HiDDeN et les autres entités, rendues possibles par la base de connaissances Koper. L'*ExecMessenger* peut dialoguer avec l'architecutre monovéhicule grâce à la formalisation de son protocole d'échange de données fourni par Koper (1). Le *Planner-Manager* peut générer un modèle de planification pour générer des requêtes vers un ou plusieurs planificateurs embarqués (2). Enfin, le *CoopMessenger* dispose d'une description des primitives d'échange (qui sont aussi des services de l'architecture monovéhicule) pour pouvoir communiquer avec les autres superviseurs locaux de l'équipe (3).

CHAPITRE 5

Supervision de l'exécution de la mission



Une phase hors ligne permet de générer un plan de mission initial, issu d'une planification HTN. Ce plan est un plan global d'équipe dont les relations entre tâches ainsi que la structure hiérarchique des HTN sont conservées par l'emploi d'un formalisme HTNi, présenté dans ce chapitre. Il permet de décrire les actions que les véhicules de l'équipe vont devoir accomplir pour atteindre les objectifs de la mission. Ce plan est distribué aux superviseurs locaux pour être exécuté. Ce chapitre présente la phase de préparation hors ligne, puis la manière dont la supervision est effectuée du point de vue d'un véhicule autonome par le MissionManager.

Sommaire

5.1 Phase de préparation hors ligne

Pour exécuter une mission, l'équipe peut utiliser un certain nombre d'informations disponibles hors ligne sur l'environnement, les véhicules, les objectifs, etc. Exploiter ces informations sous forme d'un plan initial présente l'avantage potentiel de réduire la durée du calcul décisionnel ultérieur lors du déroulement effectif de la mission. En effet, nous pouvons considérer que le facteur limitant au niveau logiciel pour de telles missions robotiques concerne les calculs de prise de décision, et non la quantité de mémoire dédiée au stockage des données. Lancer l'exécution de la mission avec le plus d'opérations décisionnelles calculées hors ligne semble donc judicieux, notamment avec une représentation sous forme de plan. Il est cependant utopique de considérer qu'un tel plan représenterait tous les états possibles du monde, ce qui implique la nécessité d'embarquer des moyens permettant de faire évoluer ce plan en ligne.

Un autre avantage pratique de la disponibilité d'un plan initial concerne le ou les opérateurs humains responsables du déroulement de la mission, chargés de donner le feu vert pour commencer l'exécution, ainsi que de surveiller son déroulement : même si l'équipe de véhicules autonomes hétérogènes est théoriquement capable de réaliser sa mission sans intervention d'un opérateur humain, ce dernier souhaitera suivre son évolution. Un plan de mission serait donc un atout en terme d'interface entre l'opérateur et l'équipe de véhicules autonomes.

5.1.1 Différents formalismes de plan

Il existe nombre de représentations pour de tels plans de mission déjà utilisées en robotique :

- des listes d'actions, ce que les planificateurs renvoient communément : exemple de BIIMAPS [Sotzing et al., 2008], dont la planification est basée sur les HTN (section 2.2.5, p.25) ; du planificateur TAL basé sur de la logique temporelle et utilisé dans [Doherty et al., 2009] (section 2.2.6, p.26) ;
- des réseaux de tâches simples explicitant les dépendances temporelles entre tâches : exemples du planificateur Europa [Bresina et al., 2005] utilisé dans l'architecture T-Rex [McGann et al., 2008] (section 2.2.4, p.24) et du planificateur IxTet (IndeXed TimE Table) [Laborie, 1995] ;
- des chronogrammes (ou Time Lines) : par exemple solutions d'un CSP (Constraint Satisfaction Problem ou problème de satisfaction de contraintes) dans l'architecture IDEA [Muscettola et al., 2002] (section 2.4.4, p.32) ;
- des structures proches des HFSM (Hierarchical Finite State Machines, ou automates finis hiérarchisés), largement employées en robotique : Retsina [Paolucci et al., 2000] (section 2.4.3, p.30) utilise des schémas de décomposition semblables aux HFSM, la planification étant réalisée par un planificateur HTN ;
- les réseaux de Petri utilisés pour représenter des plans de mission [Costelha and Lima, 2007, Montano et al., 2000], qui disposent de propriétés intéressantes et exploitables sur de telles structures, et dont une structuration hiérarchique est étudiée dans [Bonnet-Torrès and Tessier, 2006] (section 2.6.1, p.36) ;
- des représentations de politiques décisionnelles, construites par exemple à partir de MDP (Markov Decision Process) [Puterman, 1994, Teichteil-Königsbuch et al., 2011] ou associées à des approches réactives qui lient état et action [Brooks, 1991].

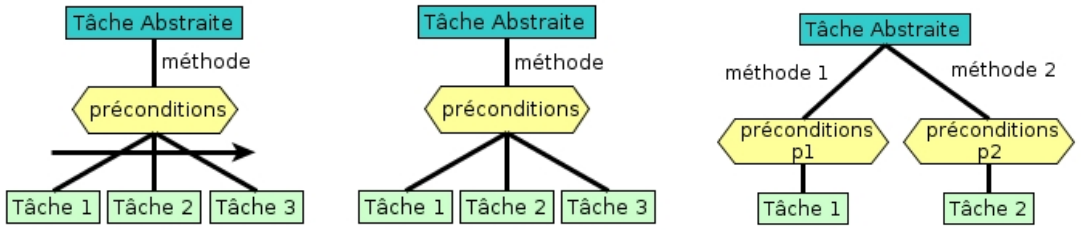
5.1.2 Choix d'un formalisme pour la planification : les HTN

Le plan de mission peut être amené à évoluer pendant la phase d'exécution, puisque l'environnement est dynamique et que des aléas auront lieu. Des réparations de plan ou des replanifications complètes doivent donc être réalisables sur le plan en cours de mission. Des auteurs [Nebel and Koehler, 1993] ont démontré que certaines formes de replanification sont plus complexes que la planification elle-même. Mais parallèlement, des résultats empiriques [Fox et al., 2006] montrent l'intérêt d'une réparation locale de plan face à une replanification globale. La structure hiérarchique de certains de ces formalismes semblent être une base intéressante à exploiter pour des réparations [Paolucci et al., 2000, Bonnet-Torrès and Tessier, 2006, Fazil Ayan et al., 2007], raison pour laquelle nous avons décidé de nous orienter vers ce type d'approche. Construire des structures hiérarchiques à partir d'un processus de planification non hiérarchique est loin d'être trivial. C'est pourquoi nous avons choisi de nous diriger vers des modèles de planification hiérarchiques, et vers les planificateurs qui leurs étaient associés. Hors génération manuelle d'un plan, un modèle de planification doit donc être construit pour pouvoir être donné en paramètre d'entrée à un planificateur.

Dans le cadre applicatif du PEA Action, nous avons choisi de travailler avec des HTN (Hierarchical Task Network) [Erol et al., 1994] pour obtenir le plan de mission. L'intérêt des HTN réside dans leur flexibilité, l'aspect hiérarchique de leur structure et le côté intuitif de leur représentation pour un opérateur humain. "La fréquence de l'étude des HTN dans le milieu de la recherche vient du fait qu'ils correspondent non seulement à une analogie d'un modèle de pensée humain classique, mais qu'ils permettent aussi d'obtenir des gains en terme de temps d'exécution, par rapport à ce que l'on aurait eu en modélisant le problème de façon classique et en utilisant des planificateurs courants" [Hogg, 2007]. En revanche, les HTN ont besoin d'une description complète de leur domaine et de leur problème. Par ailleurs, des planificateurs multiagents tels que A-SHOP [Dix et al., 2003] ont été développés spécifiquement avec ce formalisme.

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 sont atteints. Les HTN sont utilisés de façon classique pour représenter des problèmes de planification. En reprenant les définitions d'Erol, Hendler et Nau [Erol et al., 1994], 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 d est le réseau de tâches pour lequel on souhaite obtenir un plan, I l'état initial, et $\mathcal{D}$ un domaine de planification. Nous précisons ici certains aspects de leur définition avec les différents types de méthodes que nous utilisons (figure 5.1) :

- Une méthode *séquentielle* indique que l'ensemble de ses sous-tâches sont exécutées dans un ordre précis (figure 5.1(a)).
- Une méthode *non ordonnée* indique que l'ordre d'exécution des sous-tâches n'a pas d'importance, ce qu'un planificateur est potentiellement capable d'exploiter pour entrelacer diverses tâches issues de méthodes différentes (figure 5.1(b)).
- Une même tâche abstraite peut avoir différentes méthodes, dotées de préconditions différentes ou identiques (figure 5.1(c)).



(a) Méthode séquentielle. (b) Méthode non ordonnée. (c) Multi-méthode.

FIGURE 5.1 – Différents types de méthodes utilisées dans un HTN.

5.1.3 Plan de mission HTNi

Un planificateur HTN est capable de retourner un plan, i.e classiquement une liste d'actions, lorsqu'un domaine et un problème de planification lui sont fournis. Les relations hiérarchiques entre tâches sont des informations aisées à conserver lors des processus de planification HTN, et peuvent s'avérer utiles sur deux points. Le premier concerne la lisibilité du plan. Si celui-ci respecte les relations d'un domaine de planification, un opérateur mission n'aura pas de mal à en suivre la logique de déroulement et les travaux requis pour la mise au point de la supervision en seront facilités. Le second concerne les réparations de plans, dont nous parlons au chapitre suivant (chapitre 6, p.89). C'est pourquoi notre choix s'est concentré sur les représentations exploitant aussi les aspects hiérarchiques d'un plan.

Divers auteurs [Fazil Ayan et al., 2007, Paolucci et al., 2000] exploitent directement des structures HTN sous des formes équivalentes aux HFSM pour l'exécution. Pour rester à la fois proche du formalisme HTN, tout en étant adapté aux phases d'exécution comme le serait une HFSM, nous formalisons notre propre structure de plan [1].

Définition du HTNi

Nous définissons ainsi une structure d'*arbre HTN instancié*, ou HTNi, 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). Les méthodes et les tâches abstraites sont les nœuds de l'arbre. Nous utilisons la notation 2^X pour désigner l'ensemble des parties de l'ensemble X, et la notation $|X|$ pour désigner le cardinal d'un ensemble X.

Définition 1. *Un **HTNi** $\mathcal{H}$ est un n-uplet* $(E, V, Pre, Post, Te, Ta, \Re, M, t_r)$ *:*

- E *un ensemble d'Etiquettes qui sert à nommer les tâches et méthodes ;*
- V *un ensemble de Variables instanciées ;*
- Pre *un ensemble de Préconditions. Une précondition est une expression booléenne évaluée en fonction des éléments de* V*. C'est une fonction, pour* $n \in \mathbb{N}$ *:* $V^n \rightarrow true, false$ *;*
- $Post$ *un ensemble de Postconditions. Une postcondition est une expression booléenne évaluée en fonction des éléments de* V*. C'est une fonction, pour* $n \in \mathbb{N}$ *:* $V^n \rightarrow true, false$ *;*

1. Un travail préliminaire a cette thèse [Gateau, 2009] a étudié les transformations entre modèles HTN instanciés et HFSM, montrant la possibilité de passer facilement d'une représentation à une autre.

- $Te \subset 2^{Pre} \times 2^{V} \times 2^{Post}$ *un ensemble de Tâches élémentaires. Une tâche élémentaire* t_e *est un triplet* $(Pre(t_e), V(t_e), Post(t_e))$ *avec :*
  - $Pre(t_e) \in 2^{Pre}$ *une liste de préconditions ;*
  - $V(t_e) \in 2^{V}$ *une liste d'arguments ;*
  - $Post(t_e) \in 2^{Post}$ *une liste de postconditions ;*
- $Ta \subset 2^{V} \times 2^{M} \times 2^{Post}$ *un ensemble de Tâches abstraites. Une tâche abstraite* t_a *est un triplet* $(V(t_a), M(t_a), Post(t_a))$ *avec :*
  - $V(t_a) \in 2^{V}$ *une liste d'arguments ;*
  - $M(t_a) \in 2^{M}$ *une liste de méthodes,* $M(t_a) \neq \emptyset$ *; (auxquelles sont associées des préconditions) ;*
  - $Post(t_a) \in 2^{Post}$ *une liste de postconditions ;*
- $\Re$ *est un ensemble de relations d'ordre partiel qui est appliqué à l'ensemble des tâches* $(Ta \cup Te)$*. Nous considérons que* $rel \in \Re$ *est soit séquentielle, soit non ordonnée et nous noterons*
  - $rel = \prec$ *pour désigner une relation* rel *séquentielle :*
    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 *;*
  - $rel = \sim$ *pour désigner une relation* rel *non ordonnée :*
    pour $(t_1, t_2) \in (Ta \cup Te)^2$*,* $t_1 \sim t_2$ *signifie que l'ordre d'exécution entre* t_1 *et* t_2 *n'a pas d'importance.*
- $M \subset Ta \times 2^{Pre} \times 2^{(Ta \cup Te)} \times \Re$ *un ensemble de Méthodes. Une méthode* $m \in M$ *est un quadruplet* $(t_m, Pre(m), st, rel)$ *avec :*
  - $t_m \in Ta$ *la tâche abstraite à laquelle la méthode s'applique ;*
  - $Pre(m) \in 2^{Pre}$ *une liste de préconditions ;*
  - $st \in 2^{(Ta \cup Te)}$ *un ensemble de tâches, abstraites ou élémentaires ;*
  - $rel \in \Re$ *une relation soit séquentielle, soit non ordonnée, entre les éléments de* st*.* rel *permet de définir l'ordre d'exécution des tâches de l'ensemble* st*.*
- t_r *est la tâche abstraite racine qui doit être exécutée. C'est la tâche au niveau d'abstraction le plus élevé dans l'arbre* $\mathcal{H}$ *:*
  $t_r \in Ta$ *est racine de* $\mathcal{H} \iff \forall (t_m, Pre(m), st, rel) \in M, t_r \notin st$*.*

Un arbre HTNi $\mathcal{H}$ *ne comporte pas de cycles :*

$$\forall m = (t_m, Pre(m), st, rel) \in M, \exists! \tau \in Ta \text{ tel que } t_m = \tau \wedge$$
$$\forall (m, m') = ((t_m, Pre(m), st, rel), (t'_m, Pre(m'), st', rel')) \in M^2, \forall t_a \in Ta,$$
$$(m \neq m') \wedge (t_a \in st) \Rightarrow t_a \notin st' \quad (5.1)$$

Une tâche abstraite peut avoir plusieurs méthodes à sa disposition. Classiquement, la planification décide quelle méthode est choisie pour le plan final. Mais plusieurs méthodes pourraient être conservées pour la phase d'exécution. Le plan HTNi serait alors composé de "plans conditionnels" autorisant des politiques de décision en ligne plus flexibles. Des heuristiques pourraient alors être définies au niveau du choix de la méthode à exécuter. Nous verrons que nous prenons en compte cette possibilité dans la procédure d'exécution d'arbre HTNi à la section suivante. Cependant, les planificateurs que nous avons eu l'occasion de manipuler jusque-là ne permettaient pas d'obtenir de tels types de plans.

Une méthode $m = (t_m, Pre(m), st, rel)$ est dite "non ordonnée" pour $rel = \sim$ et dite "séquentielle" pour $rel = \prec$. Une méthode ne comporte pas de tâches ayant à la fois des relations séquentielles et non ordonnées.

Considérer que la relation est soit séquentielle, soit ordonnée n'est pas une limite au modèle. D'après [Erol et al., 1994], cette relation représente en fait un ordre partiel sur les sous-tâches de la méthode. Cet ensemble partiellement ordonné (ou poset) de tâches étant fini et acyclique, il est en effet aisé de démontrer que cette méthode, dont les sous-tâches associées à leur relation d'ordre partielle forment un poset, peut être décomposée en un sous-arbre HTNi ne contenant que des méthodes séquentielles ou non ordonnées. Nous proposons de démontrer cette propriété par construction (algorithme 1), inspirée de l'algorithme de tri topologique de Kahn [Kahn, 1962]. Le poset associé à une méthode est représenté par un graphe acyclique orienté $(\mathcal{V}, \mathcal{E})$, avec pour noeuds $(\mathcal{V})$ les tâches st de la méthode, et pour arcs $(\mathcal{E})$ les contraintes d'ordre sur les tâches définies par rel (un arc (n_1,n_2) indique que la tâche associée à n_1 doit être réalisée avant celle de n_2).

Dans la suite, nous utiliserons la notation $\top$ pour symboliser une condition (précondition ou postcondition) toujours valide, et par extension, nous noterons également $\top$ la tâche $(\top, \emptyset, \top)$ (préconditions et postconditions vraies, sans argument).

Algorithm 1: METHODSORT(m)

1 $m = (t_m, Pre(m), st, rel)$
2 soit $poset = (\mathcal{V}, \mathcal{E})$ le DAG associé au poset (st, rel)
3 $\mathcal{M} = \emptyset$ // ensemble des méthodes créées
4 $\mathcal{T} = \emptyset$ // ensemble des tâches créées
5 $S_1 \leftarrow$ ensemble des tâches de *poset* sans arc entrant // tâches incomparables
6 $\mathcal{T} \leftarrow \mathcal{T} \cup \{t_1 = \top\}$
7 $\mathcal{M} \leftarrow \mathcal{M} \cup \{(t_1, \top, S_1, \sim)\}$
8 $k \leftarrow 1$
9 **while** $\mathcal{V} \neq \emptyset$ **do**
10 $S_{k+1} \leftarrow \emptyset$
11 **foreach** $t \in S_k \cup \mathcal{V}$ **do**
12 $\mathcal{V} \leftarrow \mathcal{V} \setminus \{t\}$
13 **foreach** $t' \in \mathcal{V}$ *tel que* $t \prec t'$ **do**
14 $\mathcal{E} \leftarrow \mathcal{E} \setminus (t, t')$ // ôter l'arc (t, t') de $\mathcal{E}$
15 **if** t' *sans arc entrant* **then**
16 $S_{k+1} \leftarrow S_{k+1} \cup \{t'\}$
17 $\mathcal{T} \leftarrow \mathcal{T} \cup \{t_{k+1} = \top\}$
18 $\mathcal{M} \leftarrow \mathcal{M} \cup \{(t_{k+1}, \top, S_{k+1}, \sim)\}$
19 $k \leftarrow k + 1$
20 $\mathcal{M} \leftarrow \mathcal{M} \cup \{(t_m, Pre(m), \mathcal{T}, \prec)\}$
21 return $(\mathcal{T}, \mathcal{M})$

La stricte décroissance de $\mathcal{V}$ (ligne 12) permet de montrer la terminaison de l'algorithme 1. La complexité de l'algorithme 1 est similaire à celle de l'algorithme proposé par Kahn, à savoir $\mathcal{O}(|\mathcal{V}| + |\mathcal{E}|)$.

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. Par exemple la tâche élémentaire "$goto\ wptA$" désigne une action "$goto$", appliquée avec le paramètre "$wptA$", en supposant que $wptA \in V$ et $goto \in E$. Le terme "élémentaire" est très relatif. Une tâche qui serait "élémentaire" pour la supervision peut être dans

la réalité d'une grande complexité pour le véhicule chargé de son exécution. Il peut par exemple s'agir d'un balayage d'une zone par un AUV, élémentaire dans le HTNi de l'équipe de véhicules et qui implique pour l'architecture locale du véhicule sous-marin lors de son exécution :

1. un calcul d'itinéraire pour rejoindre la zone ;
2. une planification du balayage de la zone ;
3. le suivi des rails ainsi calculés après avoir plongé et enclenché ses différents capteurs (sonars...) ;
4. une remontée en surface.

Une mission complète de recherche et sauvetage d'un véhicule autonome pourrait d'ailleurs ne constituer qu'une tâche élémentaire dans un plan d'équipe, l'"élémentarité" d'une tâche ne dépendant que du niveau d'abstraction auquel on se place.

Relations et notations

Dans la suite de ce paragraphe, nous présentons quelques définitions et notations relatives aux HTNi, que nous utilisons dans la suite. Il s'agit essentiellement de relations qui existent classiquement dans les arbres, et de notations usuelles.

Pour une raison de lisibilité, nous désignerons l'arbre HTNi $\mathcal{H}$ de racine t_r également comme l'arbre HTNi t_r.

Nous utilisons la notion de parenté entre tâches et méthodes : une tâche est fille d'une méthode quand elle fait partie des tâches qui composent une méthode.

$$\tau \in \{Ta \cup Te\} \text{ fille de } m = (r, Pre(m), st, rel) \in M \iff \tau \in st \tag{5.2}$$

$$m = (r, Pre(m), st, \Re) \in M \text{ parente de } \tau \in \{Ta \cup Te\} \iff \tau \in st \tag{5.3}$$

Dans le cas particulier où toutes les tâche abstraites du HTNi n'ont dans leurs listes de méthodes qu'une seule et unique méthode, nous considérerons que les tâches de cette méthode seront filles de la tâche parente.

$$(\forall \tau = (V(\tau), M(\tau), Po(\tau)) \in Ta \mid \exists! m = (r, Pre(m), st, rel) \in M(\tau), \forall \tau' \in st) \Rightarrow (\tau' \text{ fille de } \tau) \tag{5.4}$$

Nous utliserons également la notion de parenté entre tâches : les tâches filles d'une même méthode seront sœurs.

$$\tau \text{ sœur de } \tau' \iff \exists (r, Pm, st, rel) \in M \mid (\tau, \tau') \in st^2 \tag{5.5}$$

Une tâche τ' sera descendante de τ s'il existe une chaîne de parenté entre les deux tâches.

$$\forall \tau \in Ta, \forall \tau' \in \{Ta \cup Te\}, \tau' \text{ descendante de } \tau \iff (\tau' \text{ fille de } \tau) \vee (\exists (t_a) \in Ta \mid ((t_a \text{ descendante de } \tau) \wedge (\tau' \text{ descendante de } t_a)) \tag{5.6}$$

Nous définissons également l'implication d'un véhicule v dans une tâche $\tau \in \{Ta \cup Te\}$ comme le fait que le véhicule est impliqué dans la réalisation de τ. Nous supposons que l'ensemble des véhicules est un sous-ensemble des variables instanciées V. On supposera que le type d'une

variable est fourni par $type(v)$. Nous noterons $veh(\tau) \in 2^V$ l'ensemble des véhicules impliqués dans la réalisation de la tâche τ. Nous noterons :

$$v \in V \text{ impliqué dans la réalisation de } \tau \iff ((v \in veh(\tau)) \wedge (type(v) = \text{ véhicule})) \quad (5.7)$$

Nous supposerons dans la suite que pour toute tâche τ, l'ensemble $veh(\tau)$ est connu. En pratique, il arrive que des plans ne disposent pas de cette information au niveau des tâches abstraites, l'assignation des véhicules n'étant précisée qu'au niveau des tâches élémentaires. Dans ce cas, nous avons proposé (annexe A, p.161) un algorithme de construction d'une structure de sous-équipe, qui permet de retrouver cette information. Dans la suite de cette thèse, nous pouvons donc considérer que $veh(\tau)$ est renseignée pour tout $\tau \in T_a \cup T_e$.

D'autres types de relations peuvent exister au sein d'un sous-ensemble de tâches, comme par exemple une relation de simultanéité, lorsqu'un ensemble de tâches doit s'exécuter au même instant. Ces aspects se rapprochant des domaines de la planification temporelle, nous ne les aborderons que dans les perspectives (section 8.2.2, p.146).

Signalons cependant que sur un même véhicule, deux tâches qui doivent s'exécuter simultanément peuvent être regroupées en une tâche élémentaire unique au sein du HTNi. Le cas de plusieurs tâches qui doivent s'exécuter sur plusieurs véhicules différents et simultanément est similaire. Les tâches peuvent être regroupées lors de la modélisation en une unique tâche élémentaire *conjointe*, qui donnera lieu à un traitement spécifique que nous verrons à la section suivante.

$$\tau \in Te \text{ est conjointe} \iff |veh(\tau)| > 1 \quad (5.8)$$

Dans la suite, pour une méthode ordonnée $m = (t, Pre(m), st, \prec)$, nous indicerons par $i \in [\![1..|st|]\!]$ les tâches de st $\forall (i,j) \in [\![1..|st|]\!]^2, (t_i) \neq (t_j)$ De plus, la croissance de leur indice reflétera leur ordonancement selon $\prec$. Nous noterons l'ensemble de ces tâches ordonnées $(\mathbf{t_i})_{\mathbf{1 \leq i \leq |st|}}^{\mathbf{m}}$:

$$\forall m = (t, Pre(m), st, \prec) \in M, st = (t_i)_{1 \leq i \leq |st|}^{m} \text{ tel que } \forall i \in [\![1..|st|-1]\!], (t_i \prec t_{i+1}) \quad (5.9)$$

5.2 Distribution du plan aux véhicules de l'équipe

5.2.1 Principe général

Le plan HTNi initial est fourni hors ligne, et doit donc être distribué au sein des superviseurs locaux de chacun des véhicules autonomes. L'ensemble des tâches du plan ne concerne pas toujours tous les robots. Des sous-arbres HTNi peuvent donc être ignorés en terme d'exécution par un robot en particulier lorsqu'il n'est responsable de l'exécution d'aucune des tâches de ce sous-arbre. En revanche, des tâches du HTNi peuvent avoir des contraintes d'ordre (suivant la relation d'ordre partiel de leur méthode mère) que le processus d'exécution d'équipe se doit de respecter. Supposons que les tâches t_1 et t_2 doivent être exécutées respectivement par les véhicules v_1 et v_2, et qu'en plus, la méthode m_1 à laquelle ces deux tâches appartiennent impose une contrainte d'ordre du type ($t_1 \prec t_2$). Le véhicule v_2 doit attendre que v_1 ait terminé l'exécution de t_1 pour commencer l'exécution de t_2. Les véhicules v_1 et v_2 étant distincts, le superviseur local de v_2 ne dispose pas *a priori* d'informations sur le fait que la tâche t_1 soit terminée [2], à part s'il en est averti par le superviseur

2. Sauf dans un cas où les véhicules sont coordonnées et respectent scrupuleusement leur plan de mission, mais cela s'éloigne alors de notre étude (pas de coopération), d'autant que le dynamisme de l'environnement ne se prête guère à ce genre d'approche.

local de v_1. Il est donc nécessaire d'expliciter cette communication au niveau du *MissionManager* : lors de la distribution du plan HTNi initial aux superviseurs locaux, une phase de transformation de ce plan vers les plans HTNi locaux qui seront exécutés est opérée. Cette distribution est destinée à simplifier les sous-arbres HTNi pour ne pas chercher à exécuter des tâches qui ne concernent pas le véhicule, et ajouter des tâches de synchronisation destinées à la coopération entre superviseurs locaux. Le principe général consiste à suivre les procédures suivantes, pour un véhicule v où un plan HTNi est destiné à être exécuté :

1. Dans une méthode non ordonnée $m = (t, Pre(m), st, \sim)$, les sous-tâches qui n'impliquent pas v peuvent être ôtées sans impacter sur l'exécution future du HTNi (équation 5.10).

$$\forall m = (t, Pre(m), st, \sim) \in M,\ \forall \tau \in st, v \notin veh(\tau) \text{ alors } st \leftarrow st \setminus \{\tau\} \tag{5.10}$$

2. Dans une méthode séquentielle $m = (t, Pre(m), st, \prec)$, les sous-tâches qui précédent la première tâche où le véhicule v est impliqué sont transformées (si elles existent) en une tâche de synchronisation de type *dist*, réalisée à distance et dont il va falloir attendre le signal de fin d'exécution. Par exemple, dans $(t_i)_{1\leq i\leq |st|}^m$, pour $j \in [\![2..|st|]\!]$, si t_{j-1} n'implique pas v, et $v \in veh(t_j)$, une tâche de synchronisation est créée, destinée à remplacer t_{j-1} (équation 5.11). Nous noterons son étiquette "dist[t_{j-1}]lv", avec lv la liste des étiquettes des véhicules distants ($veh(t_{j-1})$).

$$\begin{aligned}\forall (t, Pre(m), st, \prec) \in M,\ &\text{pour } (\tau_i)_{1\leq i\leq |st|}^m,\\ &\forall v \in V, (v \notin veh(\tau_i) \wedge v \notin veh(\tau_{i+1})) \text{ alors } st \leftarrow st \setminus \{\tau_i\}\\ &\forall v \in V, v \notin veh(\tau_i) \wedge v \in veh(\tau_{i+1}) \text{ alors } st \leftarrow (st \setminus \{\tau_i\}) \cup \{t_{dist}\}\end{aligned} \tag{5.11}$$

avec t_{dist} une tâche de synchronisation de type *dist*.

Et de la même manière, pour $j \in [\![1..|st|-1]\!]$ pour toute tâche $t_j \in (t_i)_{1\leq i\leq |st|}^m$ où v est impliqué et suivie d'une tâche t_{j+1} où v n'est pas impliqué, une tâche de synchronisation doit être exécutée à la suite de t_j. Il s'agit d'une synchronisation de type *com*, indiquant qu'il faut engager une communication avec au moins un véhicule distant pour lui signaler que la tâche t_j est terminée (équation 5.12). Nous la noterons "com[t_j]lv", avec lv la liste des étiquettes des véhicules distants $veh(t_{j+1})$.

$$\begin{aligned}\forall (t, Pre(m), st, \prec) \in M,\ &\text{soit } st = (\tau_i) | \forall i \in [\![1..|st|]\!], \tau_i < \tau_{i+1}\\ &\forall v \in V, v \in veh(\tau_i) \wedge v \notin veh(\tau_{i+1}) \text{ alors } st \leftarrow st \cup \{t_{com}\}\end{aligned} \tag{5.12}$$

avec t_{com} une tâche de synchronisation de type *com*.

3. Enfin, pour la réalisation de tâches conjointes, il peut être bénéfique d'ajouter des tâches de synchronisation de type *sync* pour encadrer l'exécution de la tâche conjointe : garantir que les véhicules qui vont s'engager sur la tâche conjointe sont prêts à le faire, puis garantir qu'ils l'ont tous effectivement terminée (équation 5.13).

$$\forall \tau \in Te, \tau \text{ conjointe alors } st \leftarrow st \cup \{t_{sync1}, t_{sync2}\} \tag{5.13}$$

avec t_{sync1} et t_{sync2} deux tâches de synchronisation de type *sync*.

5.2.2 Algorithme de distribution

L'algorithme 2 présente la procédure de distribution recursive pour un arbre HTNi de racine r, pour un véhicule v, et en reprenant les notations de la définition 1.

Si la tâche est abstraite (ligne 3), on vérifie les types de relation qu'ont ses méthodes. Lorsque celles-ci sont non ordonnées, l'algorithme 3 est appelé (ligne 7). Sinon, lorsqu'elles sont séquentielles, l'algorithme 4 se charge de la distribution (ligne 10). Si la tâche est élémentaire mais qu'elle n'implique pas v, elle n'est pas conservée dans le plan (ligne 25). Si elle implique uniquement v, on ne fait rien en particulier, elle est conservée telle quelle. Au contraire, si la tâche est élémentaire et est conjointe, on suit le principe de l'équation 5.13. Des tâches de synchronisation de type *sync* sont créées, t_{sync1} et t_{sync2} (lignes 16 et 17) et sont ajoutées aux sous-tâches de la méthode m (ligne 20). Les postconditions de ces tâches sont fournies par le *CoopMessenger* (ligne 16 et 17) via la procédure COOPMESSENGER::CHECKANDACK. La postcondition diffère en fonction du type de synchronisation associée à la tâche, à savoir :

- $(\tau, sync_1)$: tous les véhicules $veh(\tau)$ ont envoyé un message précisant qu'ils sont prêt à exécuter τ ;
- $(\tau, sync_2)$: tous les véhicules $veh(\tau)$ ont envoyé un message précisant qu'ils ont achevé τ ;
- (τ, com) : tous les véhicules $veh(\tau)$ ont été prévenus de l'exécution de la tâche précédent τ ;
- $(\tau, dist)$: tous les véhicules $veh(\tau)$ ont envoyé un message précisant qu'ils ont achevé τ.

L'algorithme 3 détaille la distribution d'une méthode non ordonnée : en suivant le principe de l'équation 5.10, on supprime les tâches où le véhicule v n'est pas impliqué (ligne 5). Au contraire, lorsque v est impliqué par la tâche courante, on appelle l'algorithme 2 pour appliquer la distribution à cette tâche (ligne 7).

Enfin, l'algorithme 4 détaille la distribution d'une méthode séquentielle. L'algorithme 2 est appelé pour chacune des sous-tâches de la méthode m afin de propager la distribution aux sous-arbres HTNi (lignes 5 et 22). Si v est impliqué dans la réalisation de la tâche courante t_i mais pas dans la suivante (lignes 6 et 7), le principe de l'équation 5.12 est suivi. Une tâche de synchronisation de type *com* est créée ligne 8 et aura pour rôle, lors de l'exécution, de prévenir l'ensemble des véhicules impliqués dans la tâche suivante ($veh(t_{i+1})$) que la tâche précédente (t_i) s'est terminée avec succès. Les postconditions de t_{com}, sont fournies par le *CoopMessenger* ligne 8.

De façon symétrique, si v n'est pas impliqué dans la réalisation de la tâche courante t_i mais l'est dans la suivante (lignes 12 et 14), le principe de l'équation 5.11 est suivi. Une tâche de synchronisation de type *dist* est créée ligne 16 et aura pour rôle, lors de l'exécution, d'attendre que les véhicules distants impliqués dans la tâche courante (t_i) exécutent une tâche de type *com*. Les postconditions de t_{dist} devront exprimer le fait que des messages extérieurs aient été reçus et sont fournies par le *CoopMessenger* ligne 16. En revanche, si v n'est ni impliqué dans la réalisation de la tâche courante t_i, ni dans la suivante t_{i+1} (ligne 19), t_i est ôtée de la liste des tâches de m (ligne 20).

Dans notre architecture HiDDeN, le *CoopMessenger* gère l'exécution des tâches de synchronisation, qui lui sont transmises par l'*ExecMessenger*. C'est en effet son rôle de gèrer les messages de synchronisation échangés entre les différents superviseurs locaux indispensables à la cohésion d'équipe.

Algorithm 2: DISTRIBUTE(r, v)

```
1  r = (V(r), M(r), Po(r)) ∈ T_A
2  v ∈ V tel que type(v) = véhicule
3  if r ∈ Ta then
4      foreach m_i ∈ M(r) do
5          soit m_i = (r, Pm, st, rel)
6          if rel = ∼ then
7              DISTRIBUTENONORDO(m_i,v)
8          else
9              //rel = ≺
10             DISTRIBUTESEQ(m_i,v)
11 else
12     if v ∈ veh(r) then
13         if |veh(v)| > 1 then
14             //création de tâches de synchronisation si la tâche
15             //est jointe
16             t_sync1 = (⊤, veh(t_i),COOPMESSENGER::CHECKANDACK(t_i,sync_1))
17             t_sync2 = (⊤, veh(t_i),COOPMESSENGER::CHECKANDACK(t_i,sync_2))
18             // ajout de tâches de synchronisation de type SYNC
19             // juste avant et juste après t_i
20             st ← st ∪ {t_sync1, t_sync2} tel que
21                    (t_sync1 ≺ t_i ≺ t_sync2)
22                    ∧(∀(t_p) ∈ st, (t_p ≺ t_i) ⟺ (t_p ≺ t_sync1)
23                    ∧(∀(t_q) ∈ st, (t_i ≺ t_n) ⟺ (t_sync2 ≺ t_q)
24     else
25         r ← ∅ //si v n'est pas impliqué par la tâche, il ne la
                 //conserve pas dans son plan
```

Algorithm 3: DISTRIBUTENONORDO(m, v)

```
1  v ∈ V tel que type(v) = vhicule
2  soit m = (r, Pm, st, ∼)
3  foreach t_i ∈ st do
4      if v ∉ veh(t_i) then
5          st = st \ {t_i}
6      else
7          DISTRIBUTE(t_i, v)
```

Algorithm 4: DISTRIBUTESEQ(m, v)

1 $v \in V$ tel que $type(v) = vhicule$
2 soit $m = (r, Pm, st, \prec)$
3 soit $(t_i)_{1 \leq i \leq |st|}^{m} \leftarrow st$ tel que $\forall i \in [\![1..|st|-1]\!], t_i \prec t_{i+1}$
4 **for** $i = 1..(|st|-1)$ **do**
5 DISTRIBUTE(t_i, v)
6 **if** $v \in veh(t_i)$ **then**
7 **if** $v \notin veh(t_{i+1})$ **then**
8 $t_{com} = (\top, veh(t_{i+1})$,COOPMESSENGER::CHECKANDACK(t_{i+1},*com*)))
9 // ajout tâche de synchronisation de type *com*
10 $st \leftarrow st \cup \{t_{com}\}$ tel que $t_i \prec t_{com} \prec t_{i+1}$
11 **else**
12 //$v \notin veh(t_i)$
13 //on vérifie si la tâche suivant t_{i+1} implique v
14 **if** $v \in veh(t_{i+1})$ **then**
15 //dès que t_{i+1} implique v, on construit une tâche de synchronisation (dist) à la place de t_i
16 $t_{dist} = (\top, veh(t_i)$,COOPMESSENGER::CHECKANDACK(t_i,*dist*)))
17 $t_i = t_{dist}$
18 // la tâche courante t_i sera exécutée "à distance"
19 **else**
20 $st = st \setminus \{t_i\}$ //Le successeur à t_i n'implique pas v, on peut donc ôter t_i
21 //cas aux limites
22 DISTRIBUTE($t_{|st|}, v$)

La figure 5.2 illustre la modification d'arbres HTNi simples destinés à deux véhicules suivant l'algorithme 2. La figure 5.3 montre une séquence à trois véhicules. Dans la suite, nous verrons son application à des HTNi plus complexes, à la section 5.3.

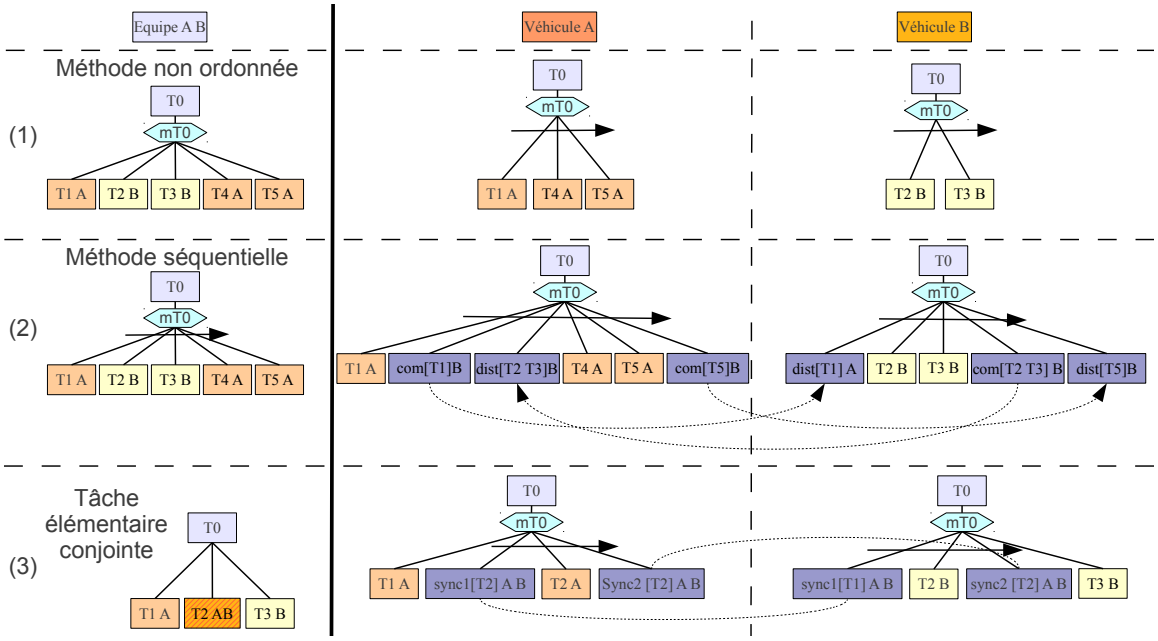


FIGURE 5.2 – Schéma de la distribution et de la génération de tâches de synchronisation, nécessaires à la coopération pour deux véhicules A et B, avec (1) une méthode non ordonnée, (2) une méthode séquentielle, (3) une tâche élémentaire conjointe. A gauche, HTNi dans le plan d'équipe. Au milieu, HTNi distribué au véhicule A. A droite, HTNi distribué au véhicule B.

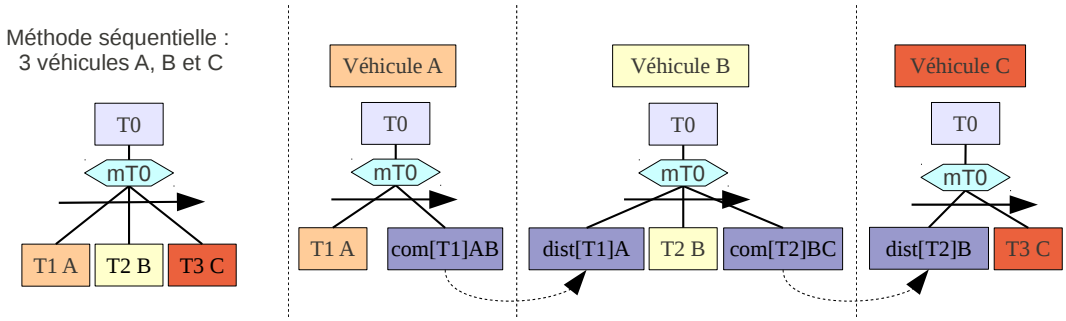


FIGURE 5.3 – Schéma de la distribution et de la génération de tâches de synchronisation, nécessaires à la coopération pour trois véhicules différents, A, B et C. De gauche à droite, le HTNi d'équipe, puis le HTNi distribué à bord de A, de B et de C.

5.3 Exemple : mission aéromaritime (PEA Action, scénario II)

Le HTNi initial peut prendre diverses formes pour l'exécution d'une même mission, suivant les modèles qui ont servis à sa génération. L'application de l'algorithme de distribution ne donne pas

le même plan à exécuter suivant ces versions, la création de tâches de synchronisation dépendant de la structure du HTNi initial fourni en entrée.

Nous allons illustrer ce fait au travers des plans HTNi obtenus pour le scénario II du PEA Action. Comme nous l'avons précédemment dit (section 1.4, p.12), il s'agit pour une équipe constituée d'un véhicule autonome aérien à voilure tournante (AAV) et d'un véhicule autonome sous-marin (AUV) que nous nommerons respectivement aav1 et auv1, de blanchir un chenal. Il s'agit de vérifier que le chenal ne contient pas de mines. La surface d'opération est divisée en trois zones, subdivisées chacune en trois sous-zones. L'AUV ne peut intervenir que si une sous-zone est garantie sans bateau (risque de collision, ou possibilité éventuelle pour le bateau de reposer des mines après passage de l'AUV). L'AAV est limité en carburant, et devra éventuellement effectuer des ravitaillements sur une zone extérieure aux précédentes.

Ce scénario a été modélisé sous forme d'un domaine et d'un problème HTN. Un planificateur adapté dont nous reparlerons au chapitre 7 (p.105) nous permet d'obtenir des plans HTNi, prêts à être exécutés par l'équipe. La figure 5.4 montre un exemple de HTNi initial, et la figure 5.5, sa distribution associée à bord de l'aav1 et de l'auv1. L'étape de distribution ajoute 6 tâches de synchronisation par véhicule aux 30 tâches élémentaires qui doivent être exécutées (9 balayages, 3 déplacements et 3 tâches de synchronisation par véhicule).

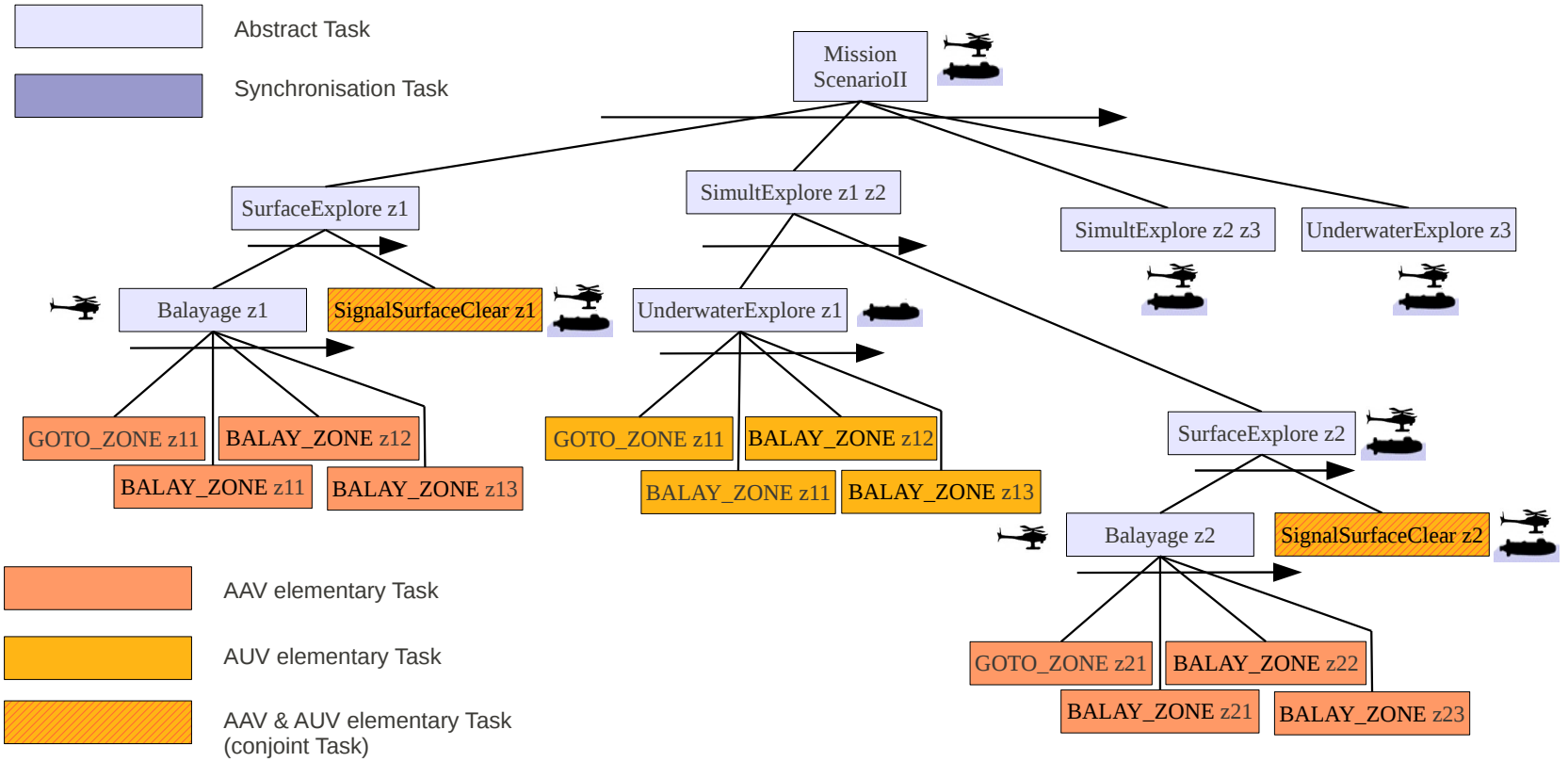


FIGURE 5.4 – HTNi partiel du scénario II du PEA action, version hiérarchique décomposée pour optimiser le temps de mission (aav1 commence à balayer la zone suivante pendant que auv1 est en plongée).

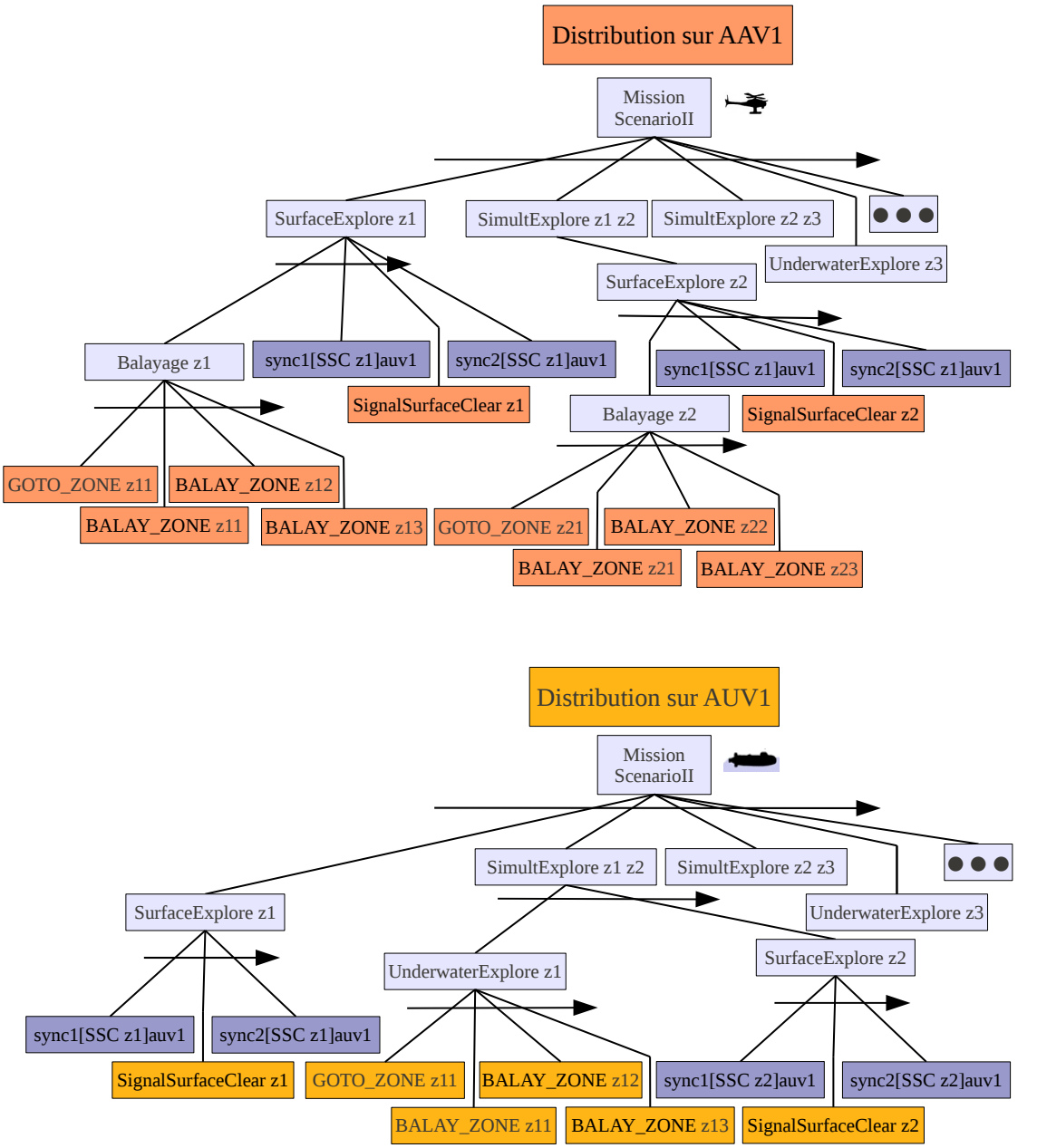


FIGURE 5.5 – HTNi partiel du scénario II du PEA action, version hiérarchique décomposée pour optimiser le temps de mission (aav1 commence à balayer la zone suivante pendant que auv1 est en plongée). Distribué sur le superviseur local de l'aav1 (en haut) et sur celui de l'auv1 (en bas).

5.4 Exécution d'un plan de mission HTNi

Après l'étape de distribution, le *MissionManager* peut lancer l'exécution de la mission, via l'architecture de contrôle locale du 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 rôle du *MissionManager* va consister à parcourir l'arbre HTNi en profondeur d'abord, en suivant la relation d'ordre partiel de chaque méthode rencontrée, pour déterminer quelles tâches élémentaires doivent être transmises à l'*ExecMessenger* qui demandera son exécution à l'architecture monovéhicule. Dans ce paragraphe, nous faisons l'hypothèse qu'aucun aléa de perturbe la supervision de la mission par HiDDeN.

Lorsqu'une tâche abstraite est atteinte, ses préconditions sont testées, et dans le cas où diverses méthodes sont disponibles et leurs préconditions valides, l'une d'entre elles 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 *MissionManager* en vérifie les préconditions. Si celles-ci sont valides, la tâche est transmise à l'*ExecMessenger*.

Ce dernier envoie explicitement l'ordre d'exécution de l'action correspondante vers l'architecture monovéhicule. Une fois l'exécution de la tâche terminée, l'architecture monovéhicule est supposée renvoyer un compte rendu d'exécution (qui peut se borner à un simple message "ok, tâche terminée") que l'*ExecMessenger* réceptionne. Le compte rendu est ensuite transmis au *MissionManager* qui peut alors vérifier si les postconditions de la tâche élémentaire sont satisfaites. Si c'est le cas, le parcours de l'arbre HTNi peut reprendre.

L'algorithme 5 décrit la procédure suivie par le superviseur local, où la tâche t passée en paramètre correspond à la tâche racine d'un HTNi, au démarrage de la mission.

Algorithm 5: MISSIONMANAGER : :EXECUTE(t)

```
1  if t ∈ Ta then
2      m = (t, Pre(t), st, rel) = MISSIONMANAGER::CHOOSEMETHOD(t)
3      // trie les éléments de st pour les exécuter dans un ordre
4      // choisi
5      (t_i)_{1≤i≤|st|} =SORT(st, rel)
6      for i = 1..|st| do
7          MISSIONMANAGER::CHECKPRECONDITIONS(t_i)
8          MISSIONMANAGER::EXECUTE(t_i)
9          MISSIONMANAGER::CHECKPOSTCONDITIONS(t_i)
10 else if t ∈ Te then
11     MISSIONMANAGER::CHECKPRECONDITIONS(t)
12     EXECMESSENGER::EXECUTE(t)
13     MISSIONMANAGER::CHECKPOSTCONDITIONS(t)
```

La procédure CHOOSEMETHOD (ligne 2) permet de sélectionner la méthode m pour exécuter la tâche t. Ses préconditions sont d'ailleurs vérifiées lors de cette sélection. La méthode choisie peut arbitrairement correspondre à la première pour lesquelles les préconditions sont valides, mais cette sélection peut être complexifiée pour exploiter des heuristiques qui guideraient l'exécution du plan. Les sous-tâches de m sont triées afin de déterminer leur ordre d'exécution (ligne 5). Ce tri doit respecter la relation rel de m. Si rel = $\prec$, l'ordre est identique. Si rel = $\sim$, l'ordre peut

être arbitraire (par exemple, l'ordre lexicographique des étiquettes associées aux tâches) mais peut être complexifié. Si la tâche à exécuter est une tâche élémentaire (ligne 10), l'action associée à cette tâche doit être exécutée, via un appel vers l'*ExecMessenger* (ligne 12). Si elle est abstraite, un appel récursif à l'algorithme 5 est lancé (ligne 8) pour parcourir et exécuter les sous-tâches de t_i. A chaque exécution d'une tâche, ses préconditions sont vérifiées (lignes 7 et 11). Lorsque l'exécution d'une tâche est terminée, ses postconditions sont à leur tour vérifiées (lignes 9 et 13). Les ensembles de sous-tâches de chaque méthode, et l'ensemble des tâches abstraites d'un HTNi étant finis, et un arbre HTNi sans cycle, l'algorithme 5 termine. Sa complexité est linéaire en $\mathcal{O}(|Ta|+|Te|)$, chacune des tâches du HTNi n'étant exécutée au plus qu'une fois (les tâches de méthodes non choisies ne sont pas exécutées).

Dans la suite du mémoire, les représentations des HTNi seront simplifiés : lorsqu'une tâche abstraite n'est composée que d'une méthode, celle-ci ne sera pas représentée. (en pratique les planificateurs que nous avons exploités ne fournissent pas de plans composés de tâches multi-méthode, celles-ci devaient être ajoutées manuellement au plan).

La figure 5.6 illustre le parcours et l'exécution d'un arbre HTNi. La tâche racine à exécuter est R. L'algorithme 5 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 $E2$, $T1$ est supposée "terminée". L'algorithme développe ensuite le sous-arbre de la tâche $T3$, et ainsi de suite. Un diagramme de séquence figure 5.7 montre les appels à l'*ExecMessenger* au fur et à mesure que l'arbre HTNi de la figure 5.6 est exécuté.

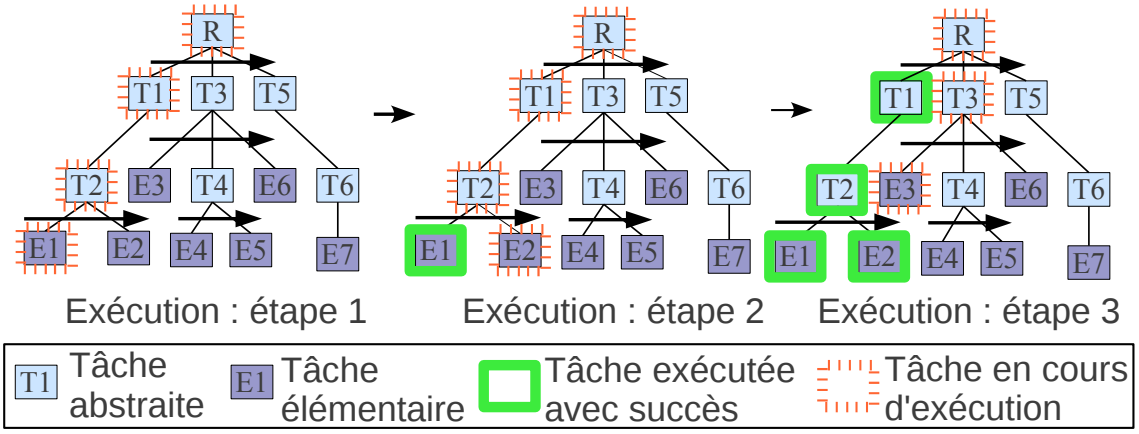


FIGURE 5.6 – Exécution d'un HTNi : l'ordre d'exécution des tâches élémentaires est $E_1, E_2, E_3, E_4, E_5, E_6, E_7$.

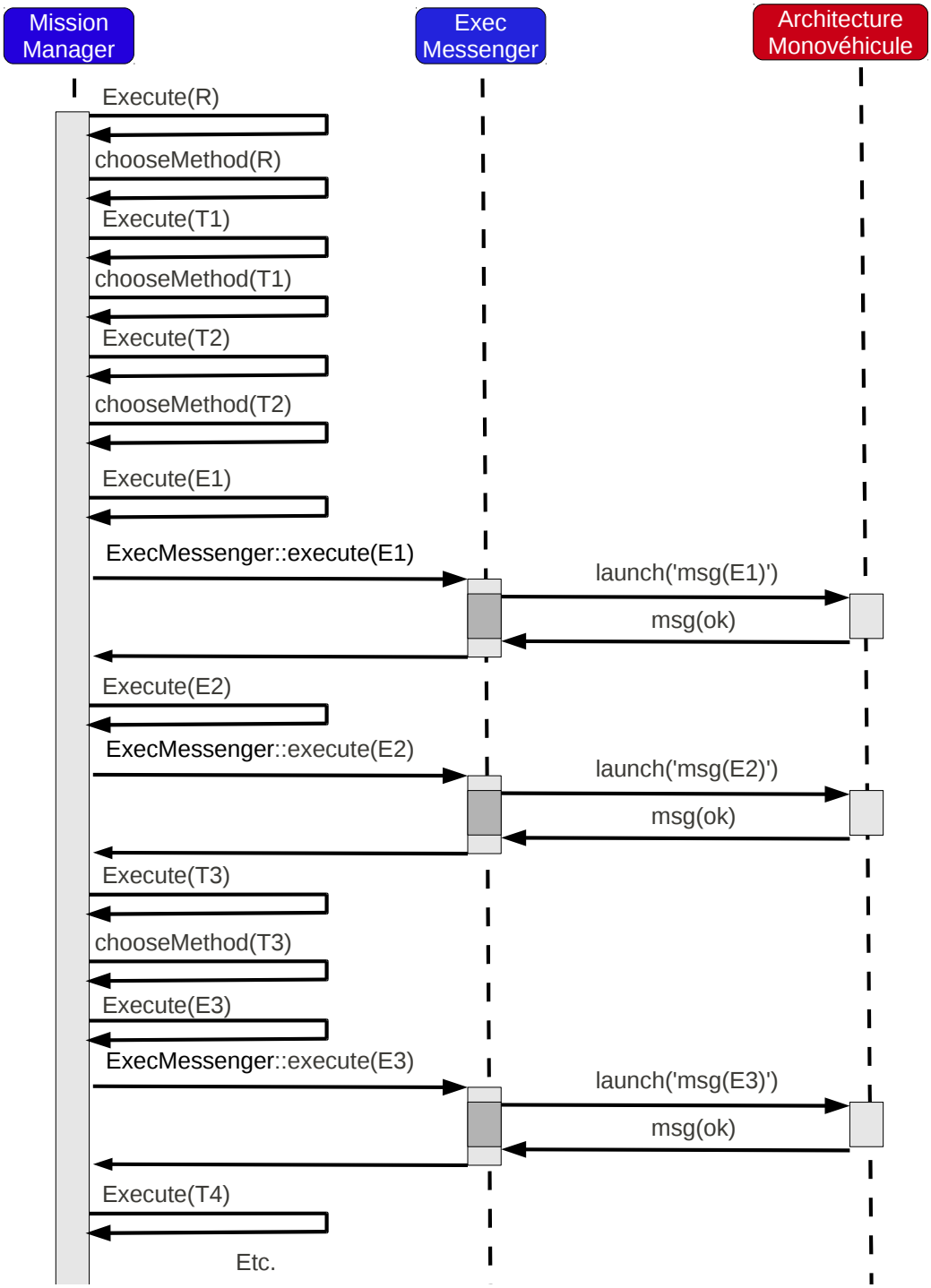


FIGURE 5.7 – Diagramme de séquence de l'exécution du HTNi de la figure 5.6.

5.5 Détection d'aléas lors de l'exécution

Lors de l'exécution du plan de mission, l'équipe est confrontée à un environnement dynamique, et des aléas peuvent avoir lieu lors de l'exécution du plan de mission. En ce qui concerne les aspects décisionnels, nous proposons de détecter cinq types d'aléas différents au niveau du *MissionManager* :

1. *Préconditions invalides* : D'après la définition des HTNi (définition 5.1.3), toute tâche possède un ensemble de préconditions. Avant de lancer l'ordre d'exécution d'une tâche dans la structure HTN, le *MissionManager* teste la validité des préconditions qui lui sont associées (algorithme 5, lignes 7 et 11). Si au moins une des préconditions n'est pas satisfaite, le contrat de la tâche (`SerContract`) décrit dans Koper (section 4.2, p.57) n'est pas valide (au travers de `ConRequire`). Le véhicule ne peut donc exécuter cette tâche et un aléa de type "Préconditions invalides" est détecté.
2. *Echec d'exécution* : L'exécution d'une tâche élémentaire consiste à interagir, *via* l'*ExecMessenger*, avec l'architecture monovéhicule. L'*ExecMessenger* formate convenablement la requête émanant du *MissionManager* pour que le véhicule puisse accomplir la tâche voulue. Du point de vue du véhicule autonome, l'exécution peut se dérouler correctement, ou échouer pour diverses raisons. Un retour d'exécution est attendu par l'*ExecMessenger* qui est chargé de son analyse, puis de sa transmission vers le *ExecMessenger*. Suivant ce retour, un aléa peut être détecté si l'exécution ne s'est pas déroulé convenablement et que l'architecture monovéhicule l'a signalé. Le niveau d'information associé à l'aléa dépend totalement de l'architecture monovéhicule. Koper dispose de tels renseignements, au travers de ses champs `ConReturn`. Par exemple, suite à l'exécution d'un "goto" d'un AGV, l'*ExecMessenger* peut recevoir un simple message "ko" pour indiquer un échec, ou un message plus complet indiquant la position à laquelle il s'est arrêté, la raison de cet arrêt (un obstacle bloquant la route par exemple), etc.
3. *Postconditions invalides* : D'après la définition des HTNi (définition 5.1.3), toute tâche possède un ensemble de postconditions. Ces préconditions correspondent aux champs `ConEnsure` de Koper (section 4.2, p.57). Lorsque la partie exécutive d'une action a été effectuée de façon supposée correcte, le *MissionManager* vérifie, lorsque c'est possible, si les effets désirés de l'action sont valides (algorithme 5, lignes 9 et 13). Si ce n'est pas le cas, le contrat (`SerContract`) décrit dans Koper n'est pas respecté et l'action est considérée comme un échec.
4. *Invariants invalides* : D'après la formalisation (section 4.2, p.57) des tâches, toute tâche possède un ensemble d'invariants `ConInvariant` (par exemple : carburant suffisant). Si ceux-ci sont détectés comme étant invalides, un aléa de type "Invariant non respecté" est détecté. En pratique, cette détection est moins aisée, l'interaction entre *ExecMessenger* et architecture monovéhicule n'étant pas toujours aisée à un niveau aussi fin. Il est cependant possible de mettre en œuvre des "observateurs" qui vérifient l'état des invariants périodiquement pendant l'exécution des tâches. La détection de ce type d'aléa n'a pas été implémenté pour l'instant.
5. *Timer terminé* : Pour éviter qu'une tâche s'éternise (et par conséquent bloque l'exécution de la suite du plan, par exemple lorsqu'un événement inattendu a eu lieu et que l'architecture locale n'a tout simplement pas renvoyé de message), le superviseur local arme un timer lors de l'ordre d'exécution d'une tâche élémentaire par l'architecture locale. Si l'action nécessite un temps supérieur à une certaine valeur (associée à la tâche, ou définie par défaut), l'exécution est considérée comme trop longue, interrompue, et l'aléa "Timer terminé" déclenché.

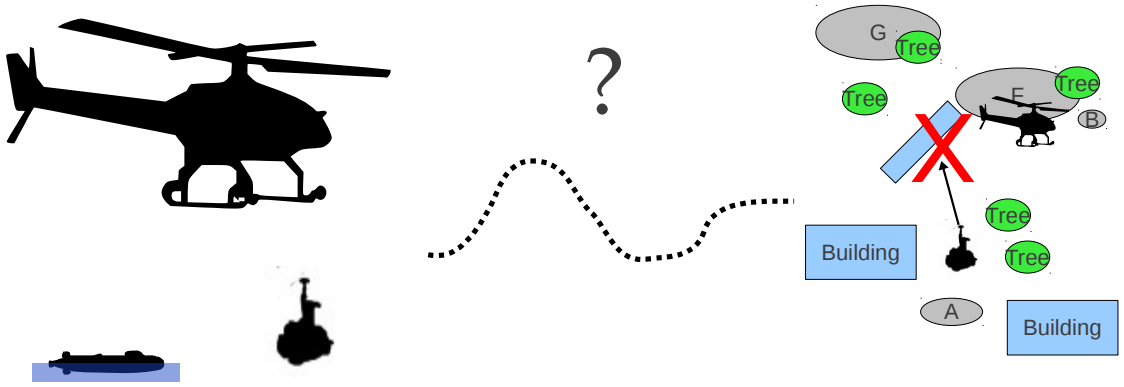
Les timers sont en particulier utiles pour détecter des états dans lequel l'architecture mono-véhicule ne s'est pas rendu compte d'être dans un état anormal, ou notamment pour déceler des rendez-vous manqués entre deux véhicules.

5.6 Conclusion

Au cours de ce chapitre, nous avons défini le formalisme de plan que nous employons : les arbres HTNi, directement inspirés des HTN. Nous avons montré comment un plan initial est distribué au sein des superviseurs locaux des véhicules de l'équipe, c'est-à-dire les modifications qui leur sont apportées pour gérer les aspects de synchronisation qui sont nécessaires pour que les véhicules de l'équipe puissent coopérer dans l'accomplissement de leur mission. Nous nous somme basés sur le scénario II du PEA Action pour illustrer ce processus. L'exécution de tels plans peut ne pas se dérouler de façon nominale. Nous avons constaté que certains aléas pouvaient être détectés par la couche décisionnelle. Une fois l'aléa détecté, un processus de réparation doit être enclenché : le plan courant ne peut plus être exécuté. Nous décrivons dans le chapitre suivant le processus de réparation que nous avons mis en œuvre.

CHAPITRE 6

Réparer en ligne le plan de mission



Pendant le déroulement de la mission, l'équipe de véhicules autonomes évolue dans un environnement dynamique et des aléas ont donc de fortes probabilités d'apparaître, invalidant potentiellement le plan en cours d'exécution. HiDDeN, supervisant l'équipe, doit pouvoir y faire face, et par conséquent fournir des solutions alternatives pour atteindre les objectifs initiaux. Nous décrivons dans ce chapitre la réponse des superviseurs locaux face à l'occurrence d'aléas, la stratégie de réparation la plus locale possible que nous avons choisi d'implémenter, ainsi que la gestion des ruptures de communication.

Sommaire

6.1 Prise en compte des aléas lors de l'exécution

L'équipe de véhicules autonomes évolue dans un environnement dynamique pour exécuter sa mission, et peut se retrouver dans des états qui n'étaient pas initialement prévus dans le plan de mission. Il est donc indispensable de considérer que les superviseurs locaux puissent décider de la démarche à suivre lors de l'occurrence d'aléas pendant la phase d'exécution. Nous nous plaçons à un niveau purement délibératif pour des aléas que les architectures monovéhicules, même en s'appuyant sur leurs capacités réactives ou délibératives individuelles, ne seraient pas capables de résoudre seules. Notre architecture HiDDeN n'a pas vocation à résoudre l'ensemble des aléas pouvant survenir durant le déroulement d'une mission robotique, le spectre de celui-ci étant considérable [Crestani and Godary-Dejean, 2012]. Dans le chapitre précédent, nous avons listé les types d'aléas qui peuvent être détectés au niveau décisionnel (section 5.5, p.87) par le *MissionManager* et c'est sur ceux-là que nous avons concentré nos efforts.

Avec notre formalisme de plan, l'occurrence d'aléas requiert une modification du plan de mission courant pour tenter d'accomplir les objectifs initialement définis : il s'agit d'une *replanification*. Nous distinguerons la *replanification totale*, où l'ensemble des objectifs est pris en compte et un nouveau plan de mission complet est généré, de la *réparation locale* qui consiste à modifier localement, lorsque c'est possible, le plan de mission courant en réponse à l'aléa.

Dans notre architecture, régénérer tout ou partie d'un plan de mission est du ressort du *PlannerManager* (figure 3.2, p.50). Ce dernier se charge des requêtes adressées à un planificateur, à qui il faut fournir les modèles de planification pour qu'il puisse retourner un plan solution. Ce plan, s'il existe, doit ensuite être pris en compte par l'équipe, pour la suite de la mission. Dans cette section, nous décrivons l'approche générale que nous adoptons pour répondre à l'occurrence d'aléas. Nous détaillons à la section suivante la façon dont nous avons implémenté cette réponse, en nous appuyant sur la structure hiérarchique des HTNi (section 6.2).

En terme de planification, des résultats empiriques [Fox et al., 2006] montrent l'intérêt de la réparation locale de plan face à une replanification globale. Sur les missions multivéhicules où les contraintes de communication ne garantissent pas la possibilité, à chaque instant, de pouvoir échanger des données entre les véhicules, la réparation locale permet de limiter :

- le nombre de véhicules concernés par la réparation : on peut intuitivement supposer que plus celle-ci est locale, moins le nombre de véhicules impliqués sera grand ;
- la quantité de données échangées : une réparation locale peut ne nécessiter qu'un sous-ensemble des données globales de la mission ;
- la durée de la réparation : en général, un sous-problème coûte moins cher à résoudre en terme de durée que le problème global.

En revanche, il faut garder à l'esprit qu'une réparation locale ne produit pas toujours un plan alternatif idéal :

- une solution globale plus efficace peut exister ;
- une branche du plan qui reste à exécuter, et non réparée, peut être involontairement rendue incohérente et non exécutable (les préconditions d'une tâche ne seront pas respectées par exemple) ;
- des problèmes de cohérence peuvent apparaître lorsque plusieurs véhicules ne disposent plus du même plan de mission.

Au vue des contraintes imposées dans les scénarios du PEA Action (fortes contraintes sur les communications), nous avons choisi de nous orienter vers des réparations locales de plan, d'autant plus que la structure des HTNi s'y prête particulièrement bien.

6.2 Réparation du plan HTNi aussi locale que possible

La stratégie mise en place doit être capable de fournir en cas d'aléas des sous-plans alternatifs à des branches du plan initial. Dans le cas le plus contraint, une replanification totale peut être nécessaire. Pour cela, l'algorithme 5 (p.84) que nous avions défini dans le chapitre précédent pour l'exécution d'un plan HTNi, est étendu en une nouvelle version, l'algorithme 6, pour prendre en compte les replanifications potentielles. Celles-ci sont gérées par le *PlannerManager* et permettent de poursuivre la mission même lorsqu'un aléa est détecté. La solution consiste à renvoyer explicitement les résultats de chaque appel aux différentes procédures et à s'appuyer sur une procédure de réparation fournie par le *PlannerManager* lorsque l'une d'entre elles échoue.

Algorithm 6: MISSIONMANAGER : :EXECUTE(t)

```
1  if t ∈ Ta then
2      m = (t, Pre(t), st, rel) = MISSIONMANAGER::CHOOSEMETHOD(t)
3      if m = nil then
4          return FAILURE
5      (t_i)_{1≤i≤|st|} = SORT(st, rel)
6      for i = 1..|st| do
7          if not(MISSIONMANAGER::CHECKPRECONDITIONS(t_i)) then
8              return FAILURE
9          else if MISSIONMANAGER::EXECUTE(t_i) returns FAILURE then
10             if PLANNERMANAGER::REPAIR(t_i) returns SUCCESS then
11                 return MISSIONMANAGER::EXECUTE(t_i)
12             else return FAILURE
13         if not(MISSIONMANAGER::CHECKPOSTCONDITIONS(t_i)) then
14             return FAILURE
15 else if t ∈ Te then
16     if not(MISSIONMANAGER::CHECKPRECONDITIONS(t)) then
17         return FAILURE
18     else if EXECMESSENGER::EXECUTE(t) returns FAILURE then
19         return FAILURE
20     if not(MISSIONMANAGER::CHECKPOSTCONDITIONS(t)) then
21         return FAILURE
22 return SUCCESS
```

La procédure SORT (ligne 5) permet de fournir un ordre pour l'exécution des tâches en fonction de la relation rel (procédure qui peut être basée sur un algorithme de tri topologique, tel celui de Kahn [Kahn, 1962]). Dans l'algorithme 6, un renvoi d'échec est ajouté (ligne 3) s'il n'existe pas de méthode aux préconditions valides (aléa de type *Préconditions invalides*). Comme dans l'algorithme 5 (p.84), une méthode est choisie pour l'exécution de la tâche t (ligne 2). Désormais, la fonction EXECUTE renvoie un résultat : SUCCESS ou FAILURE. Le résultat de l'appel à MISSIONMANAGER::EXECUTE (anciennement à la ligne 8 de l'algorithme 5) est analysé : si l'exécution n'a pas retourné SUCCESS (aléa de type *Echec d'exécution*), (ligne 9), la procédure PLANNER-

MANAGER::REPAIR(t) du *PlannerManager* est appelée (ligne 10). Si la réparation est réussie, la procédure MISSIONMANAGER::EXECUTE(t_i) est rappelée (ligne 11) pour permettre à l'exécution de continuer. En effet, la méthode permettant de réaliser t_i a été modifiée par la réparation, donc la tâche t_i peut à nouveau être exécutée. Dans l'autre cas, le résultat FAILURE est renvoyé. Enfin, lors de l'appel à l'action associée à une tâche élémentaire (ligne 12 de l'algorithme 5), si les préconditions de la tâche sont valides (ligne 16), l'action associée à la tâche est appelée (ligne 18), et le retour de cet appel également testé. Si l'exécution s'est bien déroulée, un dernier test (ligne 20) vérifie que les postconditions attendues sont valides. Si l'un de ces trois tests échoue, le résultat FAILURE est renvoyé avant de continuer : un aléa a été rencontré et les trois tests successifs permettent de déterminer de quel type d'aléa il s'agit (aléa de type *Préconditions invalides*, *Echec d'exécution* ou *Postconditions invalides*).

Avant d'appeler la procédure MISSIONMANAGER::EXECUTE d'une tâche, un timer (dont la durée dépend de la tâche, information renseignée par Koper) est systématiquement déclenché en parallèle afin de limiter la durée d'exécution de la tâche. Si le timer expire (aléa de type *Timer terminé*), la procédure MISSIONMANAGER::EXECUTE est interrompue, et le résultat FAILURE est renvoyé, remplaçant son résultat retourné : la tâche aura mis trop de temps à s'exécuter pour pouvoir être considérée comme se déroulant convenablement. Par exemple, cette démarche est particulièrement utile quand il s'agit de détecter le non-respect d'une fenêtre de rendez-vous pour un véhicule.

6.2.1 Processus de réparation locale

Détaillons plus particulièrement la procédure PLANNERMANAGER::REPAIR(t) (ligne 10). C'est une requête vers un planificateur : il s'agit de remplacer la branche du plan ayant échoué par un plan alternatif, obtenu en lançant un planificateur sur un domaine et un problème particulier. Son positionnement dans l'algorithme 6 n'est donc pas anodin car il permet de réaliser des réparations de plan qui tiennent compte de la structure hiérarchique du HTNi. Si cette réparation échoue, c'est au tour de la tâche directement supérieure dans la hiérarchie du HTNi d'être réparée : on peut considérer ce processus comme un *backtrack* dans le HTNi. Au final, toute la branche du HTNi ayant échouée est remplacée au niveau de la dernière tâche réparée. Si la tâche racine du HTNi est atteinte par backtracks successifs, nous aurons affaire à une replanification totale du plan de mission. Dans le cas où celle-ci échouerait, une procédure d'annulation de la mission peut être lancée. La figure 6.1 illustre ce processus de réparation.

Une requête de planification contient toutes les informations qui doivent être fournies à un planificateur pour qu'il puisse retourner un plan solution. Dans le cadre HTN, ces informations peuvent être décomposées en un *domaine* (les actions disponibles pour modifier l'état du monde) et un *problème* (une instance de l'état du monde, c'est-à-dire l'état initial ainsi que l'état but à atteindre). Nous reprenons ici cette décomposition. Soit D le domaine initial de planification, et P, le problème initial de planification. On note $P = (e, b)$, avec e l'état initial du système et b l'ensemble des buts qui doivent être atteints. On peut définir une nouvelle requête de planification $\mathcal{R} = (\mathcal{D}', \mathcal{P}')$, avec :

- $\mathcal{D}'$ le nouveau domaine ;
- $\mathcal{P}' = (e', b')$ le nouveau problème, avec e' l'état courant du système et b' le(s) nouveau(x) but(s) à résoudre.

Nous avons vu au chapitre 4 que la base de connaissances Koper était capable de fournir toutes les informations pour une telle requête de planification. Lorsque le *MissionManager* détecte la présence d'un aléa, il fait appel au *PlannerManager*.

Nous noterons $\mathcal{H}$ le HTNi qui était en cours d'exécution lors de la détection de l'aléa. Dans

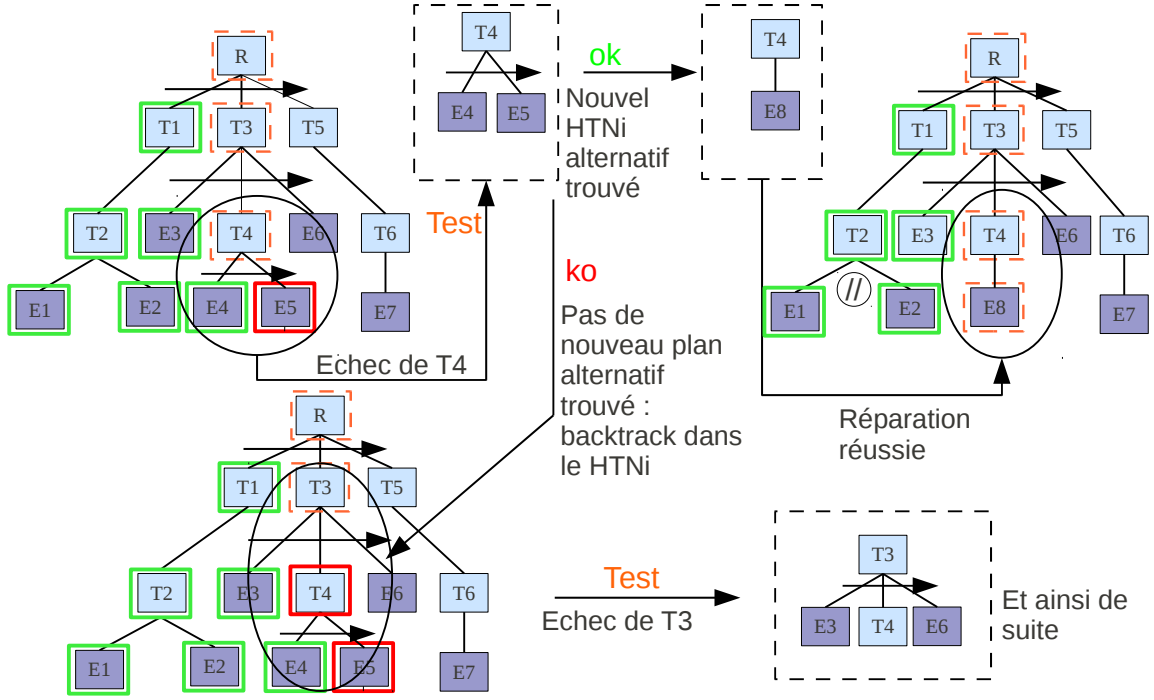


FIGURE 6.1 – Exemple de processus de réparation : la tâche élémentaire E_5 échoue, ce qui entraine une tentative de réparation de la tâche T_4 (en haut à gauche). Si un nouveau plan est disponible, la branche du HTNi correspondante est remplacée (en haut à droite) ; sinon, la tâche de plus haut niveau directement supérieure à T_4, ici T_3, est réparée (en bas).

notre cas, la procédure PLANNERMANAGER::REPAIR(t) (ligne 10) revient à formuler une requête de planification destinée à être traitée par un planificateur. Celui-ci doit renvoyer un arbre HTNi $\mathcal{H}$' ayant la même tâche t pour racine, mais dont le reste de la structure aura été modifié. Le but à résoudre b' correspond donc à la résolution de la tâche t, associée aux mêmes variables instanciées. En effet, dans le cadre de nos travaux, nous posons l'hypothèse que pour toute tâche t de nos HTNi, il existe une tâche dans le domaine du HTN qui lui est associée, guidant la requête de planification à envoyer au planificateur HTN. Cette hypothèse est aisée à valider sachant que nos HTNi sont issus d'une planification HTN réelle, et donc que toutes leurs tâches existent bien dans le domaine de planification [1]. L'état courant e' et le domaine $\mathcal{D}'$ sont générés en s'appuyant sur le le processus décrit à la section 4.3.2 (p.62) et en fonction du type d'aléa, aspect que nous précisons à la section suivante. Leur génération permet de compléter la requête de planification qui prendra la forme $\mathcal{R} = (\mathcal{D}', (e', t))$. Une fois le plan HTNi $\mathcal{H}$' (de racine t) renvoyé par le planificateur, la procédure PLANNERMANAGER::REPAIR(t) achève son déroulement en remplaçant la tâche t de l'ancien HTNi $\mathcal{H}$ par le nouvel HTNi $\mathcal{H}$'. L'exécution peut alors continuer sur le plan $\mathcal{H}$ ainsi réparé. La figure 6.2 illustre une réparation locale qui ne concerne qu'un seul véhicule.

1. Seules les tâches de synchronisation générées lors du processus de distribution pourraient en être absentes, impliquant alors un échec lors de la réparation. Cet échec conduit à un backtrack sur le HTNi, qui entraine la réparation d'une tâche parente à la tâche de synchronisation, laquelle est bien définie dans le domaine de planification.

6.2.2 Types d'aléas et influence sur la réparation

Les différents types d'aléas, *Préconditions invalides*, *Echec d'exécution*, *Postconditions invalides*, *Timer terminé* et *Invariants invalides* (section 5.5, p.87) ont généralement un impact différent sur le processus de réparation.

Dans le cas où il s'agit d'un aléa de type *Préconditions invalides*, le domaine de planification est identique à la dernière replanification du véhicule. En effet, les définitions des tâches ne sont pas remises en cause, seul l'état du monde est *a priori* responsable de l'aléa. Par exemple, pour se référer à la figure 6.2, la présence d'un bâtiment de surface a été détectée dans une zone où le plan initial n'en prévoyait pas. Le planificateur fournira donc la tâche à exécuter pour obtenir les mêmes effets initialement désirés. Pour reprendre notre exemple, la nouvelle tâche à exécuter consisterait probablement à chasser le bâtiment de surface hors de la zone. Un aléa de type *Invariants invalides* verra sa réparation se dérouler d'une façon similaire au type d'aléa *Préconditions invalides*, l'état des variables (liées aux invariants) ayant évolué dans la base de connaissances Koper. La génération du problème pourra donc bien en tenir compte.

Les autres types d'aléas peuvent signifier que la description de la tâche élémentaire dans le domaine est invalidée : son exécution a pu échouer (l'architecture locale a renvoyé un message le précisant), a pris trop de temps, ou n'a pas eu les effets escomptés. En supposant ses préconditions valides, répéter la tâche à l'exécution impliquerait un échec similaire [2]. Effectuer une requête de planification sur le même domaine de la même tâche renverrait le même HTNi associé à la tâche. Grâce à Koper, la liste des services d'un robot (`RobServices` sur la figure 4.1, p.58) peut être mise à jour lors de la détection de l'aléa. Il est possible de préciser que le service associé à la tâche qui a échoué n'est par exemple plus exploitable, ou nécessite une durée d'exécution supérieure. Cela permet de générer un nouveau domaine de planification modifié, tenant compte du fait que la tâche ayant échouée n'est plus disponible.

Dans le cas d'aléas du type *Postconditions invalides*, la solution que nous préconisons consiste à ajouter aux préconditions du contrat de la tâche (`ConRequire` sur la figure 4.1, p.58) la négation de l'état courant dans lequel se trouve le véhicule. Ainsi, lors de la génération du domaine de planification, la tâche fautive ne peut plus être sélectionnée lors de la phase de planification.

Pour un aléa du type *Echec d'exécution*, la réparation dépend des informations mises à jour dans Koper *via* le retour d'exécution reçu par l'*ExecMessenger*. Si la cause de l'échec est explicite, celle-ci peut être analysée par le *MissionManager* qui peut alors mettre à jour Koper (par exemple, lors d'un déplacement, un véhicule se retrouve coincé contre un mur et peut donc préciser non seulement que l'action a échoué, mais qu'il est en plus bloqué par un obstacle). Si l'architecture monovéhicule n'apporte aucune donnée relative à l'échec, on se retrouve dans le même cas que l'aléa de type *Postconditions invalides*, et le même genre de solution peut être appliquée (le véhicule se rend compte que son déplacement à échoué, mais ne donne pas d'autre précision...)

Enfin, un aléa de type *Timer terminé* s'appuie généralement sur le contexte d'exécution de la tâche. S'il s'agissait d'une tâche de type rendez-vous, il est fort probable que la terminaison du timer est justifiée par le fait que le ou les véhicules distants n'aient pas respecté une fenêtre temporelle imposée dans le plan de mission. Le processus de réparation peut donc être déclenché, les données temporelles ayant évoluées. En revanche, s'il s'agit d'une tâche élémentaire classique,

2. En pratique, dans un monde réel, ce n'est pas toujours vrai : répéter une action n'ayant pas eu les effets désirés ne se solde pas toujours par un nouvel échec. Nous laisserons ce point pour les stratégies d'implémentation, où plusieurs exécutions de la même action peuvent être tentées (par exemple l'allumage d'un capteur qui ne répondrait pas toujours à la première sollicitation, etc.).

on peut supposer que le véhicule autonome n'a pas détecté qu'il était dans un état non nominal d'exécution (par exemple, un rover terrestre dont l'accomplissement d'une action goto ne serait basé que sur la comparaison de ses coordonnées par rapport au point de destination ne détecte pas nécessairement un cas où il serait embourbé et que ses roues tourneraient toujours). On se retrouve alors dans une configuration d'*Echec d'exécution*, sans information.

Nous venons d'étudier le cas monovéhicule ($|veh(t)| = 1$). Dans le cas multivéhicule ($|veh(t)| > 1$) décrit à la section suivante, la réparation est plus complexe, car il faut gérer les communications entre véhicules, s'assurer que les bases de connaissances Koper des différents véhicules sont synchronisées et enfin que l'ensemble des véhicules impliqués par l'aléa aient bien réparé leur plan HTNi courant à l'issu du processus. Le principe reste cependant identique, comme le montre la figure 6.3 qui illustre une réparation multivéhicule.

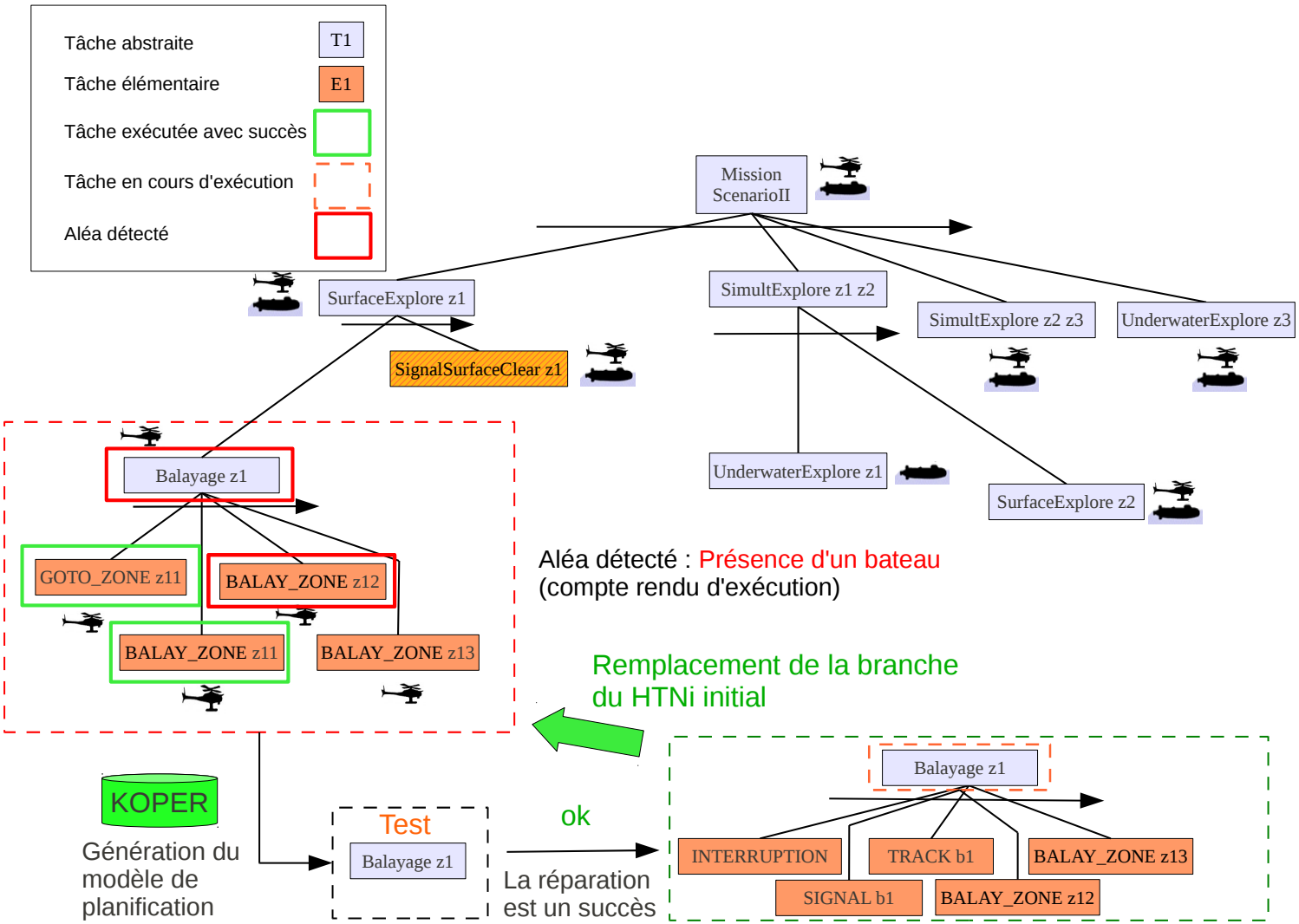


FIGURE 6.2 – Exemple de réparation monovéhicule dans le scénario II du PEA Action.

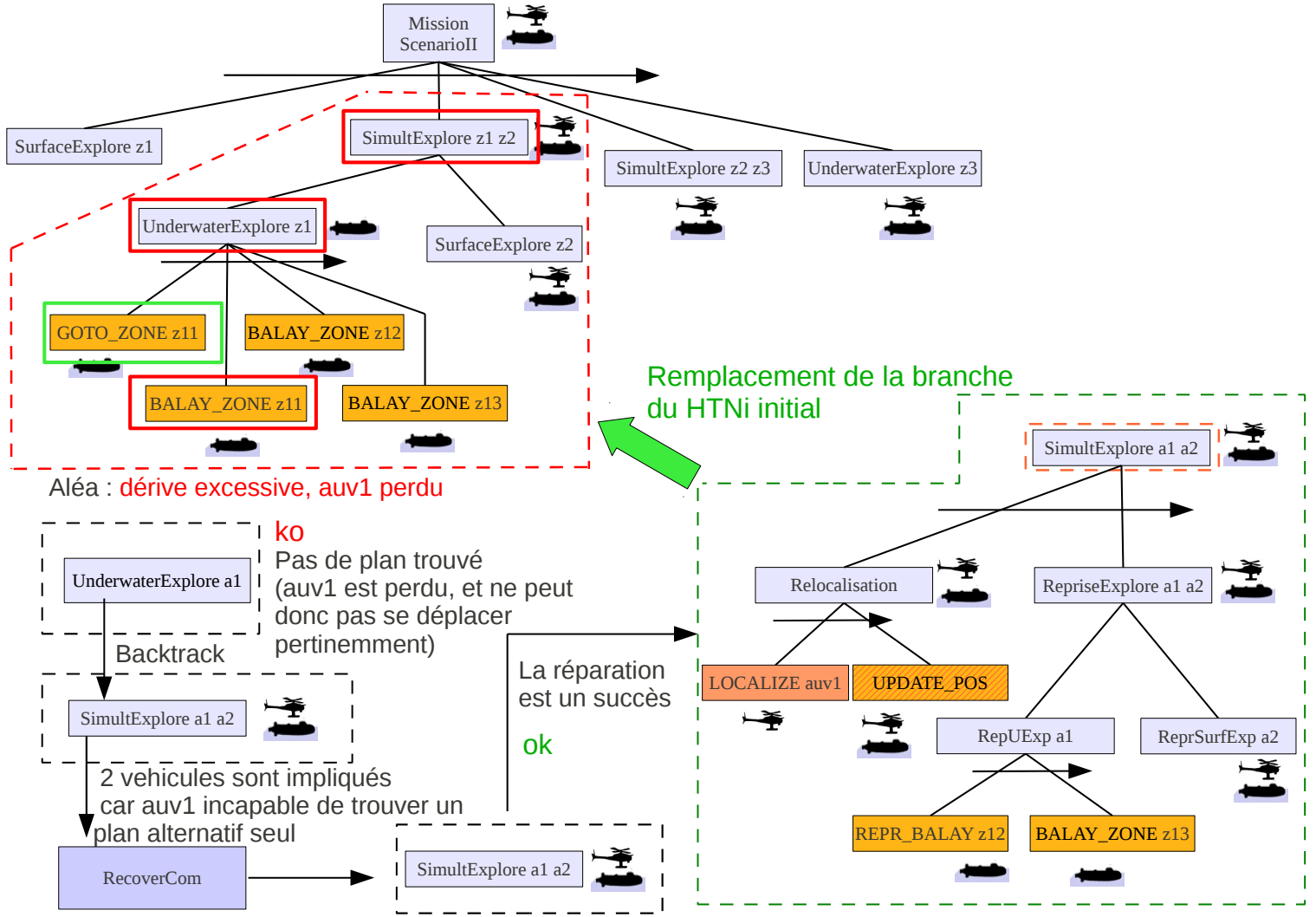


FIGURE 6.3 – Exemple de réparation multivéhicule dans le scénario II du PEA Action.

6.3 Phase de synchronisation pour une réparation d'équipe

Le fonctionnement de la réparation mettant en œuvre plusieurs véhicules dépend des capacités de communication dans l'équipe. Pour des systèmes multirobots coopérants, des stratégies de recouvrement des communications doivent donc être explicitement mises en place, et non reléguées à des "boîtes noires" mal spécifiées [Parker, 1998]. Lorsqu'un véhicule est confronté à une réparation qui implique ses coéquipiers, la priorité devient de rétablir la communication avec eux. Une fois cette phase effectuée, les bases de données des véhicules doivent être synchronisées, puis la phase de réparation effectivement lancée, et appliquée à tous les véhicules concernés.

Nous utilisons des schémas HTN pour guider la phase de réparation d'équipe. Nous nous inspirons pour cela de travaux sur l'entrelacement de la planification et de l'exécution en HTN [Paolucci et al., 2000]. La première tâche à résoudre pour la planification consiste donc à retrouver la communication entre les véhicules. S'entremêlent ensuite exécution partielle de plan (tant que les tâches se déroulent comme elles étaient prévues dans le schéma) et replanification, lorsqu'une communication qui était estimée possible dans le plan échoue par exemple, ou qu'une tâche forçant un appel à la planification est rencontrée. La figure 6.4 (p.100) illustre l'utilisation d'un schéma HTN lors des réparations impliquant plus d'un véhicule (ici, la figure présente le HTNi résultant de son instanciation). La tâche ayant échouée est remplacée par un arbre HTNi temporaire, qui sera lui-même amené à subir des évolutions pendant son exécution. La première phase de ce schéma consiste à établir les liens de communication nécessaires entre v (le véhicule victime de l'aléa) et les véhicules distants. C'est au planificateur de trouver le moyen le plus efficace. Si la communication était déjà disponible, v peut directement prévenir ses coéquipiers. Sinon, il peut avoir à effectuer différentes manœuvres permettant de recouvrer les communications (se diriger vers la dernière position connue du véhicule, aller à un point de rendez-vous où les communications ont plus de chance d'être disponibles...) Lorsque les communications sont rétablies, le signalement de la présence d'un aléa est envoyé, suivi d'une mise à jour des bases de connaissances de v en fonction des véhicules distants. Celui-ci se charge alors d'effectuer la réparation de plan [3] qu'il peut alors renvoyer à ses coéquipiers. Les véhicules peuvent ensuite mettre à jour leur plan courant, y relancer un processus de distribution pour maintenir la cohérence avec les véhicules distants, puis continuer leur exécution.

D'autres schémas HTN que celui illustré par la figure 6.4 (p.100) sont utilisés pour répondre à des besoins particuliers. Dans le cas où toutes les replanifications tentées ont échoué, une procédure d'annulation de la mission peut être déclenchée. Des schémas décrivent aussi les procédures de recouvrement de communication quand cette dernière n'est pas disponible mais que des échanges de données doivent avoir lieu. Ces schémas correspondent en partie à une expertise que l'opérateur fournit lorsqu'il décrit l'environnement de la mission dans lequel l'équipe va évoluer.

3. Nous traitons le cas d'une planification centralisée dans un véhicule. On pourrait aussi utiliser un processus de replanification distribué entre les différents véhicules, comme de l'allocation de tâches par exemple.

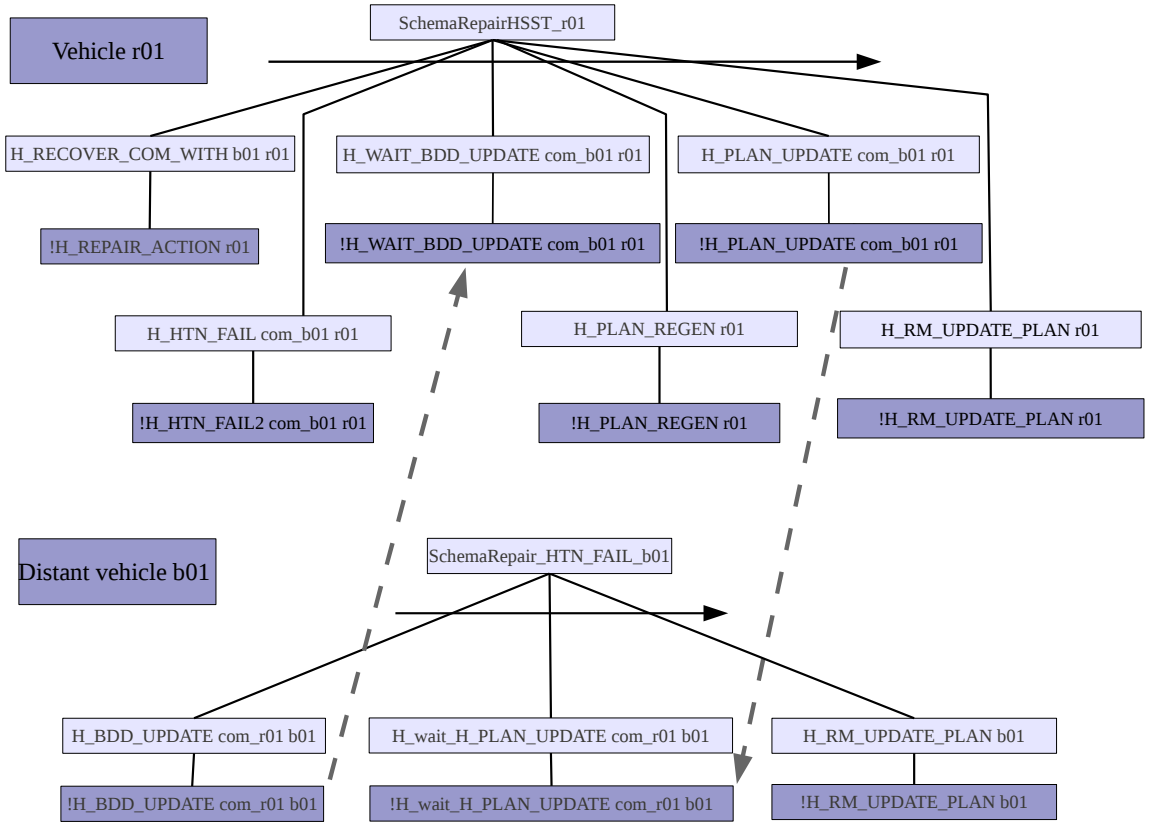


FIGURE 6.4 – Exemple de schéma utilisé lorsqu'un aléa implique plusieurs véhicules. Celui qui a détecté l'aléa doit vérifier qu'il dispose d'un lien de communication avec les véhicules distants concernés (tâche *H_RECOVER_COM_WITH*), puis leur signaler que le plan courant ne se déroule pas comme prévu (tâche *H_HTN_FAIL*). Il attend alors de recevoir leurs connaissances actuelles (tâche *H_WAIT_BDD_UPDATE*) pour mettre à jour les siennes. La phase de génération du plan partiel peut être effectivement lancée (tâche *H_PLAN_UPDATE*). Le plan obtenu est transmis à tous les autres véhicules en même temps qu'une mise à jour de leurs données. Le plan courant est mis à jour en fonction de ce plan partiel, à bord de chacun des véhicules (tâche *H_RM_UPDATE_PLAN*).

Les diagrammes de séquence suivants (figure 6.5 et figure 6.6) illustrent les échanges qui ont lieu entre deux véhicules lors d'une phase de réparation d'équipe. Le premier (figure 6.5) montre un cas "nominal" de réparation, tandis que le second (figure 6.6) met en relief un cas où les comunications ne sont pas disponibles. Il s'agit de diagrammes issus de missions d'exploration. Les véhicules doivent atteindre des points d'intérêt. Nous supposons que $r1$ est un AGV (un véhicule terrestre) et $r2$ un AAV (un véhicule aérien). L'équipe, composée des deux véhicules $\{r1,r2\}$, est chargée d'exécuter le HTNi $T4$ de la figure 6.1. $r1$ est responsable de $E4$ et $r2$ de $E5$.

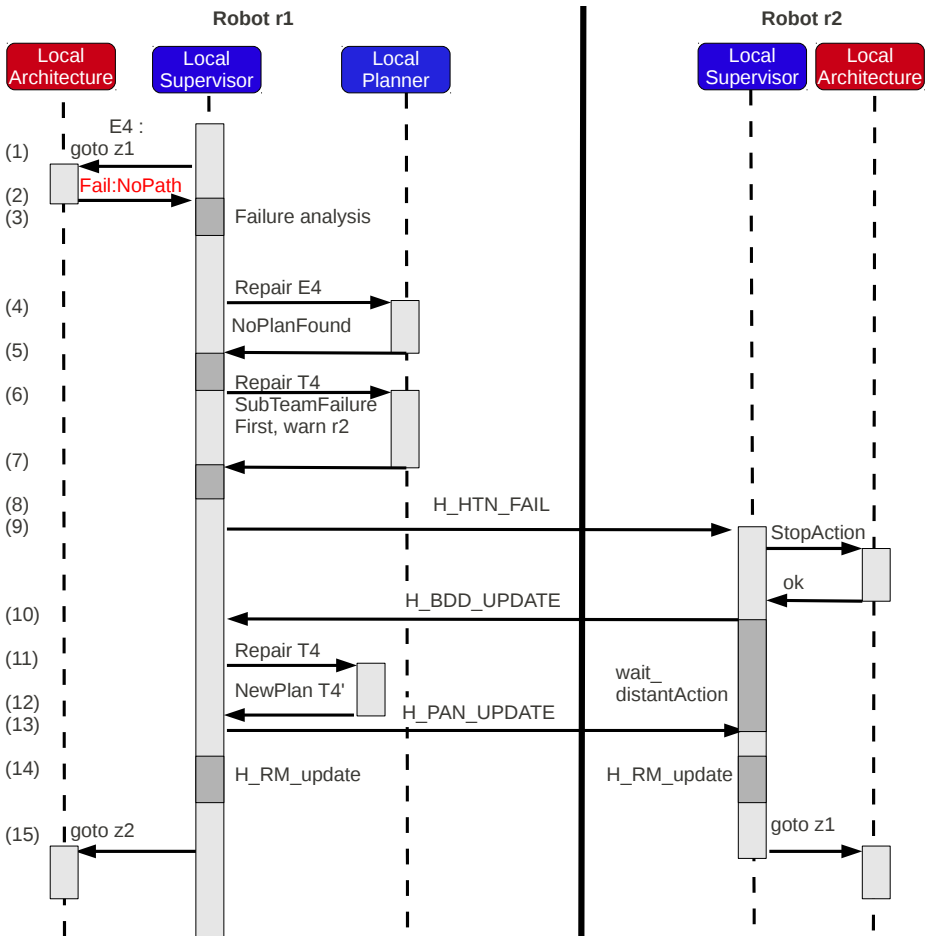


FIGURE 6.5 – Diagramme de séquence illustrant un processus de réparation d'équipe à deux véhicules, sans problème de communication. Lors de l'exécution de l'action "goto z1", l'architecture locale renvoie le retour d'exécution "Fail :noPath" (2). Ce retour est analysé (3) par l'*ExecMessenger* qui en informe le *MissionManager*. Un processus de réparation est donc lancé (4) pour $E4$ par le superviseur local de $r1$ (*PlannerManager*) pour tenter d'atteindre autrement le point $z1$, mais celui-ci échoue (5) (pas de chemin disponible pour atteindre le point d'intérêt). Le processus de réparation remonte par backtrack dans le HTNi et constate que $T4$ implique également $r2$ (7). Ce véhicule doit donc être prévenu de la réparation en cours (8). $r2$ stoppe l'activité qu'il était en train d'effectuer (9), et effectue une mise à jour des connaissances de $r1$ (10). $r1$ peut ainsi lancer son processus de réparation sur $T4$ (11), obtenir un plan alternatif pour $T4$ (12). Il transmet la mise à jour du plan (et de ses connaissances) à $r2$ (13). Les deux véhicules peuvent alors mettre à jour leur plan courant (14) et continuer l'exécution de la mission (15).

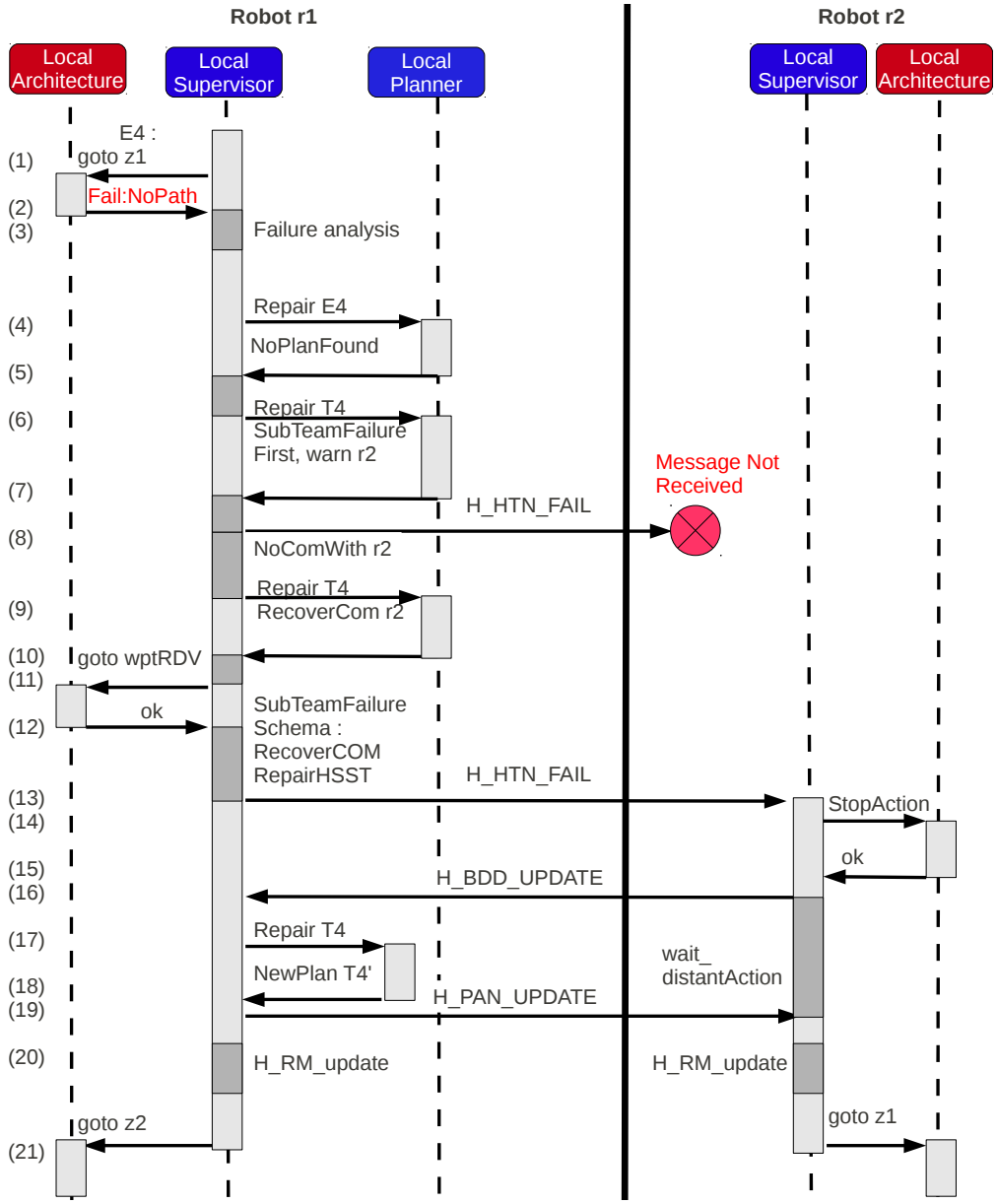


FIGURE 6.6 – Diagramme de séquence illustrant un processus de réparation d'équipe à deux véhicules, avec un échec d'une tâche de communication. Le déroulement initial est identique à celui du diagramme précédent (figure 2.5) jusqu'à l'étape (7). A ce stade, la communication échoue. Un nouveau processus de réparation est donc appelé sur cette tâche de communication (9), pour tenter de trouver un moyen d'établir une liaison avec $r2$. Le nouveau plan de $r1$ consiste à se diriger d'abord vers un point de rendez-vous (11), puis de retenter la communication (13). Cette fois, on suppose que celle-ci est réussie, et $r2$ est averti de l'aléa en cours. Le reste de la démarche rejoint ainsi la précédente.

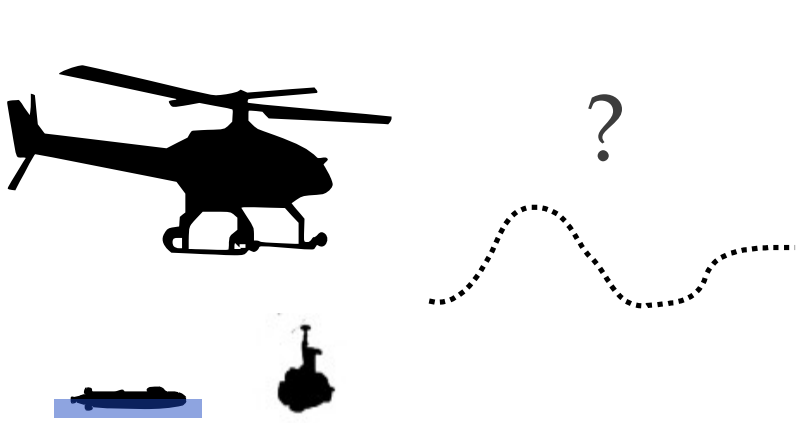
6.4 Conclusion

Nous avons vu qu'il était possible de détecter des aléas au niveau du *MissionManager* (section 5.5, p.87), et nous avons explicité dans ce chapitre la stratégie que nous avons choisi pour y faire face. En se basant sur sa structure hiéarchique, le HTNi est réparé aussi localement que possible. Cela procure le double avantage de limiter à la fois le coût de replanification et le nombre de véhicules impliqués dans le processus, et donc les communications. Lorsqu'un aléa implique plus d'un véhicule, une phase de synchronisation que nous avons décrite est alors indispensable entre les véhicules concernés pour mener à bien la réparation requise.

Tout au long des quatre derniers chapitres, nous avons décrit l'architecture HiDDeN et son fonctionnement. Nous disposons de Koper en tant que base de connaissances, qui nous permet de formaliser les données nécessaires à l'interaction entre HiDDeN et les architectures monovéhicules, mais aussi d'aider à la génération de requêtes de planification, notamment pour les réparations de plans effectuées en réponse à la détection d'un aléa par notre couche décisionnelle distribuée. Dans le chapitre 7 (p.105) nous présentons la mise en oeuvre concrète du système HiDDeN, au travers de simulations destinées à valider son fonctionnement et son comportement, mais aussi sur les expérimentations réelles que nous avons eu la chance de mener dans le cadre du PEA Action. Outre de nombreuses simulations des deux premiers scénarios, les différentes phases du scénario I ont notamment pu être expérimentées sur le terrain, validant les possibilités de coopération entre l'AAV ReSSAC de l'Onera à voilure tournante, et le rover AGV Mana du LAAS pour explorer une zone rurbaine à la recherche d'un intrus.

CHAPITRE 7

Simulations et expérimentations



Nous décrivons dans ce chapitre les différentes évaluations que nous avons menées sur le système HiDDeN, les simulations que nous avons pu effectuer pour les scénarios I et II du PEA Action ainsi que sa mise en œuvre expérimentale pour le scénario I de ce même PEA.

Sommaire

7.1 Métriques et évaluation

7.1.1 Des critères très différents selon les architectures

L'évaluation d'une architecture multirobot dépend des objectifs pour lesquels elle a été conçue. Comparer deux architectures ne répondant pas aux mêmes besoins n'est donc pas pertinent dans la mesure où le spectre des critères à évaluer est trop général et trop lié au cadre de déploiement. Diverses métriques ont été proposées dans la littérature pour évaluer les architectures en termes de robustesse, de modularité (intégration d'un nouvel élément), de généricité (par rapport à un type de mission ou d'engin), d'indépendance par rapport aux logiciels décisionnels, de facilité d'interfaçage avec des systèmes existants, de fiabilité, de documentation fournie...

Le critère de MTBF (Mean Time Between Failures) est régulièrement utilisé pour mesurer la tolérance aux fautes de systèmes robotiques [Bjerknes and Winfield, 2010, Crestani and Godary-Dejean, 2012]. Dans notre cas, ce critère n'est pas pertinent, puisqu'il rend compte de l'état global du système robotique, et pas seulement de l'architecture multirobot qui le contrôle. Les fautes qui apparaissent au niveau des architectures monovéhicules ou du planificateur par exemple seraient prises en compte, alors qu'elles peuvent être indépendantes du fonctionnement de HiDDeN. Les expérimentations du scénario I du PEA Action illustrent la sensibilité du système déployé. Il se compose de plusieurs modules logiciels en cours de développement dont les fautes non gérées localement compromettent presque systématiquement la mission, sans compter les erreurs d'origine humaine (section 7.4.5). En se basant sur des analyses quantitatives, des auteurs [Stancliff et al., 2005] affirment même que dans certains cas, pour deux équipes de robots, l'une composée d'un grand nombre de robots, et l'autre d'un nombre plus réduit de robots mais dont ces derniers seraient individuellement plus fiables, la première équipe (la plus large) permet pourtant d'accomplir des missions avec plus de fiabilité.

Des alternatives au MTBF ont été proposées. La méthode HWB (Hamilton-Walker-Bennett [Hamilton et al., 1996]) évalue la robustesse d'un système, mais en se basant sur sa redondance, ce qui n'est pas adapté à nos expérimentations où l'intérêt du système multirobot concerne plus la complémentarité que la redondance des services utilisés. Kannan et Parker [Kannan and Parker, 2007] proposent d'évaluer à la fois l'efficacité (la capacité du système à optimiser temps et ressources pour accomplir les tâches dont il a la charge), la robustesse (la capacité du système à identifier l'apparition de fautes et à s'en rétablir), et l'apprentissage (la capacité à s'adapter aux changements de l'environnement opérationnel et à extraire les informations pertinentes résultant de l'exécution d'une tâche). Encore une fois, le système est évalué dans sa globalité, ce qui rend l'évaluation totalement dépendante de l'ensemble des véhicules déployés. Les auteurs des architectures présentées en début de mémoire (chapitre 2, p.17) proposent d'autres types d'évaluation. Ainsi, des expérimentations réelles sont la preuve que de tels systèmes sont implémentables et ont été testés en environnement réel [Tambe, 1997, Parker, 1998, Brummit and Stentz, 1998, Dalgalarrondo, 2001, Vidal et al., 2002, Parker and Tang, 2006, Sotzing et al., 2008, Doherty et al., 2009]. Les traces de déplacement des véhicules pour les expérimentations comme pour les simulations sont régulièrement employées [Chaimowicz et al., 2001, Vidal et al., 2002]. D'autres métriques sont aussi exploitées, comme l'efficacité moyenne du système en fonction de la qualité des communications [Sotzing et al., 2008]. Pour évaluer des plans de mission, certains quantifient le nombre moyen d'actions nécessaires pour atteindre un but dans un domaine spécifique [Fazil Ayan et al., 2007], ou encore la durée des replanifications successives pour une même mission, ainsi que la qualité des plans obtenus [Fox et al., 2006].

7.1.2 Evaluation de HiDDeN par rapport à nos spécifications

Nous avons défini un ensemble de spécifications auxquelles l'architecture HiDDeN doit répondre dans la section 3.1 du chapitre 3 (p.47). A travers l'intégration des services proposés par les différentes architectures des scénarios I et II du PEA Action dans la base de données Koper, nous démontrons ci-après la compatibilité du système de supervision avec des architectures monovéhicules hétérogènes (section 7.2, p.121). Celles-ci voient leurs capacités individuelles, qui étaient déjà opérationnelles, exploitées dans le cadre d'une mission d'équipe (section 7.4.5, p.139), tout en conservant leur architecture monovéhicule d'origine.

L'occurrence d'aléas relevant du niveau décisionnel auquel se situe HiDDeN est prise en compte, que ce soit au niveau de leur détection (section 5.5 du chapitre 5, p.87) ou au niveau des réparations de plan qu'ils occasionnent (chapitre 6, p.89). Des schémas de recouvrement des communications sont par ailleurs proposés pour prendre en compte des cas où la réparation de plan impacterait différents membres de l'équipe, requérant alors une synchronisation des véhicules concernés (section 6.3 du chapitre 6, p.99). A travers les simulations suivantes (section 7.1.3, p.108 et section 7.1.4, p.111) nous analysons le comportement de l'architecture HiDDeN face à l'occurrence d'aléas qui se doivent d'être pris en compte par la couche décisionnelle.

7.1.3 Simulation du benchmark "Logistics"

Nous avons souhaité comparer notre approche distribuée de la supervision à une approche centralisée pour évaluer leurs besoins respectifs en terme de communication lors de l'occurrence d'aléas, et en particulier leur mise en œuvre dans des environnements où ces communications seraient contraintes. Nous avons également simulé des scénarios où intervenaient un plus grand nombre de véhicules que ce qui nous est actuellement permis par les contraintes expérimentales. Ces simulations restent cependant purement académiques, étant réalisées avec des architectures émulées, dans un simulateur considérant les actions des véhicules (autres que les communications) de façon déterministe.

Domaine de planification "Logistics"

Nous avons choisi le domaine "Logistics", benchmark standard pendant plusieurs années de l'IPC (International Planning Competition), comme cadre de nos simulations. L'environnement est décomposé en villes, elles-mêmes décomposées en places. L'objectif est d'amener chaque paquet de sa place initiale vers une place de destination qui lui est propre. Les paquets peuvent être chargés à leur place initiale et déchargés sur d'autres places. Dans ces simulations, nous ne nous intéressons qu'aux avions, qui seront nos véhicules autonomes : ils se déplacent entre des places spécifiques de chaque ville, les aéroports (avec l'hypothèse d'un aéroport par ville). Pour générer un plan hiérarchique, nous considérerons que des groupes et sous-groupes de paquets peuvent être livrés à leur destination par des groupes et sous-groupes d'avions. Nous introduisons des aléas lors de l'exécution du plan de mission : un avion peut ne pas pouvoir emprunter certaines voies aériennes entre deux aéroports [1]. Une réparation de plan est alors nécessaire, et peut être menée localement, ou impliquer une replanification totale du plan de mission. Pour définir la notion de disponibilité des communications, nous introduisons une portée de communication ρ entre les aéroports, qui reflète

1. Nous ne souhaitons pas prendre en compte le cas où il n'existe pas de solution au problème, par exemple un paquet devant être amené vers une ville qu'aucun avion ne pourrait atteindre, ou des cas où une ville serait totalement isolée.

la possibilité que des messages puissent être envoyés vers une ville pour une action de communication donnée. Cette portée indique les villes avec lesquelles un avion peut communiquer, et donc les avions distants avec lesquels il pourra échanger des données (la communication avec un avion en mouvement étant fournie par la ville d'arrivée). $\rho = 0$ signifie qu'aucune communication ne passe, tandis que $\rho = 1$ indique que toutes les actions de communication tentées aboutiront. Par exemple, $\rho = 0.3$ signifie qu'une ville peut communiquer avec 30% des autres villes. Au sein d'une même ville, les communications sont toujours disponibles. Cependant, la planification initiale ne prend pas en considération l'impossibilité de communiquer, qui n'est révélée que lors de l'exécution de la mission, ce qui impliquera une fois encore des réparations de plan.

Simulation

Nous comparons deux types d'approches : l'une centralisée, l'autre décentralisée. Dans l'approche centralisée, les fonctions de planification ne sont disponibles que dans un module centralisé, situé dans une ville définie. En cas d'aléa, le véhicule l'ayant subi doit communiquer avec ce module central. Si la communication n'est pas disponible, il doit se déplacer vers d'autres villes jusqu'à ce qu'il puisse établir un contact. Ces déplacements sont considérés comme des pénalités dues à l'absence des communications. Une fois le contact établi avec ce module central, celui-ci effectue une replanification totale de la mission, pour l'ensemble des véhicules, qui pourront à l'issu de celle-ci continuer l'exécution en utilisant le nouveau plan. Dans un souci de simplification de l'implémentation, le module central peut envoyer des informations à tous les véhicules, quelle que soit leur localisation, et sans se soucier des communications. Le module central a donc une vue globale très avantageuse de l'ensemble du système.

Dans l'approche décentralisée, nous exploitons les stratégies décrites tout au long de ce mémoire : lors de l'occurrence d'un aléa, le véhicule tente d'effectuer la réparation aussi localement que possible. Ici, les aléas concernent l'impossibilité d'effectuer un mouvement entre deux villes. La réparation consiste donc à trouver un nouveau chemin pour atteindre la destination. S'il n'en existe pas, un backtrack a lieu au sein du HTNi. Le véhicule doit communiquer avec les avions du sous-groupe le plus restreint pour effectuer une réparation de plan d'équipe, par exemple pour échanger leurs paquets. Si la réparation n'est pas possible à ce niveau, un nouveau backtrack a lieu, qui impliquera qu'un groupe d'avions plus important soit pris en compte pour des échanges de paquets potentiels. Les avions peuvent communiquer suivant les contraintes définies ci-dessus. Plus la portée des communications est limitée, et moins le nombre d'agents disponibles pour une réparation est grand, ce qui entraine moins de possibilités pour des réparations éventuelles.

Nous avons défini un problème de 10 aéroports, 72 paquets et 5 avions. Les localisations des paquets et des avions sont distribuées au hasard entre les villes, tout comme la destination des paquets. Nous avons fait varier ρ de 100% à 10%, et le nombre d'aléas "voie aérienne ne pouvant être empruntée pour un avion" de 0 à 20 par tranche de maximum 5. Nous avons choisi d'utiliser un planificateur "sur étagère", JSHOP (une implémentation Java de SHOP, prédécesseur à SHOP2) [Ilghami and Nau, 2001] qui nous a permis d'effectuer les premiers tests de planification initiale et d'appels de replanification.

Résultats

Le tableau 7.1 montre les résultats obtenus. Les nombres en gras illustrent un exemple où la portée des communications est de 30% (les villes peuvent communiquer avec moins d'un tiers des autres villes), 10 aléas ont été rencontrés. L'approche centralisée a requis 180 actions pour atteindre

les objectifs de mission, contre 171 pour l'approche décentralisée (5% de moins pour cette dernière). Si l'on ne considère que les actions supplémentaires qui n'étaient pas dans le plan nominal, le ratio passe à 32% de tâches supplémentaires pour l'approche centralisée. En moyenne par rapport à la portée des communications, les deux approches nécessitent un nombre comparable d'actions (lignes grisées claires). En revanche, lorsque la portée des communications passe de 60 à 50%, l'approche décentralisée devient moins coûteuse en termes d'actions élémentaires que l'approche centralisée (cellules grisées foncées).

TABLEAU 7.1 – Nombre d'actions nécessaires pour atteindre l'objectif de mission (tous les paquets sont à la bonne place) en fonction du nombre d'aléas et de la portée des communications ρ. Le nombre d'actions en nominal (c'est-à-dire sans aléa) N est de 143 dans les deux cas. C (respectivement D) indique le nombre d'actions nécessaires à l'approche centralisée (respectivement décentralisée) pour atteindre l'objectif de mission. $\frac{C-N}{D-N}$ représente le nombre d'actions supplémentaires dues aux replanifications pour l'approche centralisée par rapport à celui de l'approche décentralisée.

ρ	5 aléas			10 aléas			15 aléas			20 aléas			Moyenne		
	C	$\frac{C}{D}$	$\frac{C-N}{D-N}$	C	$\frac{C}{D}$	$\frac{C-N}{D-N}$	C	$\frac{C}{D}$	$\frac{C-N}{D-N}$	C	$\frac{C}{D}$	$\frac{C-N}{D-N}$	C	$\frac{C}{D}$	$\frac{C-N}{D-N}$
1	151	0.96	0.53	157	0.92	0.5	169	0.88	0.54	179	0.87	0.57	164	0.90	0.54
0.9	154	0.97	0.73	164	0.96	0.75	178	0.93	0.73	188	0.91	0.71	171	0.94	0.72
0.8	154	0.97	0.73	167	0.98	0.86	184	0.96	0.85	194	0.94	0.81	174.75	0.96	0.82
0.7	154	0.97	0.73	171	1.00	1.00	187	0.98	0.92	197	0.96	0.86	177.25	0.98	0.88
0.6	154	0.97	0.73	171	1.00	1.00	190	0.99	0.98	207	1.00	1.02	180.5	0.99	0.97
0.5	157	0.99	0.93	174	1.02	1.11	193	1.01	1.04	213	1.03	1.11	184.25	1.01	1.06
0.4	160	1.01	1.13	180	1.05	1.32	199	1.04	1.17	222	1.08	1.25	190.25	1.05	1.22
0.3	160	1.01	1.13	**180**	**1.05**	**1.32**	199	1.04	1.17	222	1.08	1.25	190.25	1.05	1.22
0.2	163	1.03	1.33	183	1.07	1.43	205	1.07	1.29	231	1.12	1.4	195.5	1.08	1.35
0.1	163	1.03	1.33	183	1.07	1.43	205	1.07	1.29	231	1.12	1.4	195.5	1.08	1.35
Moy.	157	0.99	0.93	173	1.01	1.07	190.9	0.99	0.99	208.4	1.01	1,04	182.33	1.00	1.00
D	158			**171**			191			206			181.75		
N	143														

Pour les deux approches, plus le nombre d'aléas augmente, et plus le nombre d'actions nécessaires à l'accomplissement de l'ensemble des objectifs augmente, ce qui était prévisible. Pour l'approche centralisée, le nombre d'actions augmente également avec la réduction de la portée des communications. En revanche, l'approche décentralisée ne dépend pas des portées de communication. Dans cette approche en particulier, une réparation locale (un chemin alternatif) pouvait être systématiquement trouvé par le véhicule, qui n'avait pas besoin d'effectuer de réparation d'équipe. Par contre, contrairement à l'approche centralisée, l'absence d'une voie aérienne n'est pas prise en compte pour les exécutions futures, ce qui entraîne l'occurrence du même aléa lorsqu'un avion tente d'emprunter cette voie. La figure 7.1 illustre un phénomène intéressant obtenu à partir du tableau 7.1 : moins la portée des communications est élevée, et plus l'approche décentralisée s'avère intéressante en terme de nombre d'actions exécutées. Les courbes de la figure 7.1 passent en moyenne au dessus de 1 (ligne rouge) lorsque la portée de communications devient inférieure à 60% : le coût en terme

de nombre d'actions pour "s'approcher à portée de communication" devient plus élevé que le coût d'emprunter des chemins alternatifs (non optimaux) entre les places suite aux réparations locales.

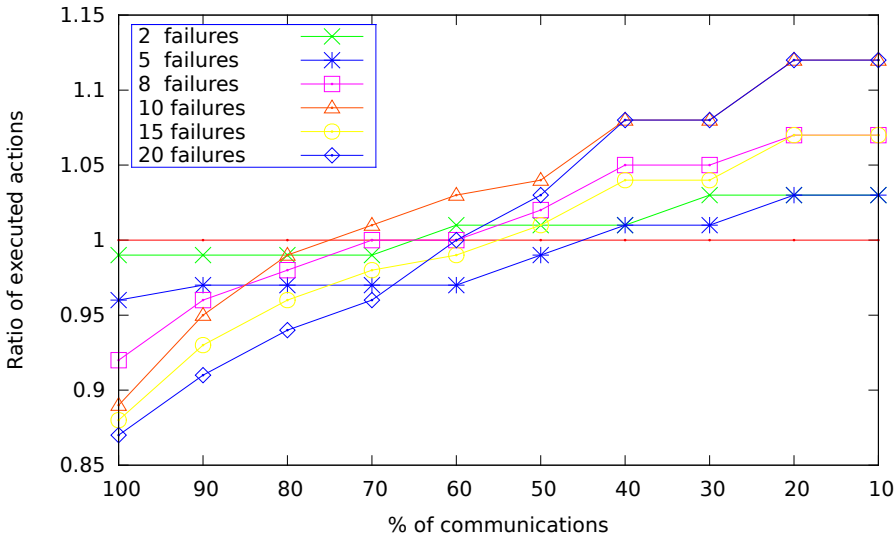


FIGURE 7.1 – Evolution du rapport, pour un nombre d'aléas donné, entre le nombre d'actions exécutées pour atteindre l'objectif de mission par l'approche centralisée et le nombre d'actions exécutées pour atteindre l'objectif de mission par l'approche décentralisée : $\frac{C}{D}$.

Cette simulation a également permis de tester une mission menée par cinq superviseurs locaux (un par avion), et non pas uniquement par deux superviseurs locaux comme dans le cadre des scénarios I et II du PEA Action.

Au début de nos travaux et pour ce benchmark "Logistics", nous avons utilisé JSHOP comme planificateur fournissant des plans HTNi au superviseur. Cependant, des comportements imprévus sous JSHOP, ainsi que la nécessité d'avoir certaines fonctionnalités telles que la conservation de la hiérarchie dans le plan de sortie et la définition de variables numériques nous ont poussés dans un premier temps à modifier JSHOP, puis à le délaisser au profit du planificateur AcPlan, développé pour les besoins du PEA Action.

7.1.4 Benchmark basé sur le scénario II du PEA Action

Domaine de planification

Le scénario II du PEA Action a servi de base pour mettre en place un benchmark destiné à évaluer le fonctionnement de HiDDeN. Deux véhicules sont impliqués dans ce scénario, l'AAV ReSSAC (véhicule aérien à voilure tournante) et l'AUV Daurade (véhicule sous-marin). L'objectif de la mission est de blanchir un chenal, c'est-à-dire vérifier que ce dernier ne contient pas de mines, pour permettre par exemple à un SNLE (Sous-marin Nucléaire Lanceur d'Engins) de quitter son port en sécurité.

Nous considérons que le chenal à blanchir est découpé en neuf zones à explorer. Ces zones peuvent contenir des bâtiments de surface qui doivent alors être "chassés" par l'AAV pour permettre à l'AUV d'entamer son exploration sous-marine à la recherche de mines. Si une mine est localisée, il doit remonter en surface pour en signaler la présence à un opérateur humain. S'il est trop loin

des côtes (ce qui est toujours le cas dans notre simulation) ou que la houle est forte, il doit utiliser un relais de communication, qui est un des rôles de l'AAV. Celui-ci a une autonomie en carburant limitée, et doit régulièrement se rendre sur une zone particulière qui sera qualifiée de "base" pour ses recharges énergétiques. Nous ne prenons pas en compte dans cette simulation le fait que l'AUV puisse se perdre. *A priori* les zones sont supposées vides de bâtiments de surface et de mines lors de la planification initiale. Leur présence est considérée comme un aléa lors de l'exécution de la mission. Le simulateur permet de placer des mines et des bâtiments de surface à volonté. Les architectures monovéhicules sont émulées, et permettent de gérer l'interaction entre les superviseurs locaux et l'environnement simulé. L'AAV peut se déplacer d'une zone à une autre, explorer une zone. Il peut donner son feu vert à l'AUV pour lui signaler qu'une zone est vide de bâtiments de surface et que ce dernier peut lancer son exploration sous-marine. Il peut aussi se poser sur sa base pour s'y ravitailler. Il peut également faire signe à un bâtiment de surface de dégager, et le suivre jusqu'à ce qu'il quitte le chenal pour vérifier que ce dernier est effectivement sorti. Les bâtiments de surface ont un comportement coopératif. Toutes ces actions ont un coût en terme de consommation de carburant, en prenant notamment en compte la taille de chaque zone, et la distance entre zones. Pour simplifier la représentation des zones, nous les plaçons sur une grille de 3x3, et nous ajoutons la base dans une case adjacente à une des cases de la grille.

L'AUV peut se déplacer entre les zones, plonger et les explorer à la recherche de mines. Il signale à l'AAV qu'il a terminé l'exploration d'une zone. Sa consommation énergétique n'est pas prise en compte, son autonomie énergétique étant plus élevée que celle de l'AAV. Lorsqu'il trouve une mine, il doit faire surface, signaler la présence de la mine qui est alors considérée comme traitée, et qui ne sera pas à nouveau détectée. Il reprend ensuite son exploration là où il l'avait laissée. Pour les besoins de cette simulation, nous générerons des aléas de communication par perte de messages. Avant de lancer une simulation, nous définissons la probabilité qu'une action de communication exécutée par l'un des véhicules échoue. Un pourcentage $pLoss$ indique ainsi la qualité des communications en terme de perte de messages. Par exemple, pour $pLoss = 0.3$, toute action de communication tentée dans la mission a 30% de chances d'échouer. Les actions de communications sont celles définies dans le plan, qui tient compte du fait que l'AUV ne peut pas communiquer lorsqu'il est en plongée, et que les deux véhicules ne peuvent pas communiquer s'ils sont trop éloignés l'un de l'autre. Si des actions de communication sont tentées malgré tout dans ces conditions, elles échouent systématiquement.

Nous distinguons trois types d'aléa :

- l'aléa "présence d'un bâtiment de surface" ne concerne que l'AAV : c'est donc un aléa monovéhicule ;
- l'aléa "présence de mine" lors de la localisation d'une mine, que l'AUV ne peux réparer seul : c'est un aléa multivéhicule.
- les aléas de communication, lorsqu'une action de communication échoue. Le schéma de recouvrement des communications (section 6.3 du chapitre 6, p.99) est le suivant : la première réparation que le véhicule tente consiste à répéter l'action. Si celle-ci échoue à nouveau, il tente de se déplacer vers la dernière position connue de l'autre véhicule, et retente une action de communication. Si elle échoue à nouveau, le véhicule rejoint un point de rendez-vous planifié et tente en boucle plusieurs actions de communication.

Simulation

Lors des simulations, nous faisons varier le pourcentage de messages de communication perdus de 0 à 90%, le nombre de bâtiments de surface présents de 0 à 6 et le nombre de mines à localiser

de 0 à 4. Lorsqu'au plus 2 entités (mines ou bâtiments de surface) étaient présents, nous avons pu lancer plus de cent simulations. Au-delà, au moins 10 simulations ont été lancées par configuration.

Planification et plans

La planification initiale a été effectuée une seule fois pour produire le plan HTNi utilisé. Une partie de celui-ci est visible sur la figure 7.2. La figure 7.3 montre la distribution associée à bord de l'AAV et de l'AUV. Un plan HTNi a déjà été présenté pour ce scénario II (figure 5.4 du chapitre 5, p.82). Cette première version visait une réduction de la durée d'exécution en modélisant une exploration parallèle des zones par les deux véhicules. Elle est cependant moins bien adaptée à la réparation (les fenêtres de rendez-vous doivent notamment être recalculées). Nous proposons donc d'utiliser un autre HTNi, présentant l'avantage de rendre l'équipe robuste face à l'occurrence d'aléas en hiérarchisant les tâches à effectuer par zone, et en traitant celles-ci les unes à la suite des autres. Dans cette deuxième version, une tâche de synchronisation est ajoutée par zone visitée lors de l'étape de distribution, pour chacun des deux véhicules (18 tâches qui requièrent des communications sont ajoutées) et les tâches conjointes de rendez-vous ne sont par conséquent plus nécessaires. Les tâches "void" sont des reliquats du processus de planification. Le nombre de tâches élémentaires est comparable à celui de la modélisation précédente (figure 5.4, p.82).

Contrairement aux premières simulations (section 7.1.3), nous avons utilisé cette fois le planificateur AcPlan, spécifiquement développé dans le cadre du PEA Action [Bigot, 2011, Gobillot, 2012]. C'est un planificateur HTN basé sur les algorithmes de SHOP2 [Nau et al., 2003]. L'un des avantages de ce planificateur concerne les formats utilisés, compatibles avec ceux exploités par HiDDeN.

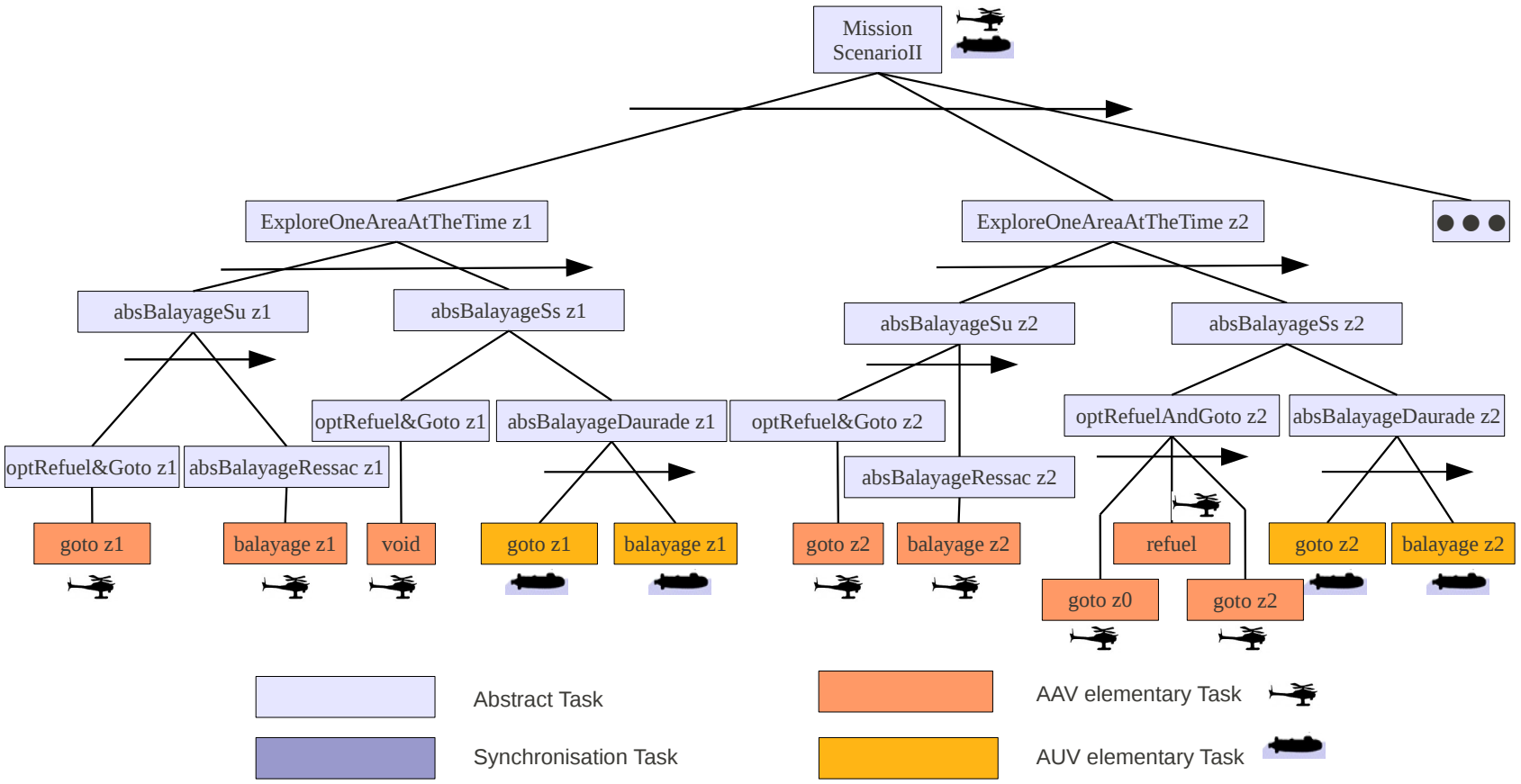


FIGURE 7.2 – HTNi partiel du scénario II du PEA action, version hiérarchique décomposée par zone (une seule zone traitée à la fois). La tâche abstraite $absBalayageSu$ (Su pour surface) désigne un balayage de zone en surface, incluant le déplacement pour rejoindre la zone, et un éventuel ravitaillement, explicité par la tâche abstraite $optRefuel\&Goto$. La tâche abstraite $absBalayageSs$ (Ss pour zone sous-marine) consiste pour l'AUV à faire un balayage sous-marin de zone, incluant le déplacement pour s'y rendre.

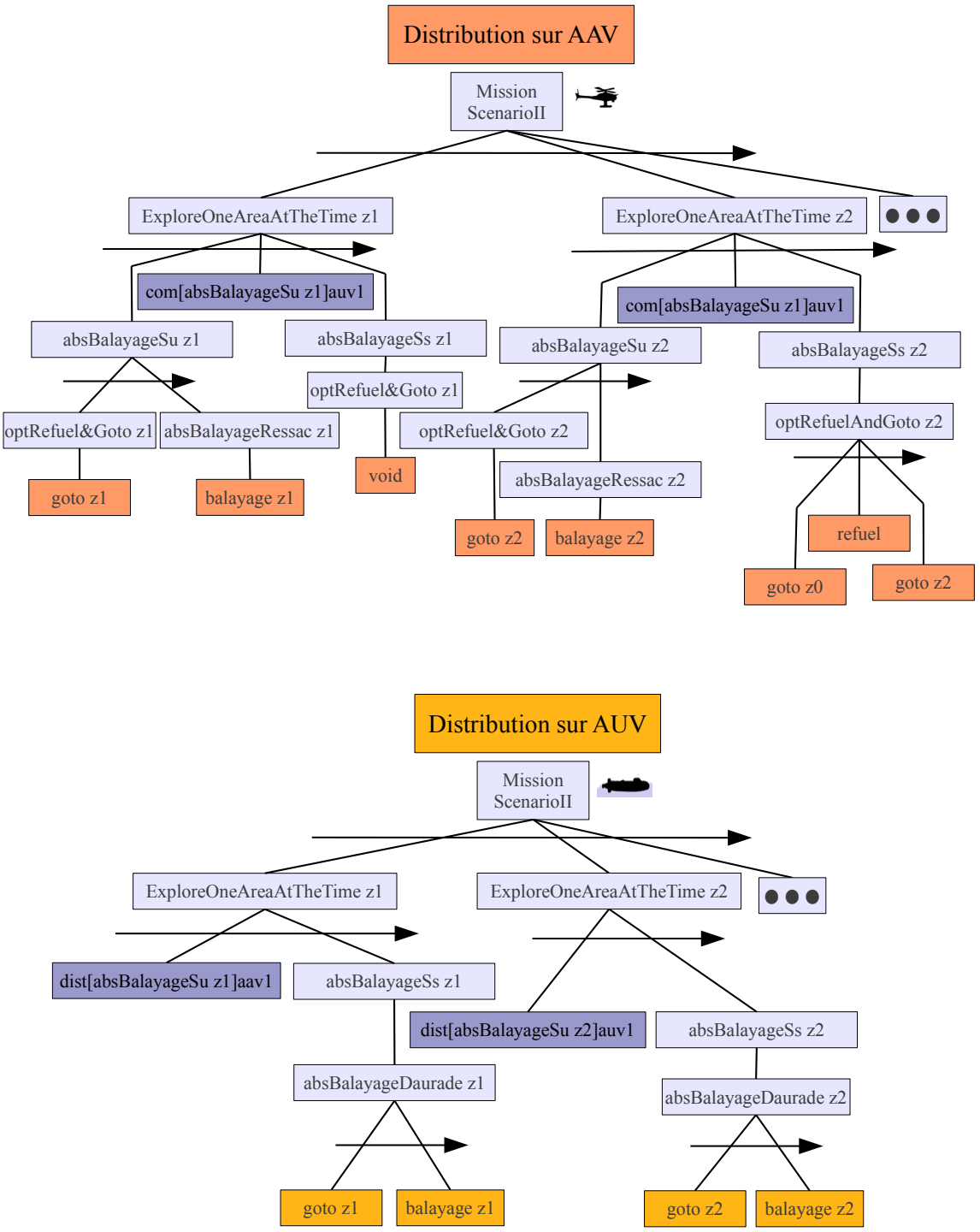


FIGURE 7.3 – HTNi partiel du scénario II du PEA action, version hiérarchique décomposée par zone (une seule zone traitée à la fois). Il est distribué sur le superviseur local de l'AAV (en haut) et sur celui de l'AUV (en bas).

Résultats

La figure 7.4 montre l'évolution du nombre moyen d'actions nécessaires à l'équipe pour atteindre les objectifs de mission, en fonction de la qualité des communications $pLoss$, et du nombre d'aléas "présence d'un bâtiment de surface" et "présence de mine" : la légende, Bx_My indique que l'équipe à rencontrée x bâtiments de surface et y mines.

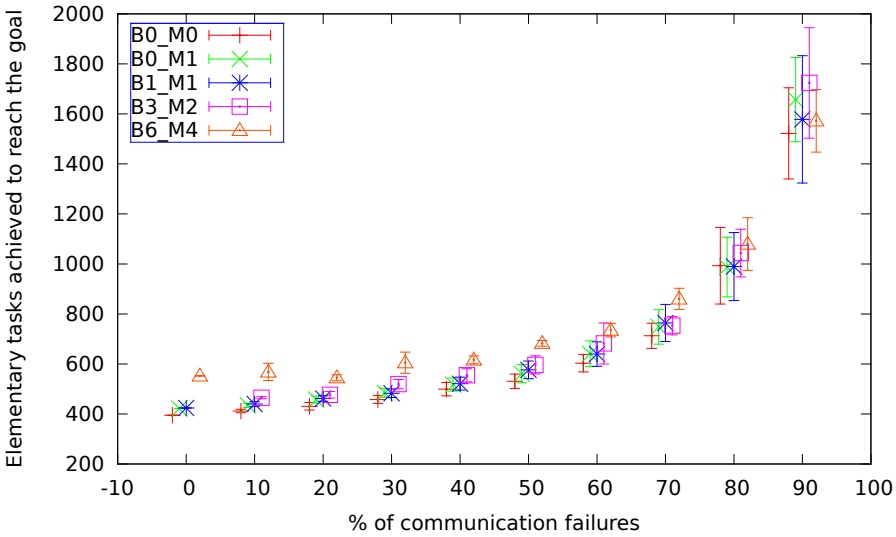


FIGURE 7.4 – Nombre moyen de tâches élémentaires (et écarts types associés) nécessaires à la réalisation de la mission en fonction de la probabilité d'échec des actions de communication. Le pas de variation du pourcentage de communication est de 10%, allant de 0 à 90% pour chacune des configurations.

La figure suivante (figure 7.5), avec la même légende, précise le nombre moyen de requêtes de planification que les superviseurs locaux ont dû effectuer pour accomplir la mission, en fonction de la qualité des communications $pLoss$.

Ces deux premières courbes étaient prévisibles : plus il y a d'aléas (tous types confondus), et plus le nombre de tâches élémentaires et de requêtes de planification nécessaires pour accomplir la mission est grand. Des tâches supplémentaires sont ajoutées lors des réparations successives pour permettre à l'équipe d'apporter des solutions face aux aléas.

La figure 7.6 est un nuage de points qui représente le nombre de tâches élémentaires accomplies par rapport au nombre d'aléas rencontrés sur l'ensemble des simulations. On peut constater que le nombre de tâches élémentaires exécutées pour atteindre les objectifs de mission croît linéairement par rapport au nombre d'aléas rencontrés. Ainsi, la réparation locale du plan permet d'accomplir la mission, au prix d'une sous-optimalité du plan résultant. Il est également intéressant de constater que notre approche est robuste à un grand nombre d'aléas survenant pendant l'exécution, et passe donc bien à l'échelle pour des missions complexes, en environnements dynamiques et incertains.

Intérêt de la hiérachie pour les réparations locales

Nous avons souhaité comparer les exécutions des deux types de HTNi par HiDDeN : celui déjà présenté figure 7.2 que nous noterons $\mathcal{H}_1$ et un HTNi "plat" (figure 7.7), où toutes les tâches élémentaires sont au même niveau et ont pour parente une unique tâche abstraite, la racine. Nous

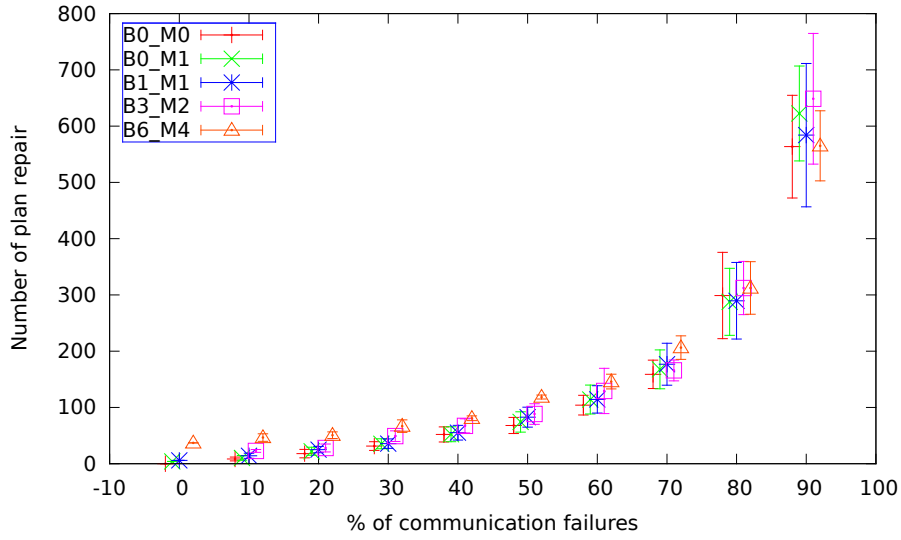


FIGURE 7.5 – Nombre moyen de requêtes de planification nécessaires à la réalisation de la mission en fonction de la probabilité d'échec des actions de communication. Le pas de variation du pourcentage de communication est de 10%, allant de 0 à 90% pour chacune des configurations.

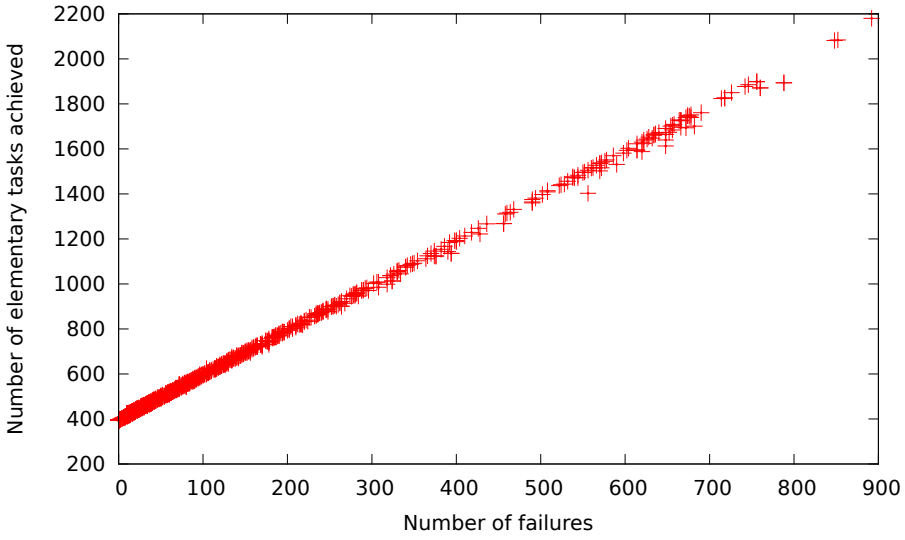


FIGURE 7.6 – Nuage de points du nombre de tâches élémentaires effectuées en fonction du nombre d'aléas rencontrés, sur l'ensemble des simulations lancées.

noterons ce dernier $\mathcal{H}_2$. Il s'agit d'un plan séquentiel, mis sous un format que HiDDeN est capable de gérer.

La première conséquence de cette différence de structure va concerner les requêtes de planification : lors de l'occurrence d'un aléa, pour un véhicule exécutant la version distribuée de $\mathcal{H}_2$, la réparation concerne systématiquement la tâche racine. Chaque requête de planification est une replanification globale de $\mathcal{H}_2$, et concerne tous les véhicules.

La figure 7.8 montre la durée qu'ont demandée les requêtes de planification en fonction du

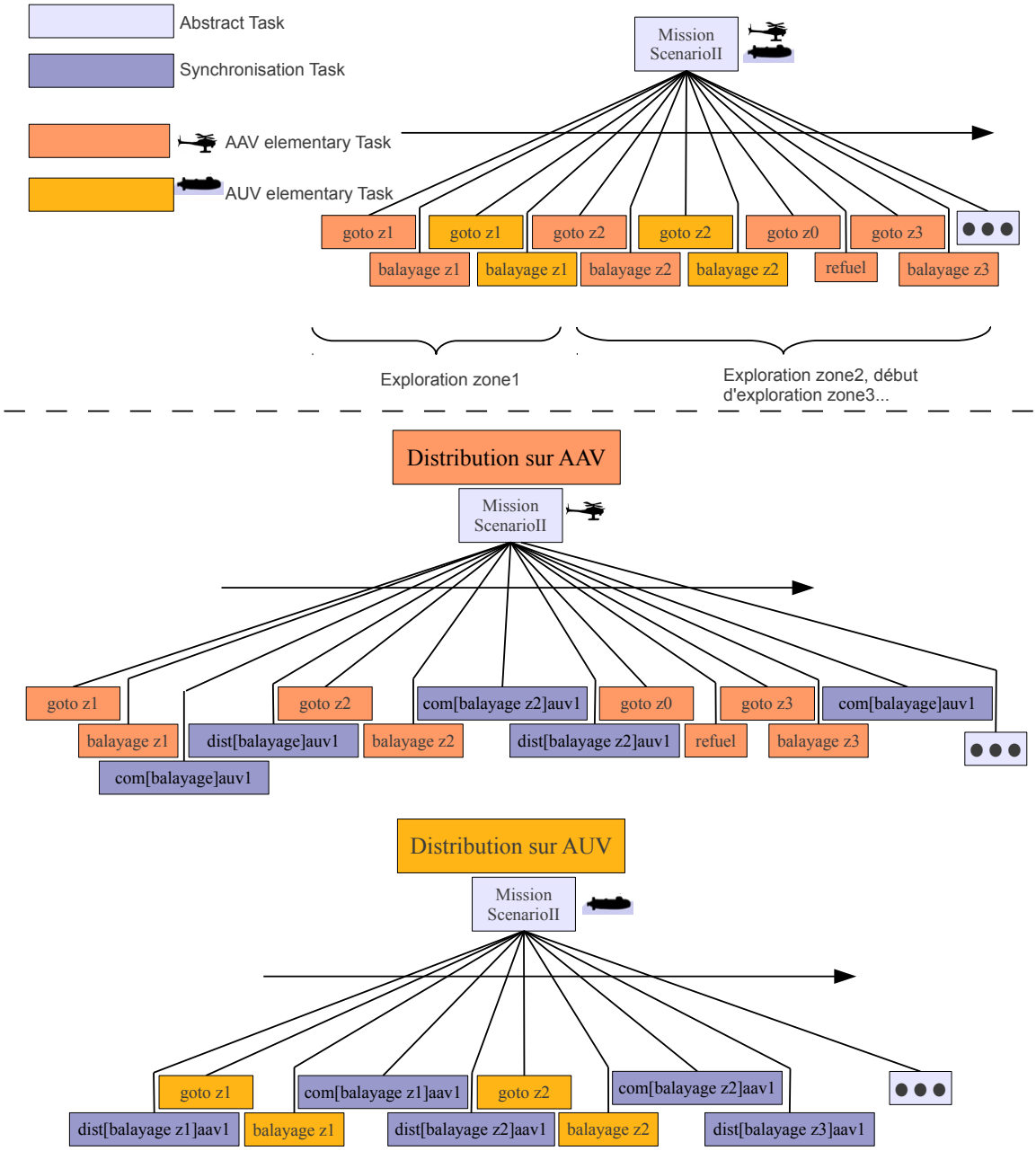


FIGURE 7.7 – Vue partielle d'une version non hiérarchisée du HTNi $\mathcal{H}_2$ du scenario II du PEA action.

nombre d'aléas rencontrés pour une série d'exécutions de $\mathcal{H}_1$ et de $\mathcal{H}_2$. Pour obtenir ces résultats, nous avons considéré que les communications étaient parfaites lors des exécutions de $\mathcal{H}_2$. Dans le cas de $\mathcal{H}_1$, seules les simulations dont le nombre d'aléas était inférieur à 10 apparaissent. On voit nettement la disproportion de la durée de planification requise en cas d'aléa entre les deux arbres

HTNi : même si celle-ci reste linéaire par rapport au nombre d'aléas rencontrés, le coût en temps des calculs pour obtenir les réparations du HTNi $\mathcal{H}_2$ sont considérables par rapport à celui qui est nécessaire aux réparations locales de $\mathcal{H}_1$. La figure figure 7.8 ne montre que des exécutions où se sont déroulés au plus dix aléas. Elle peut être mise en relation avec la figure 7.9 qui reprend l'ensemble des simulations exécutées en se basant sur l'arbre $\mathcal{H}_1$.

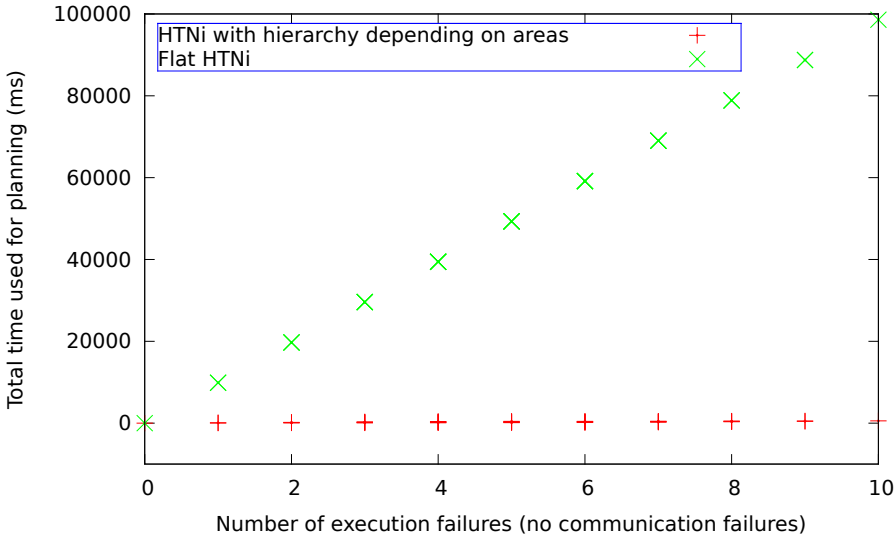


FIGURE 7.8 – Durées cumulées des replanifications successives en fonction du nombre d'aléas rencontrés, pour une exécution effectuée avec le HTNi "plat" $\mathcal{H}_2$ (Flat HTNi) et le HTNi hiérarchisé par zones $\mathcal{H}_1$ (HTNi with hierarchy depending on areas).

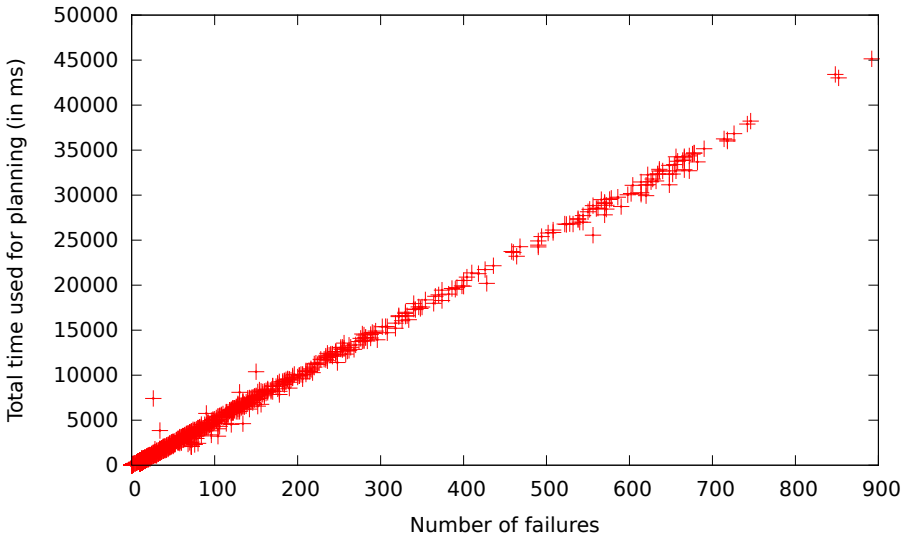


FIGURE 7.9 – Nuage de points des durées cumulées des replanifications successives en fonction du nombre d'aléas rencontrés, sur l'ensemble des simulations lancées avec $\mathcal{H}_1$.

La durée de planification augmente linéairement en fonction du nombre de requêtes de planifica-

tion effectuées pour $\mathcal{H}_1$ et $\mathcal{H}_2$. Mais le mécanisme de réparation locale limite la taille du problème à replanifier, ce qui maintient sa résolution à une durée moyenne d'environ 50 ms par aléa, à comparer aux 9910 ms nécessaires à la planification initiale [2] de $\mathcal{H}_1$. En revanche, $\mathcal{H}_2$ qui a demandé autant de temps en terme de planification initiale voit ses réparations demander des durées comparables à cette planification initiale : en effet, l'ensemble du HTNi $\mathcal{H}_2$ est remis en cause à chaque aléa. Dans certains cas, ce type d'approche pourrait être bénéfique, par exemple lorsque la mission est évaluée en terme de qualité de plan (critère de consommation de ressources à optimiser par exemple), et non de durée : une replanification totale pourrait être menée de façon à produire un plan optimal, alors qu'une réparation partielle du plan n'implique pas nécessairement cette propriété. En revanche et dans notre cas d'étude, nous cherchons plutôt à éviter la replanification totale, qui implique de synchroniser l'ensemble des connaissances de l'équipe, et donc de nombreuses communications.

La figure 7.10 compare les exécutions de deux types de HTNi ($\mathcal{H}_1$ et $\mathcal{H}_2$) en terme de messages échangés, en fonction du nombre d'aléas rencontrés. Pour les tracés concernant $\mathcal{H}_1$, nous avons pris la moyenne des simulations dans lesquelles les communications étaient parfaites. Dans le cas du plan séquentiel $\mathcal{H}_2$, le nombre moyens de messages échangés augmente linéairement par rapport au nombre d'aléas rencontrés. En effet, quelque soit le type d'aléa (monovéhicule, ou multivéhicule), la structure séquentielle du plan implique une replanification globale, et donc une synchronisation entre véhicules, tâche génératrice de messages à échanger.

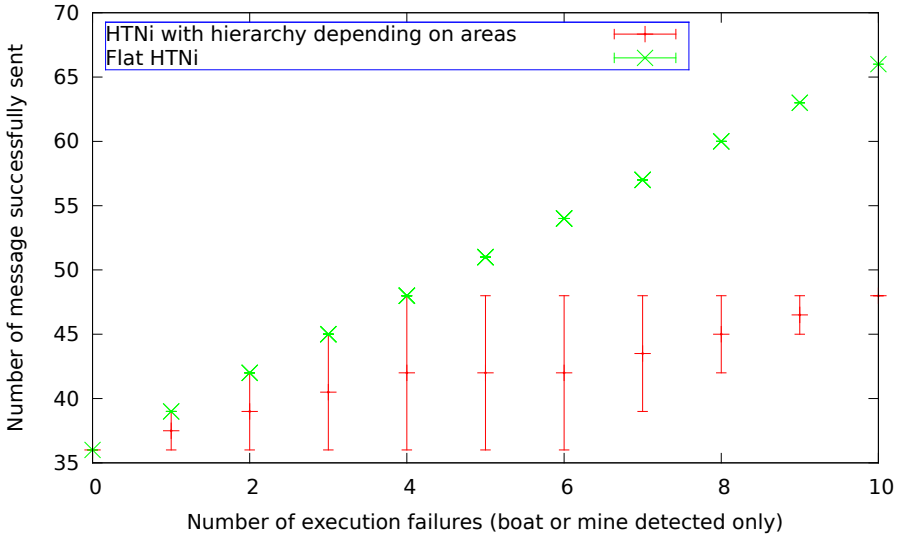


FIGURE 7.10 – Comparaison du nombre moyen de messages (avec l'écart type représenté) envoyés par aléa de type "présence d'un bâtiment de surface" ou de type "présence d'une mine" entre l'exécution du HTNi $\mathcal{H}_1$ (HTNi with hierarchy depending on areas), dont la hiérarchie s'organise suivant les zones, et un HTNi "plat" $\mathcal{H}_2$ (Flat HTNi), où toutes les tâches élémentaires sont au même niveau.

Par contre et en moyenne [3], moins de messages sont échangés lors des réparations pour l'approche basée sur $\mathcal{H}_1$. La réparation locale est particulièrement efficace à ce niveau, puisqu'elle

2. Processeur Intel® Xeon® de quatre cœurs cadencés à 2.67GHz, 6Go de RAM

3. L'importance des écarts types associées à $\mathcal{H}_1$ s'explique par le fait qu'il y ait au maximum quatre aléas de type "présence de mine" et six de type "présence de bâtiment de surface" dans nos simulations : par exemple, pour quatre aléas, il est équiprobable de tomber sur une à quatre réparations monovéhicules. Le comportement basé sur $\mathcal{H}_1$ est donc meilleur, ou au moins aussi bon dans les pires cas.

exploite la hiérarchie du plan $\mathcal{H}_1$: seuls les aléas multivéhicules nécessitent la génération de messages supplémentaires pour accomplir une réparation de plan. Les aléas monovéhicules sont traités localement, par chacun des superviseurs locaux.

La stratégie de réparation locale est fortement liée à la hiérarchie du HTNi considérée. La production de tels plans hiérarchisés nécessite une expertise humaine lors de l'écriture des domaines HTN, pour la description des tâches abstraites, et qui sert à la planification du HTNi. S'appuyer dessus est largement bénéfique en terme de durée de planification et du nombre de messages échangés entre véhicules dans le cas de notre benchmark, pour peu que le HTNi soit "suffisamment" hiérarchisé.

Le choix d'une structure hiérarchique nous a permis d'effectuer des réparations locales, qui s'avèrent bénéfiques par rapport à une version séquentielle du plans.

7.2 Intégration des véhicules du PEA Action dans Koper

7.2.1 Présentation générale

L'enjeu majeur de cette thèse est de disposer d'un système de supervision de mission pour la réalisation des missions du PEA Action. Les deux premiers scénarios ont été instanciés dans la base de connaissances Koper.

Les simulations peuvent être classées en deux catégories. La première consiste à émuler l'ensemble des systèmes, puis à vérifier le fonctionnement de HiDDeN. Un visualiseur a été ajouté par la suite pour faciliter les vérifications des placements et explorations des véhicules. La seconde, plus réaliste, repose sur le simulateur MORSE [4] [Echeverria et al., 2011,Echeverria et al., 2012], capable de déployer les véritables architectures des véhicules autonomes lorsqu'elles sont interfacées au simulateur. Nous avons de plus eu la chance de valider le superviseur sur le terrain à travers les expérimentations du scénario aéroterrestre I. Le scénario aéromaritime II a été évalué en simulation seulement, les expérimentations prévues étant planifiées ultérieurement à mon travail de thèse. Dans ce qui suit, nous présentons des aspects plus techniques dans la mise en œuvre des deux premiers scénarios. Nous décrivons notamment les informations sur lesquelles Koper s'appuie.

7.2.2 L'AAV ReSSAC

Description

L'AAV RMax/ReSSAC de l'Onera (figure 7.11) est un hélicoptère de marque Yamaha Motor, utilisé au Japon pour des traitements agricoles et plus particulièrement pour l'épandage de pesticides sur les cultures de riz. Il a reçu en juin 2007 l'autorisation par la DGAC de "Vol automatique" : vols avec hauteur inférieure à 70m, vitesse sol de déplacement inférieure à 20m/s, distance du pilote de sécurité inférieure à 200m, vent inférieur à 9m/s. Les charges expérimentales embarquées sur le RMax/ReSSAC et utilisées dans le cadre des scénarios du PEA Action sont :

- Un GPS RTK (précision en différentiel inférieure à 1m, en RTK de 2cm à 20cm) ;
- Une caméra vidéo couleur dirigée vers le bas ;
- Un télémètre laser de classe II pour la mesure de la hauteur sol entre 0m et 10m (portée 10m, précision 3mm) ;

4. http://www.openrobots.org/wiki/morse/

– Un lidar dirigé vers le sol utilisable pour la cartographie et pour le relevé de terrain (plan laser), et notamment pour la mesure de la hauteur sol entre 10m et 70m (portée 80m, précision 2cm).

FIGURE 7.11 – AAV RMax/ReSSAC de l'Onera.

L'AAV RMax/ReSSAC communique par support WLAN avec une station sol (figure 7.12) qui est composée :

– d'un poste de pilotage permettant à un opérateur de connaître en temps réel l'ensemble des paramètres du RMax/ReSSAC et de se rendre compte de son comportement ;
– d'un poste de navigation permettant la gestion du plan de vol et le suivi de la mission ;
– d'un écran de contrôle vidéo permettant de visualiser les informations de la caméra frontale couleur, dirigée vers le bas (non utilisée pour le traitement embarqué des données).

(a) Véhicule abritant la station sol. (b) Vue d'ensemble des Interfaces Homme-Machine de l'AAV RMax/ReSSAC.

FIGURE 7.12 – Station sol de l'AAV RMax/ReSSAC.

Architecture monovéhicule

HiDDeN doit s'interfacer avec l'architecture monovéhicule du RMax/ReSSAC basée sur Orocos [Teichteil-Königsbuch et al., 2011]. Orocos [5] (Open RObot COntrol Software project [Soetens and Bruyninckx, 2005]) est une librairie C++ pour la robotique, qui fournit un cadre de développement temps réel et modulaire. Elle permet de créer des composants distribuables, et de garantir des communications temps-réel et sécurisées entre ces composants. Chaque composant possède une interface standardisée qui décrit ses ports de données et les services qu'il fournit. Un composant peut réagir à des événements, traiter des requêtes, et exécuter des scripts en temps réel. La figure 7.13 illustre la complexité de l'architecture déployée dans le cadre des scénarios du PEA Action.

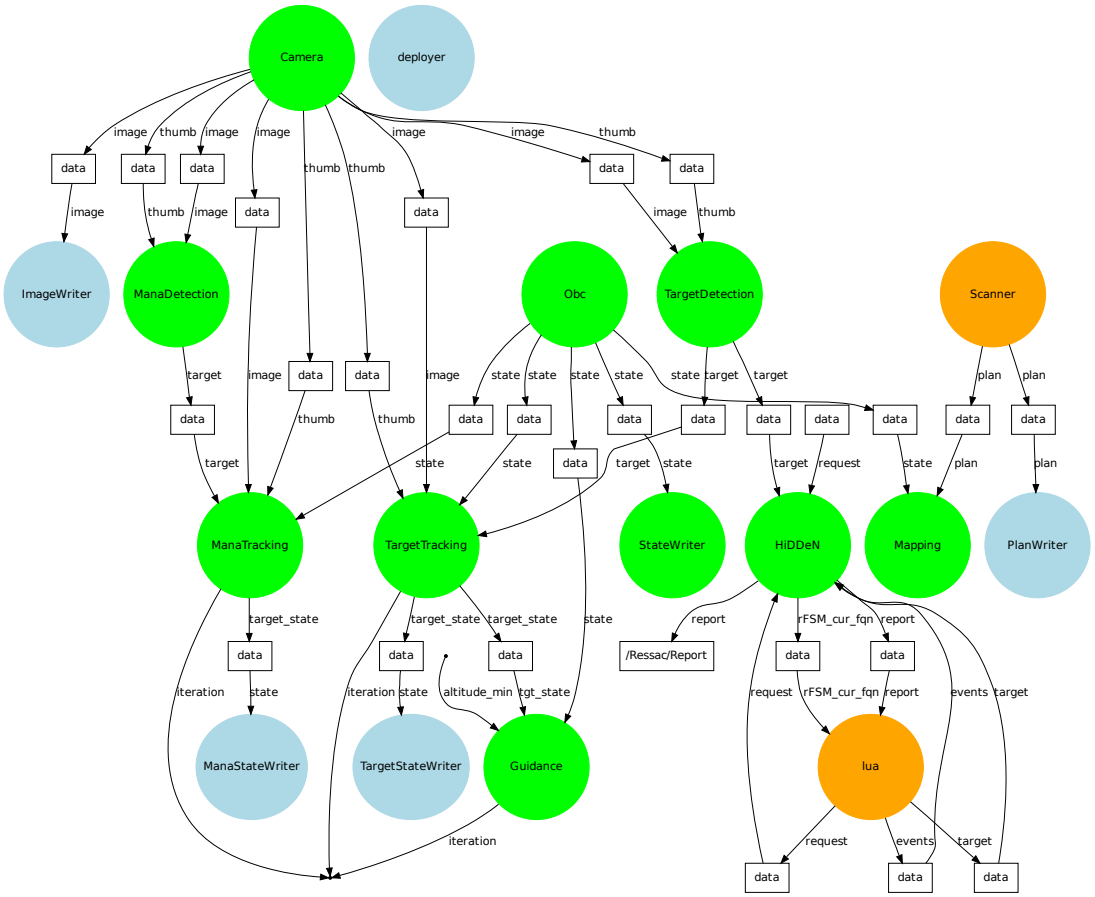


FIGURE 7.13 – Vue d'une instanciation des composants Orocos pour les scénarios du PEA Action.

Services - Description dans Koper

Le tableau 7.2 liste les différents services du RMax/ReSSAC intégrés dans Koper. La notion de service synchrone ou asynchrone est présentée dans la section 4.2.1 du chapitre 4 (p.57). Nous

5. http://www.orocos.org/

noterons pour un service S :

- S^O : Service effectué par un opérateur ;
- S^N : Service non exploité en expérimentation ;
- S^I : Service exploité dans le scénario I du PEA Action ;
- S^{II} : Service exploité dans le scénario II du PEA Action ;
- [in] : un paramètre attendu en entrée par le service du véhicule ;
- [out] : un paramètre fourni en sortie par le service du véhicule.

TABLEAU 7.2 – Listing des services de l'AAV RMax/ReSSAC. Les variables sont désignées par un " ?" suivi de leur nom.

Service	Type	Description	Paramètres
start^0	synchrone	Signal de départ de la mission	
stop^0	synchrone	Signal d'interruption de la mission	
takeOff^0	synchrone	Décollage du véhicule jusqu'à l'altitude fournie en paramètre	[in] ?alt
land^0	synchrone	Atterrissage du véhicule au waypoint désigné	[in] ?wptTo
goto^{I,II}	synchrone	Déplacement du véhicule vers le waypoint désigné	[in] ?wptTo
localiseTarget^{I,II}	synchrone	Le véhicule localise la cible désirée dans l'image actuelle	[in] ?targetName
track^{I,II}	synchrone	Le véhicule poursuit la cible préalablement localisée	[in] ?targetName
getGPSPoint^{I,II}	synchrone	Le véhicule renvoie sa position GPS	[out] ?wpt1
carto^I	synchrone	Le véhicule génére une carte, cartographie du terrain entre deux points, renvoyée dans un fichier	[in] ?wpt1 [in] ?wpt2 [out] ?file
refuel^N	synchrone	Ravitaillement du véhicule, repos de l'équipe	
balayage^{II}	synchrone	Balayage par le véhicule d'une zone (rectangle du point haut-gauche, et point bas droite) à la recherche de bâtiments de surface	[in] ?wptHG [in] ?wptBD
repriseBalayage^{II}	synchrone	Reprise de balayage sur la zone en cours	
signeBoat^{II}	synchrone	Le véhicule fait un signe au bâtiment de surface pour lui faire évacuer la zone	
stationnaireCOM^{I,II}	synchrone	Le véhicule reste en stationnaire en vue de la réception d'un message	
targetDetectionOn^{I,II}	asynchrone	Le véhicule dispose d'une détection permanente d'une cible particulière	[out] ?targetName [out] ?wptTarget

7.2.3 L'AGV Mana

Description

L'AGV Mana du LAAS-CNRS (figure 7.14) est construit sur une plate-forme de type "Segway RMP400", constituée de 4 roues non directionnelles et non suspendues. Cette plate-forme peut embarquer une charge utile de 90kg, et se déplacer à une vitesse maximale de 8m/s, pour 10 à 24 km d'autonomie selon les conditions de terrain. L'ensemble des capteurs ont été séparés en deux groupes, associés aux deux calculateurs embarqués :

- Un PC basé sur un processeur Pentium M cadencé à 1.4 GHz solidaire du châssis qui reçoit l'ensemble des capteurs dits "bas-niveaux" : gyromètre (modèle e-core 1000 de KVH), centrale inertielle de type "mems" (modèle MTi de xSens), odomètres du châssis (c'est via ce PC que transitent les commandes de déplacement du châssis) et une interface Ethernet dédiée qui permet de recevoir les données du télémètre Velodyne, qui sont principalement exploitées pour la modélisation de l'environnement.
- Un PC portable (processeur Core-2 Duo cadencé à 2.2 GHz) qui est dédié aux traitements de vision : les deux caméras qui constituent le banc stéréoscopique monté sur une tourelle orientable (modèle PTU D46-17 de Directed Perception) y sont connectées. Les traitements de localisation par SLAM visuel sont effectués par ce calculateur. Enfin, la centrale inertielle xSens y est raccordée pour la fusion des données vision / inertiel.

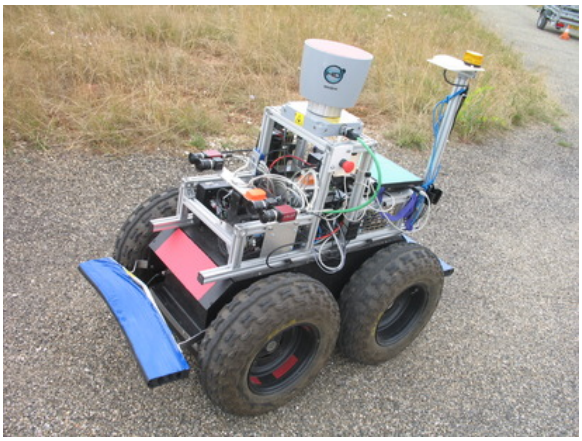


FIGURE 7.14 – L'AGV Mana du LAAS-CNRS.

Architecture monovéhicule

L'architecture monovéhicule de l'AGV est organisée par couches. Une couche décisionnelle monovéhicule assure la supervision individuelle de l'AGV et une couche commande du véhicule permet de contrôler les déplacements de l'AGV. La couche monovéhicule est également constituée d'un ensemble de composants logiciels implémentés sous la forme de modules GenoM (outil qui permet de structurer un ensemble de fonctions logicielles en modules dont les interfaces de contrôle et de données obéissent à une description standardisée [Fleury et al., 1997]). L'ensemble de ces composants est supervisé selon le formalisme d'agents-ressources ROAR [Degroote and Lacroix, 2011].

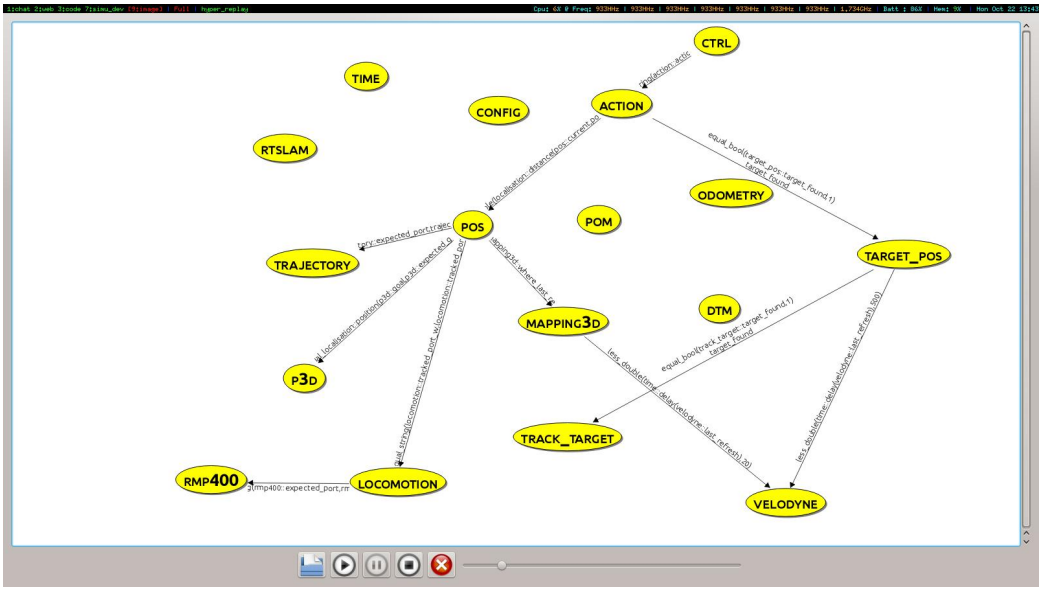


FIGURE 7.15 – Ensemble de modules GenoM supervisés par ROAR. Ici, la configuration nominale de l'AGV Mana pour le scénario I du PEA Action.

Services - Description dans Koper

Le tableau 7.3 liste les différents services de l'AGV Mana intégrés dans Koper.

TABLEAU 7.3 – Listing des services de l'AGV Mana.

Service	Type	Description	Paramètres
start^0	synchrone	Signal de départ de la mission	
stop^0	synchrone	Signal d'interruption de la mission	
goto^I	synchrone	Déplacement du véhicule vers le waypoint désigné	[in] ?wptTo
follow^I	synchrone	Le véhicule suit la cible localisée aux coordonnées données	[in] ?wptTarget
track^I	synchrone	Le véhicule poursuit la cible préalablement localisée	[in] ?targetName [out] ?targetState
setPosition^I	synchrone	Relocalisation du véhicule aux coordonnées données	[in] ?wptPos
setCarto^I	synchrone	Transfert d'une nouvelle carte de terrain au véhicule	[in] ?mapfile
iAmLostDetector^I	asynchrone	Le véhicule ne parvient plus à se localiser avec précision	[int] ?posEstim
iAmBlockedDetector^I	asynchrone	Le véhicule est bloqué par un obstacle imprévu	[out] ?wpt1 [out] ?wpt2
TargetDetectionOn^I	asynchrone	Le véhicule dispose d'une détection permanente d'une cible particulière	[out] ?targetName [out] ?wptTarget

7.2.4 L'AUV Daurade

Description

La Daurade (Démonstrateur d'AUv de ReA DiscrEt, REA pour Rapid Environmental Assessment) est un AUV de 5m de longueur et 0.7m de diamètre. Il pèse environ 1t. Il peut plonger jusqu'à des profondeurs de 300m, à une vitesse maximale de 8 noeuds. Son autonomie énergétique à une vitesse nominale moyenne de 4 noeuds est de 10h. Son rayon de giration est de 15m. Pour la navigation, l'AUV est équipé d'un système de navigation inertiel photonique, d'un capteur de vitesse (DVL, Doppler Velocity Log), d'un capteur de pression et d'un GPS différentiel (DGPS). Pour ses missions de recueil d'informations, l'AUV est équipé d'un sonar latéral, d'un sondeur multifaisceaux, d'un sondeur de sédiment, d'un sondeur vertical, d'une caméra, d'un sonar frontal, d'un capteur de conductivité-température-pression et d'un célérimètre. La communication peut passer par WLAN quand il est en surface, ou avec une liaison acoustique bas débit quand il est en plongée. L'AUV Daurade est opérationnel pour des missions de levé hydrographique et de levé océanographique.

FIGURE 7.16 – L'AUV Daurade de la DGA.

Architecture monovéhicule

L'architecture monovéhicule de l'AUV Daurade est adaptée de l'application embarquée LHOVAS (Levé Hydrographique et Océanographique pour Véhicule Autonome Sous-marin) (figure 7.17) qui permet de superviser en ligne la réalisation de missions individuelles et de recueillir dynamiquement des données sur l'environnement (missions de REA). LHOVAS s'appuie sur ProCoSA® (Programmation et Conduite de Systèmes à forte Autonomie [Barbier et al., 2006]), un logiciel Onera permettant le développement d'architectures décisionnelles logicielles embarquées, dont la supervision est basée sur le formalisme des réseaux de Petri.

Grâce à cette architecture décisionnelle monovéhicule, l'AUV Daurade peut :

- rejoindre une zone définie par des points de coordonnées WGS84 ;
- descendre jusqu'à une immersion donnée ;
- balayer à une immersion donnée une zone définie par des points de coordonnées WGS84 ; la planification des différents rails est réalisée dans cette couche ;

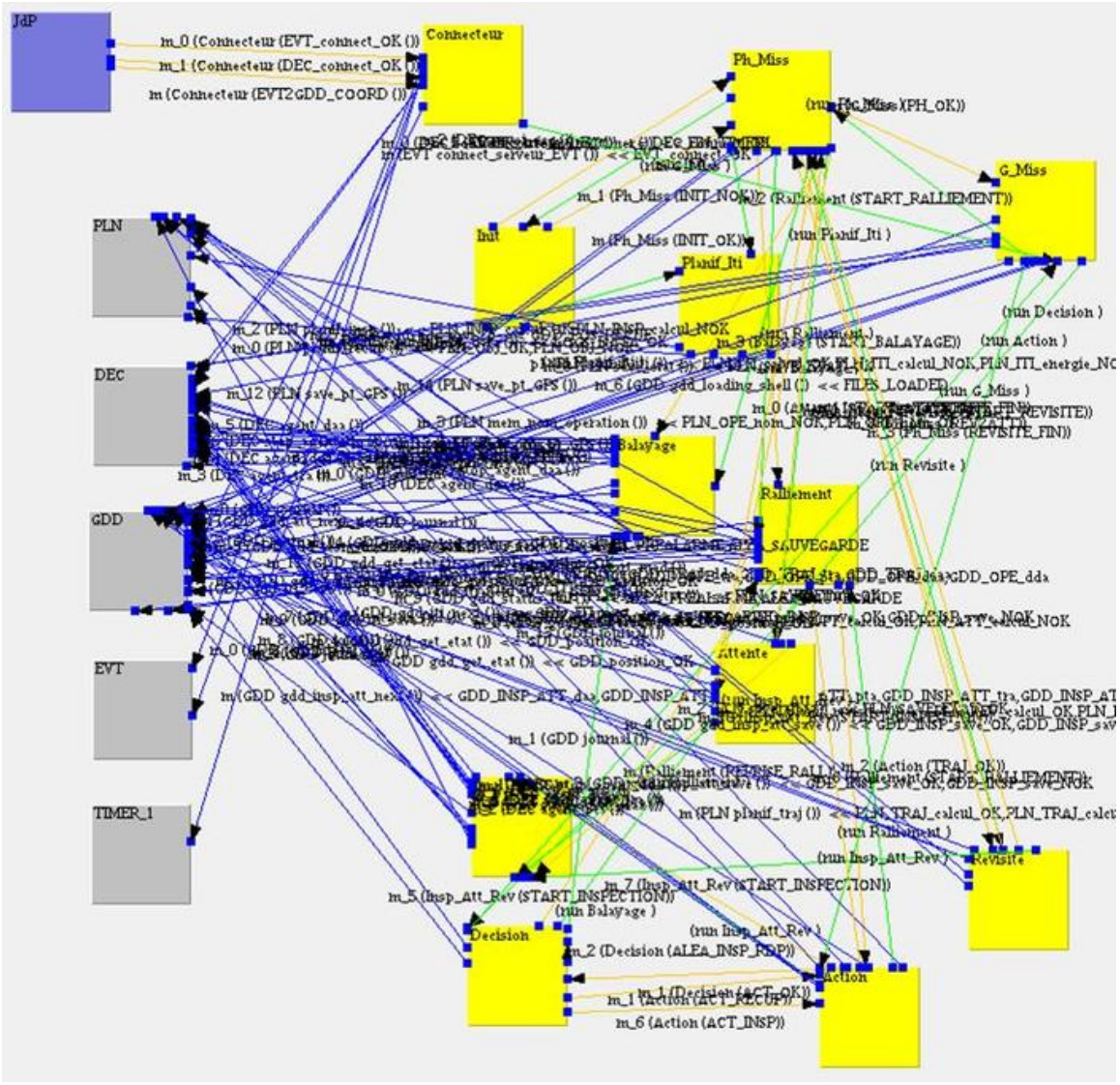


FIGURE 7.17 – Architecture monovéhicule de Daurade, avec les relations entre modules logiciels (à gauche) et réseaux de Petri (à droite).

– arrêter le balayage d'une zone donnée suite à un premier contact avec une cible et réaliser l'inspection de cette cible par un mini-rail ;
– reprendre un balayage interrompu ;
– remonter à la surface, lorsque le véhicule a localisé une mine, lorsqu'il s'estime perdu ou en fin de balayage d'une zone ;
– demander un point GPS lorsque le véhicule est en surface.

Services - Description dans Koper

Le tableau 7.4 liste les différents services de l'AUV Daurade intégrés dans Koper.

TABLEAU 7.4 – Listing des services de l'AUV Daurade.

Service	Type	Description	Paramètres
start^{0}	synchrone	Signal de départ de la mission	
stop^{0}	synchrone	Signal d'interruption de la mission	
goto^{II}	synchrone	Déplacement du véhicule vers le waypoint désigné	[in] ?wptTo
getGPSPoint^{II}	synchrone	Le véhicule renvoie sa position GPS	[out] ?waypoint1
balayage^{II}	synchrone	Le véhicule balaye une zone (incluant une plongée si nécessaire) à la recherche de cibles (les mines dans le scénario II)	[in] ?areaFile
repriseBalayage^{II}	synchrone	Reprise de balayage de la zone en cours	
remontee^{II}	synchrone	Le véhicule remonte en surface	
hippodromesCOM^{II}	synchrone	Le véhicule fait des hippodromes en attente de réception d'un message	
driftDetector^{II}	asynchrone	Le véhicule a détecté une dérive importante. Il doit se relocaliser	
updatePos^{II}	synchrone	Le véhicule met à jour sa position	[in] ?wptPos
sonarActivated^{II}	asynchrone	Le véhicule peut détecter des mines	[out] ?posMine
requestMinePos^{II}	asynchrone	Le véhicule renvoie l'état de la mine détectée et localisée	[out] ?posMine

7.3 Simulations du scénario II du PEA Action sous MORSE

Le scénario II, présenté à la section 1.4 (p. 12) puis précisé à la section 7.1.4 a été testé en simulation avec le simulateur MORSE. Le chenal est décomposé en trois zones adjacentes à explorer, de 100 mètres de large par 150 mètres de long (figure 7.18). Des points de passage sont ajoutés pour définir les points de départ et d'arrivée des véhicules, ainsi que des points de repère particuliers des zones, notamment utilisés à des fins coopératives entre les véhicules.

Dans un travail préliminaire, nous avons mis en place la structure logicielle de la figure 7.19 afin d'évaluer la fonction "exécution" de l'architecture HiDDeN, sans gestion des aléas, ni de réparations. Le but était essentiellement de valider le dialogue entre superviseurs et architectures

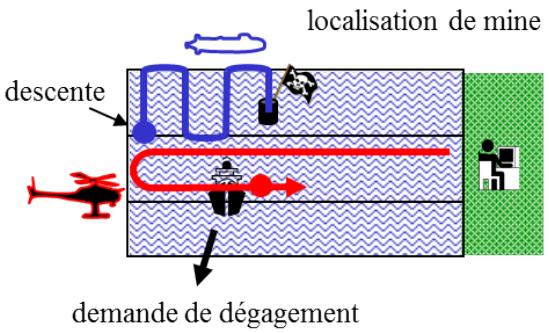


FIGURE 7.18 – Déroulement des explorations de zones du scénario II du PEA Action (ici sur les deux premières zones).

locales. L'environnement et la position des véhicules étaient visualisés à l'aide du simulateur robotique MORSE. Une architecture simulée servait à émuler l'architecture locale future du véhicule. Ce module effectuait la liaison entre le superviseur local du véhicule et les actions exécutées par le véhicule sous le simulateur. Les communications entre les modules étaient gérées par YARP [6]. Nous avons dans un premier temps simulé le balayage de deux zones par l'AAV et l'AUV comme le prévoit le scénario II, ainsi qu'un rendez-vous entre les deux balayages pour valider le fait que la couche HiDDeN puisse échanger des informations.

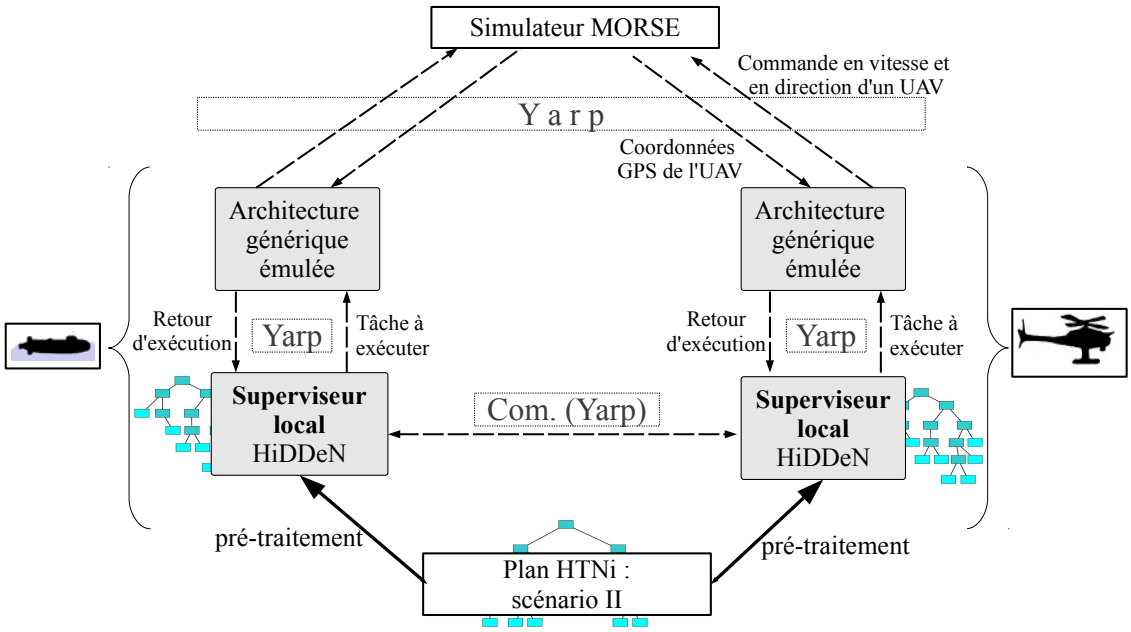


FIGURE 7.19 – Structure logicielle pour la simulation sous MORSE.

Après une période de maturation des différents logiciels, de nouvelles simulations ont été effectuées. Désormais, les aléas sont pris en compte, la planification initiale et les requêtes de plan-

6. http://eris.liralab.it/yarp/

ification gérées par AcPlan. Les architectures des véhicules sont toujours émulées, mais gèrent un interfaçage plus complexe avec le simulateur, enrichi d'entités telles que des mines et des bâtiments de surface. Les aléas de détection de bâtiments de surface, de mines, et de la dérive de l'AUV peuvent être simulés. La figure 7.20 montre les interfaces graphiques des deux superviseurs locaux HiDDeN, qui permettent de faciliter le suivi de la mission par un opérateur, et les figures 7.21 et 7.22 montrent quelques images extraites de ces simulations.

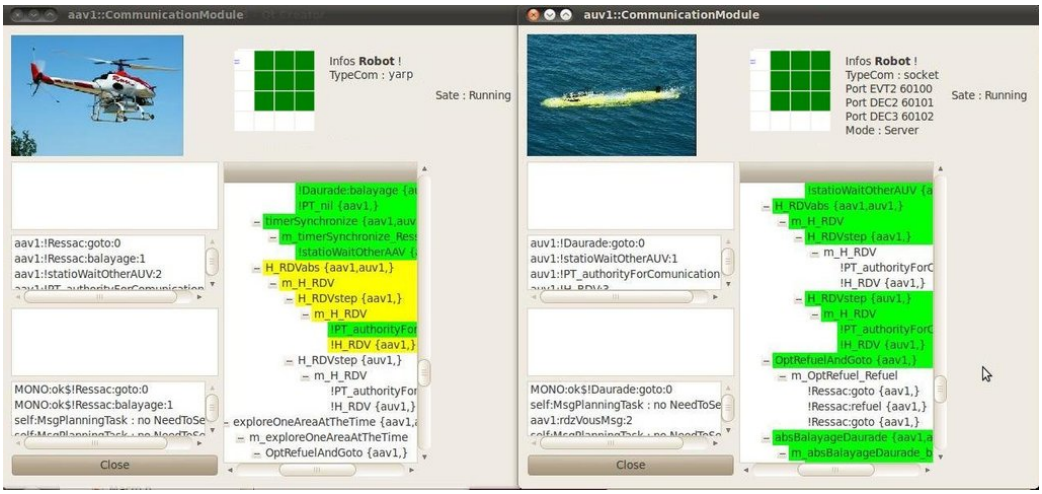


FIGURE 7.20 – Interfaces graphiques des superviseurs locaux HiDDeN en train d'exécuter la mission "Scénario II".

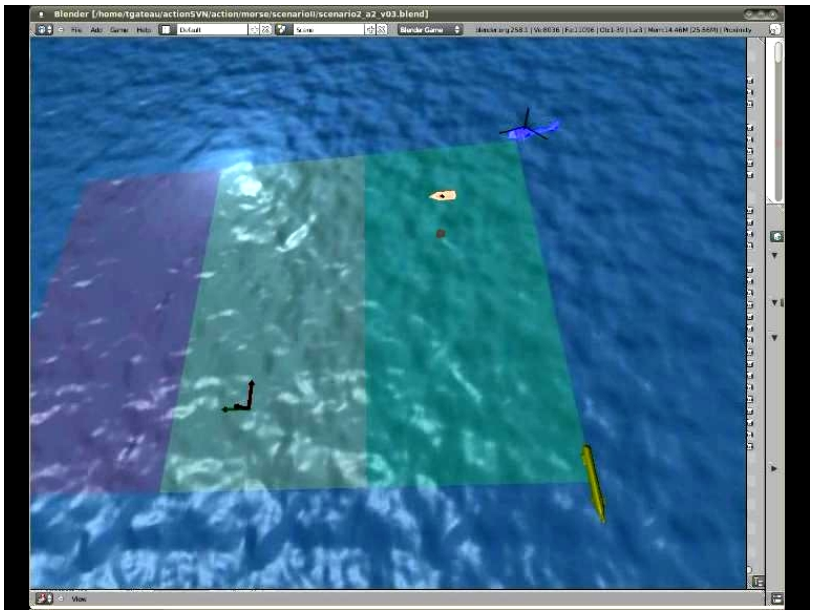


FIGURE 7.21 – Vue d'une simulation en cours d'exécution pour le scénario II du PEA Action, sous MORSE.

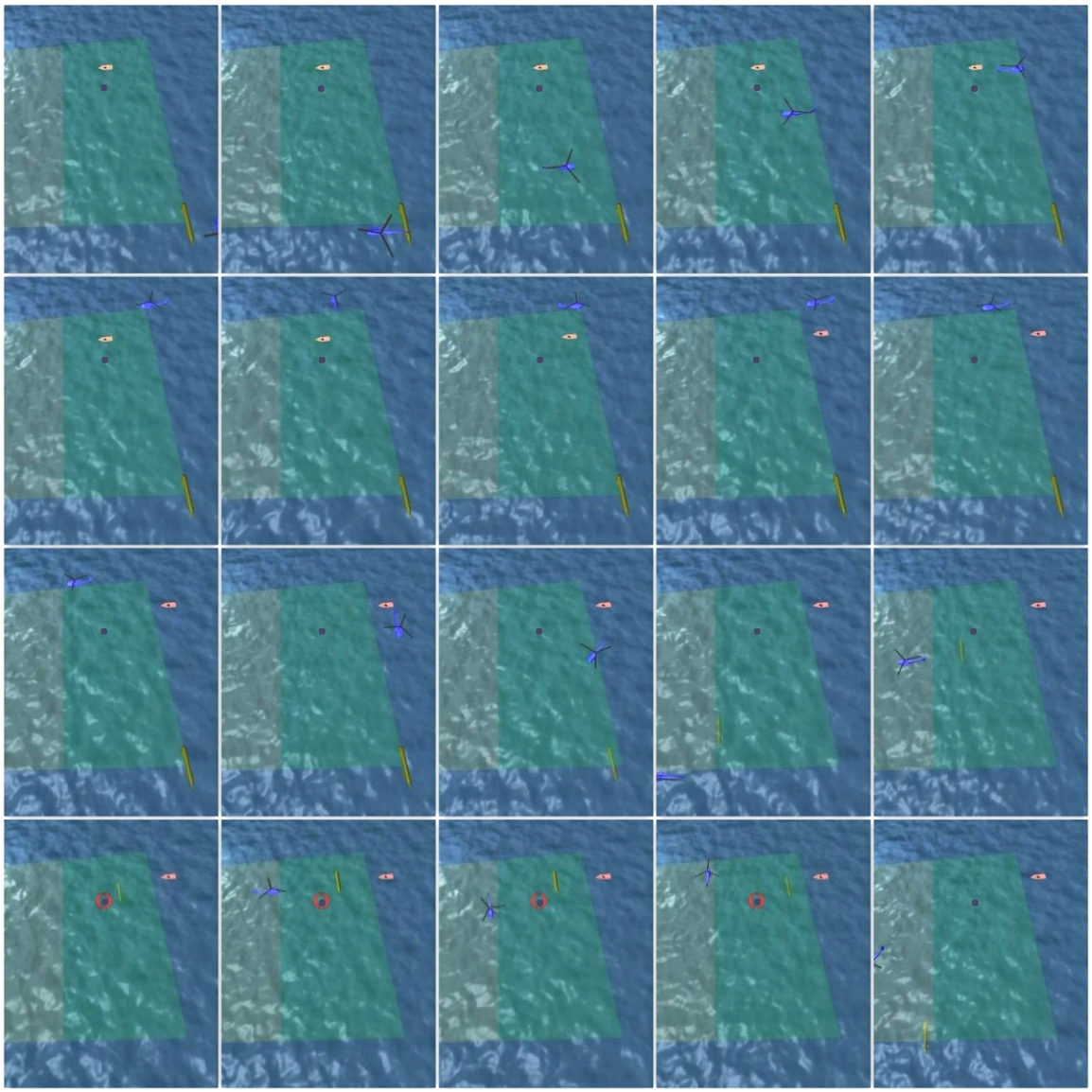


FIGURE 7.22 – Séquence illustrant le déroulement du scénario II. Nous repérons les images par leur ordre, de gauche à droite, et de haut en bas. L'AAV ReSSAC explore la première zone (images 1 à 6) jusqu'à détecter la présence d'un bâtiment de surface (image 7). Il lui fait signe et vérifie qu'il a quitté la zone (images 8 et 9), puis reprend son balayage (image 10). Il finit son balayage et donne ensuite le feu vert à l'AUV Daurade qui peut à son tour explorer la première zone (images 11 à 15). Pendant l'exploration sous-marine, l'AAV ReSSAC entame l'exploration de la surface de la seconde zone (image 15). AUV Daurade détecte une mine (image 16). Il remonte donc en surface. Il doit patienter (image 17-18) pour que l'AAV ReSSAC se mette à portée de communication pour la lui signaler (l'AAV ReSSAC effectue régulièrement des rails vers la zone où l'AUV Daurade est en plongée). Enfin, la mine étant signalée, l'AUV peut replonger et la mission peut continuer (image 20).

7.4 Simulations et expérimentation du scénario I du PEA Action

7.4.1 Le scénario de démonstration

La démonstration officielle du scénario I consiste pour les deux véhicules, l'AAV ReSSAC et l'AGV Mana, à à explorer la zone de mission en suivant les chemins calculés par le planificateur puis à suivre une cible qui aura été détectée et localisée sur la zone. La zone de mission choisie pour les simulations et expérimentations est une partie du village de combat du camp militaire de Caylus (Tarn-et-Garonne). La planification AcPlan a aussi prévu deux rendez-vous, le premier à mi-chemin, le second en fin de parcours. Les premiers plans calculés par AcPlan n'étaient pas réalisables expérimentalement, car les données terrain disponibles étaient insuffisantes (notamment en fonction des zones non desherbées dans lesquelles le véhicule terrestre ne pouvait pas toujours circuler). Nous avons donc défini sept points de passage pour chaque véhicule et le planificateur a calculé le chemin pour chacun des deux véhicules (figure 7.23). Des aléas peuvent être introduits par les opérateurs, ou tout simplement survenir réellement : l'AGV peut se perdre "AGV_PERDU" (figure 7.25) et doit alors être relocalisé, ou peut rester bloqué "AGV_BLOQUE" auquel cas il doit attendre que l'AAV lui fournisse une cartographie de la zone où il se trouve. Les deux véhicules peuvent également détecter un intrus "CIBLE_LOCALISEE", qu'ils doivent alors poursuivre, action prioritaire à toutes les autres. Des ruptures de communications sont à prendre en compte lorsque les deux véhicules sont trop éloignés l'un de l'autre (figure 7.24). Le scénario se termine lorsque le second point de rendez-vous est atteint, ou qu'un intrus a été chassé de la zone opérationnelle.

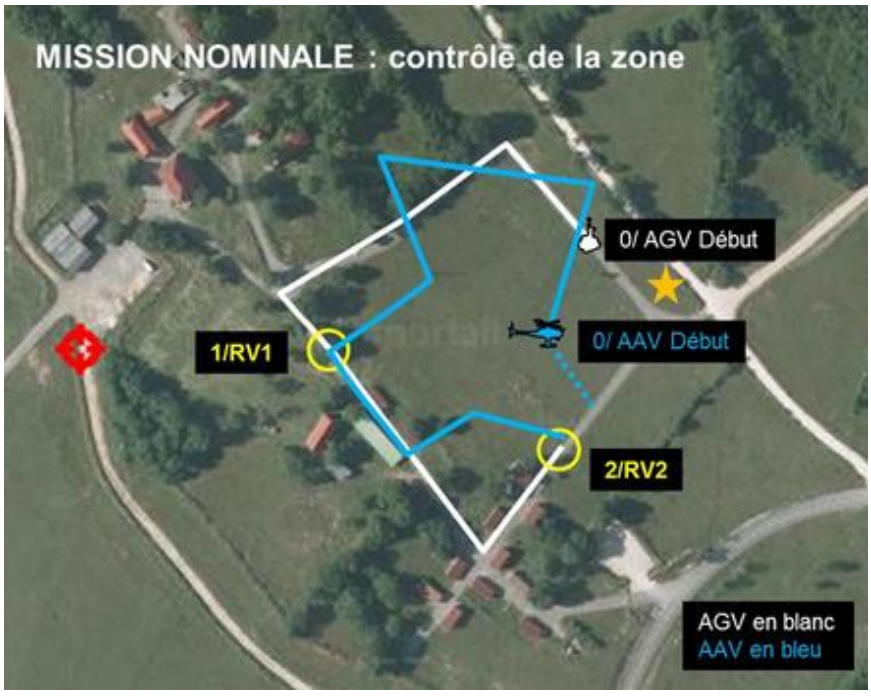


FIGURE 7.23 – Schéma de la première démonstration de la première mission.

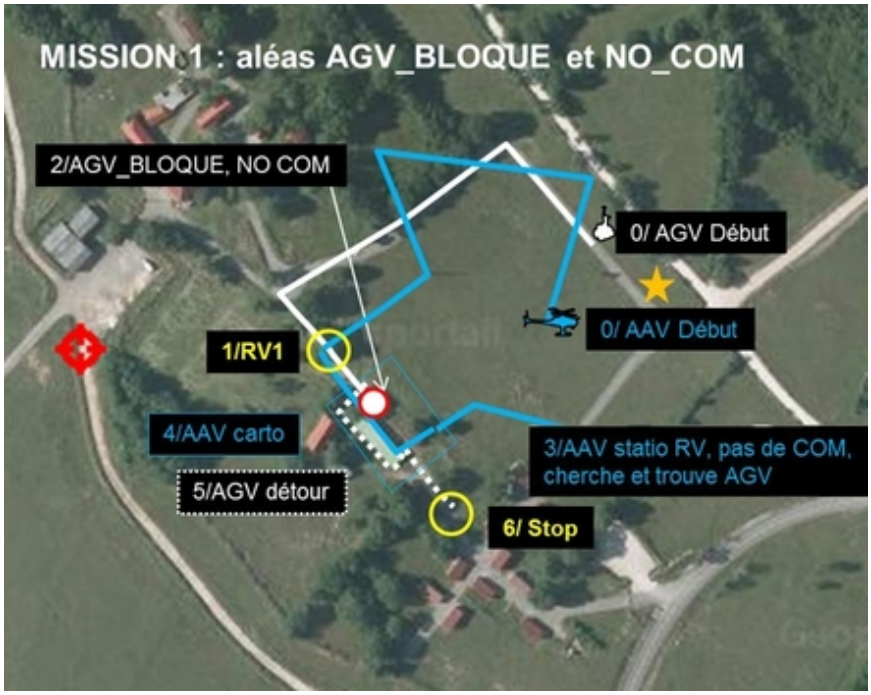


FIGURE 7.24 – Schéma de la seconde mission : aléa AGV_BLOQUE, et les communications ne sont pas disponibles au déclenchement de l'aléa.

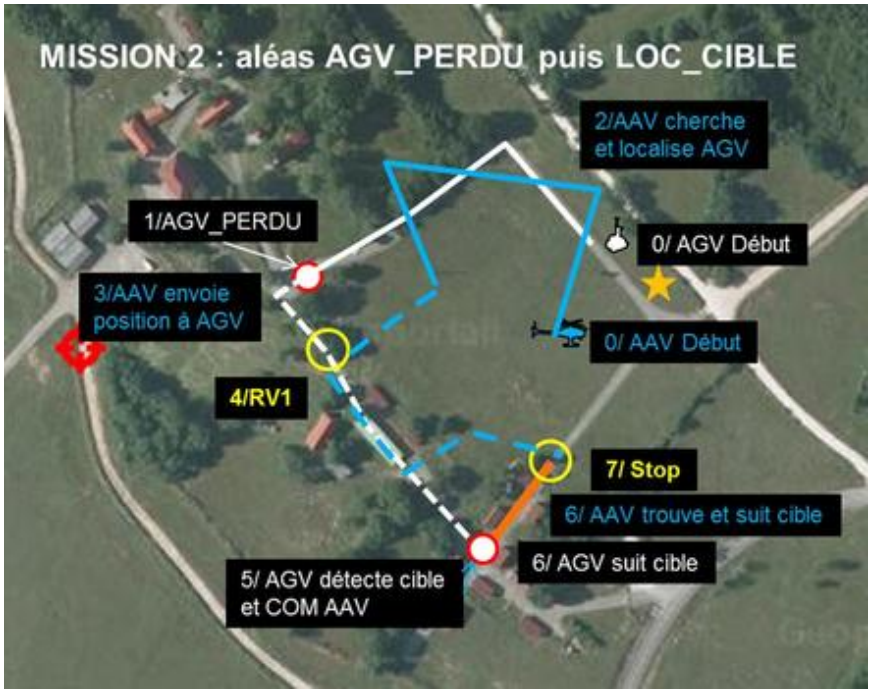


FIGURE 7.25 – Schéma de la troisième mission : aléa AGV_PERDU, puis CIBLE_LOCALISEE.

7.4.2 Simulations sous MORSE

Des simulations sous MORSE du scénario I du PEA Action (figure 7.26) ont permis de valider l'ensemble des logiciels avant leur déploiement pour les expérimentations. Les architectures locales des véhicules étaient au début des architectures émulées, comme dans les simulations du scénario II, puis les véritables architectures pour les expérimentations. HiDDeN s'interfaçait à elle, pour gérer la supervision du scénario.

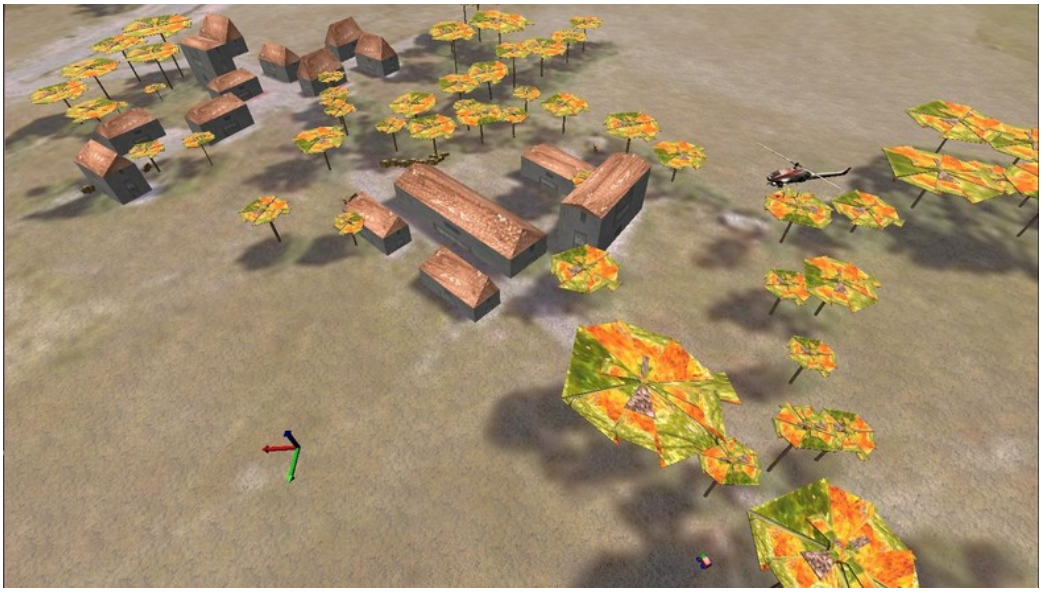
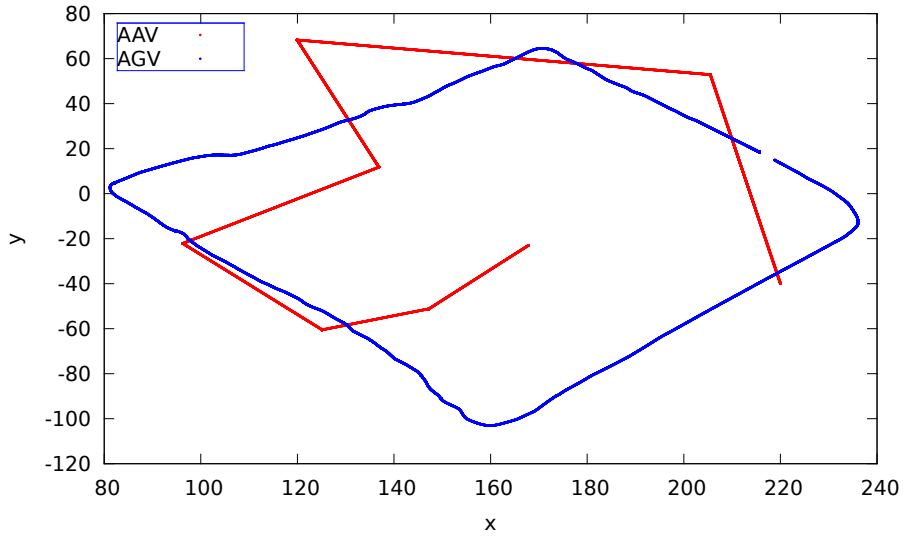


FIGURE 7.26 – Vue d'une simulation en cours d'exécution pour le scénario I du PEA Action.

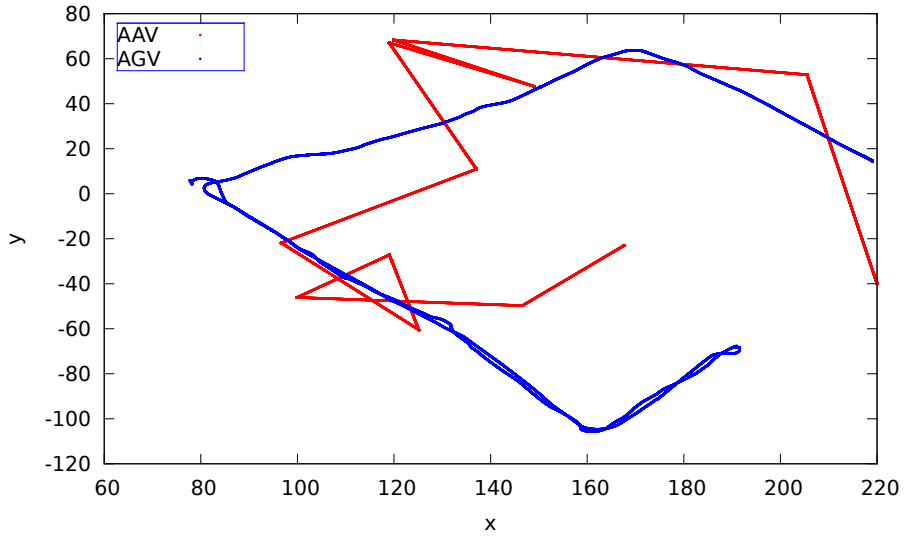
7.4.3 Simulations hybrides : l'AGV réel, et l'AAV simulé

Des simulation hybrides ont pu être menées sur le terrain militaire de Caylus. Elles impliquaient l'AGV Mana réel, et l'AAV ReSSAC simulé sous MORSE. La figure 7.27 montre les traces de deux exécutions de mission s'étant correctement déroulées. Dans le scénario nominal (figure 7.27(a)), les deux véhicules parcourent dans l'ordre leurs waypoints respectifs, et effectuent leurs rendez-vous au moment défini. L'AAV part du point (220,-40) pour finir au point (170,-20). L'AGV part du point (210,20) pour décrire une boucle quasi complète jusqu'au waypoint (220,10), en suivant la route.

Des tests ont été réalisés pour vérifier le comportement de l'équipe en présence d'aléas. la seconde trace (figure 7.27(b)) illustre une telle exécution : par rapport au scénario nominal, on constate que l'AAV a fait un détour lorsqu'il a atteint le point (120,60). Il est allé relocaliser l'AGV qui s'était perdu. De même, vers les coordonnées (120,-40), l'AAV a effectué la cartographie entre deux points (120,-25) et (100,-45) fournis par l'AGV qui était bloqué par un obstacle. On constate que la trace de l'AGV revient sur ses pas au lieu de terminer la mission : un faux intrus a été détecté au point (190,-80), ce qui a lancé l'action de poursuite. L'opérateur de l'AGV Mana a alors décidé d'arrêter la mission, et l'a envoyé rejoindre le point (80,10).



(a) Scénario I nominal.



(b) Scénario I, 2 aléas.

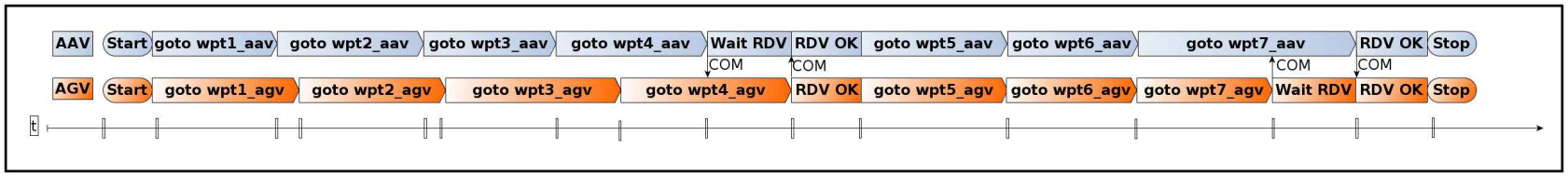
FIGURE 7.27 – Traces pour 2 scénarios hybrides dans un repère orienté Est Nord.

7.4.4 Expérimentations

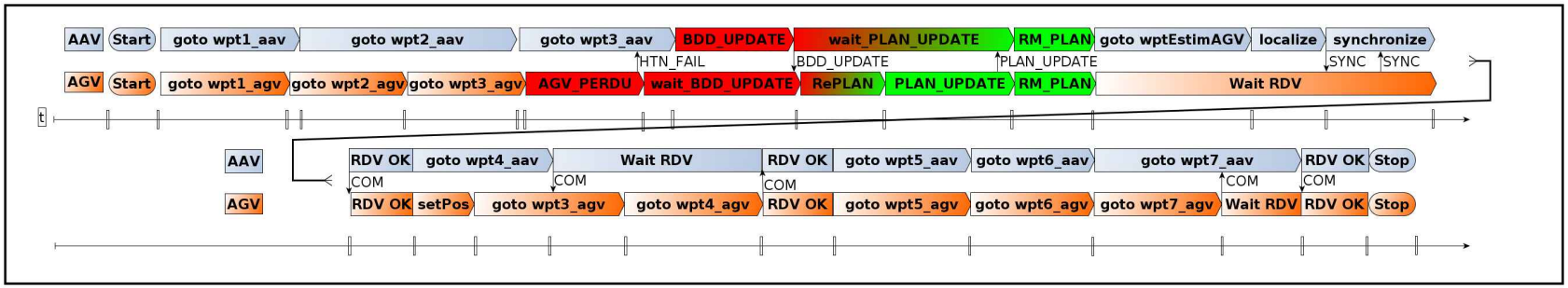
Lors des premières expérimentations, des plans sous forme de "scripts" ont été largement employés, afin de diminuer la complexité et les sources d'erreurs pour une régénération de plan en ligne, durant une expérimentation réelle. Bien que peu satisfaisante sur le plan théorique (les véhicules autonomes se contentent de suivre la liste d'actions qu'un opérateur a décrit hors ligne, et sélectionne sans réel processus de décision quel script exécuter en cas d'aléas), l'aspect expérimental a été riche en enseignements. L'émulation de la composante planification nous a permis de valider notre modèle d'architecture pour la supervision, ainsi que de disposer de premiers retours en situation réelle pour les premiers scénarios du PEA Action. Cela contribue également à montrer

l'aspect modulaire d'une telle architecture, qui peut se doter de la composante de planification la plus adaptée à ses besoins, suivant le contexte de la mission.

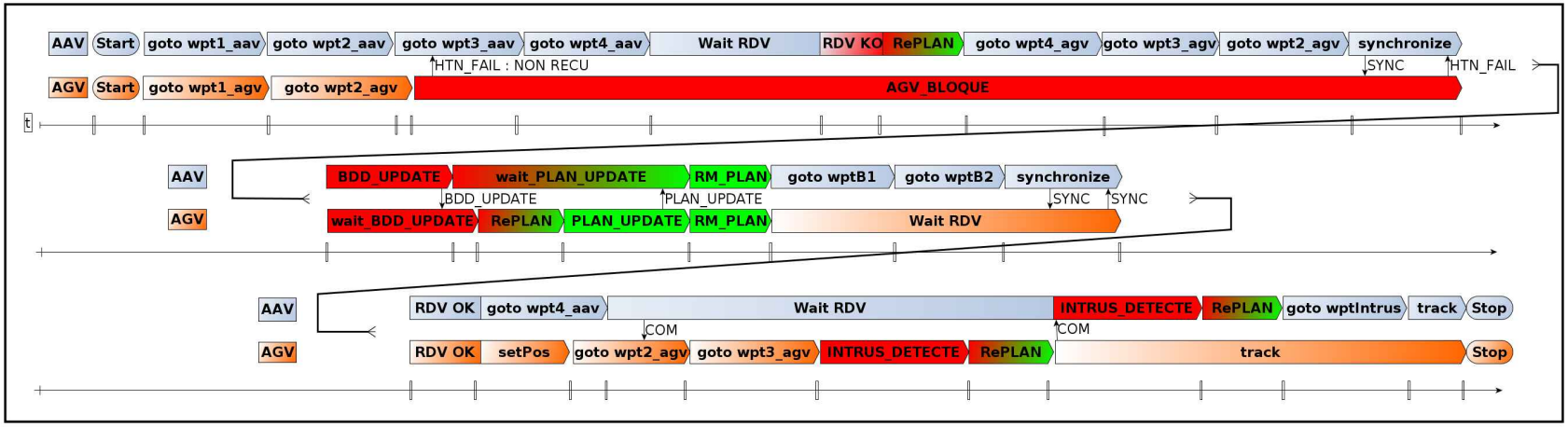
Ensuite, la campagne d'expérimentations en vue de la démonstration du scénario I du PEA Action a été lancée. Nombre d'expérimentations sur le terrain ont été menées. Elles se sont révélées d'une grande richesse en terme d'expérience acquise. La réalité du terrain est effectivement très différente des simulations même si ces dernières sont tout de même un atout considérable pour augmenter les chances de réussite d'un essai réel. Les diagrammes de la figure 7.28 illustrent le déroulement de différentes expérimentations.



(a) Exemple de déroulement nominal.



(b) Aléa AGV_PERDU. L'AAV vient relocaliser l'AGV. Les deux véhicules peuvent alors reprendre leur mission.



(c) Aléa AGV_BLOQUE, hors de portée de communication lors de l'occurence de l'aléa. L'AAV détecte l'aléa lors du premier rendez-vous non respecté.

FIGURE 7.28 – Time lines de trois déroulements différents du scénario I du PEA Action.

7.4.5 Extrait du journal des essais

Cette section est extraite du journal des expérimentations menées à Caylus. Le tableau 7.4.5 récapitule l'ensemble des essais dans le cadre du scénario I du PEA Action, en se focalisant sur HiDDeN. Les essais étaient menés avec un ou deux véhicules. Ceux-ci étaient réels ou émulés. Des expérimentation hybride (AGV réel, AAV simulé dans MORSE) ont également été réalisées.

TABLEAU 7.5 – Historique des essais

Date	Type et révision HiDDeN	Description des essais
20/07/2011 - essai 1	AAV réel -r3515	Mission : suivre une route constituée de 3 waypoints pour tester l'interface entre HiDDeN et Orocos. Déroulement : premier waypoint atteint, puis vol arrêté pour cause de pluie.
20/07/2011 - essai 2	AAV réel -r3515	Mission identique au vol 1. Déroulement : nominal. Interface validée entre HiDDeN et Orocos en embarqué.
20/07/2012 - essai 1	AAV émulé AGV réel -r6067	Problème de port du superviseur local de l'AGV qui n'a pas reçu l'ordre de départ. Mission arrêtée.
20/07/2012 - essai 2	AAV réel AGV réel -r6067	Succès du premier rendez-vous mais AGV se retrouve dans les herbes, qui provoque un blocage sans déclencher AGV_BLOQUE ("ko" envoyé au superviseur, donc réparation en boucle du goto, qui n'aboutit à rien). Mission arrêtée.
20/07/2012 - essai 3	AAV émulé AGV -r6067	Test de aléa CIBLE_DETECTE. Un waypoint envoyé, cible détectée et correctement suivie. Cible perdue, échec de replanification (cas hors scénario - envoi de l'action suivi (track) en boucle).
17/09/2012 - essai 1	AAV réel AGV réel -r6067 (backup)	Mission : scénario I nominal. Problème logiciel dans l'architecture monovéhicule de l'AAV, mission arrêtée.
17/09/2012- essai 2	AAV réel AGV réel -r6067 (backup)	Mission : scénario I nominal. 4 premiers waypoints validés, puis vol arrêté pour cause d'un problème de sécurité de l'AAV.
10/10/2012 - essai 1	AAV réel AGV réel -r6904	Test scénario I nominal. Mission arrêtée, échec du premier rendez-vous. Plusieurs serveurs YARP fonctionnaient pour une même connexion.
10/10/2012 - essai 2	AAV réel AGV réel -r6904	Test scénario I nominal. Côté AAV-ReSSAC, arrivé au second point de rendez-vous, le pilote de sécurité est gêné par l'AAV en stationnaire devant le soleil → déplacement du dernier waypoint pour la prochaine mission. Aléa côté AGV : une cible a été détectée[7], le panneau "Stop" dans le village → masquage du panneau pour limiter les fausses détections.

7. La détection est hors du périmètre du PEA Action, elle est simplifiée avec l'utilisation de cibles munies de plaques rétro-réfléchissantes.

10/10/2012 - essai 3	AAV réel AGV réel -r6904	Test scénario I nominal. Côté AAV-ReSSAC, arrivée au second point de rendez-vous, sur le waypoint wpt7-aav1, ne correspondant pas à celui modifié. Cause problème : erreur humaine au niveau de la modification du point wpt7-aav1 dans la base Koper embarquée sur l'AGV. La synchronisation des deux bases a considéré que celle de l'AGV était la plus récente concernant ce waypoint, et l'a donc mise à jour. Sinon, la mission peut être considérée comme un succès, les deux véhicules ayant atteint leur objectif, et validé leur dernier rendez-vous → correction des coordonnées du waypoint fautif.
10/10/2012 - essai 4	AAV réel AGV réel -r6904	Scénario I, test pour valider AGV_PERDU avant le premier rendez-vous. Réparation réussie avec succès au niveau de HiDDeN, carte correctement transmise. Arrêt volontaire de la mission.
10/10/2012 - essai 5	AAV réel AGV réel -r6904	Scénario I, test pour valider AGV_PERDU avant le premier rendez-vous. Réparation réussie avec succès au niveau de HiDDeN, carte correctement transmise. Trajet AAV : wptInitAav → wpt1 → position AGV_PERDU → carto → wpt1 → wpt2 → Fin de mission. Trajet AGV wptInitAgv → wpt1 → perdu → reloc → wpt2 → Fin de mission. Arrêt volontaire de la mission. Remarque : vol stationnaire très long de l'AAV lors de l'attente de l'AGV.
10/10/2012 - essai 6	AAV réel AGV réel -r6904	Scénario I, test AGV_PERDU avant le premier rendez-vous. Départ retardé de l'AAV pour éviter un vol stationnaire trop long en attendant AGV. Lors de l'occurrence de l'aléa AGV_PERDU, le superviseur local de l'AAV entre en phase de réparation, mais semble continuer sur son plan. Il ne va relocaliser l'AGV qu'une fois ses deux premiers waypoints accomplis. Lors de cette relocalisation, l'équipe décide d'annuler la mission, car le comportement envoyé par le superviseur local de l'AAV semble étrange. En fait, le décalage de l'initialisation de la mission des deux véhicules a entrainé un décalage des plans de missions des véhicules. Le planificateur de l'AAV considère prioritaire d'effectuer les deux premiers waypoints avant de relocaliser l'AGV. La mission aurait donc pu continuer.
11/10/2012 - essai 1	AAV réel AGV réel -r6904	Test AGV_PERDU entre le waypoint 3 et le waypoint 4. Mission arrêtée, erreur dans l'architecture monovéhicule de l'AAV (flux de données non transmis)
11/10/2012 - essai 2	AAV réel AGV réel -r6904	Test AGV_PERDU entre le waypoint 3 et le waypoint 4. Comportement anormal de l'architecture monovéhicule de l'AGV qui doit être réinitialisé. Le superviseur local embarqué sur l'AGV réinitialisé pour pouvoir renvoyer wpt1_agv vers l'architecture monovéhicule de l'AGV. Mission arrêtée, erreur dans HiDDeN, réception de l'acquittement "ok" du waypoint 1 ignoré (à cause de la réception de la commande START en double)

17/10/2012 - essai 1	Hybride -r6904	Test scénario I nominal. Exploration en parallèle de la zone de mission. L'AGV détecte et localise une cible non prévue : les plaques d'un véhicule garé. L'AGV se met en mode suivi de cible. La mission s'arrête.
17/10/2012 - essai 2	Hybride -r6904	Test scénario I nominal. Mauvaise initialisation de composants de l'architecture monovéhicule de l'AGV. La mission a été arrêtée.
17/10/2012 - essai 3	Hybride -r6904	Test scénario I nominal. Exploration, premier rendez-vous entre les deux véhicules, qui poursuivent l'exploration. L'AGV détecte et localise une deuxième cible non prévue : le panneau routier. Annulation de la commande de suivi de cible, les véhicules reprennent l'exploration. Traces visibles sur la figure 7.27(a).
17/10/2012 - essai 4	Hybride -r6904	Avant le premier rendez-vous, deux aléas sont générés par l'opérateur : AGV_PERDU et AGV_BLOQUE. Dans le premier cas, l'AAV est allé renvoyer sa position à l'AGV virtuellement perdu, qui a continué la mission. Dans le deuxième cas, l'AGV a fourni à l'AAV deux waypoints, ce dernier a réalisé un survol du rail ainsi créé, a transmis la carte à l'AGV qui a continué la mission sur le même chemin (il n'était pas réellement bloqué). La mission s'est terminée sur le deuxième point de rendez-vous, comme prévu (pas de cible positionnée sur la zone). Traces visibles sur la figure 7.27(b).

Les essais hybrides et réels montrent la supervision correcte de HiDDeN, qui gère à la fois le comportement nominal et la réaction aux aléas, avec des réparations locales de plan :

- l'exploration coopérative de la zone de mission suivant le plan calculé, avec la gestion des rendez-vous et de l'échange de données entre les deux véhicules lors de ces rendez-vous ;
- la détection/localisation et le suivi d'une cible par le véhicule aérien (effectué en dehors des expérimentations résumées ci-dessus) ;
- la détection/localisation et le suivi d'une cible par le véhicule terrestre ;
- la prise en compte de l'aléa AGV_PERDU avec la coopération des deux véhicules pour aider à sa relocalisation ;
- la prise en compte de l'aléa AGV_BLOQUE avec la coopération des deux véhicules pour aider à son déblocage.

7.5 Conclusion

Le benchmark "Logistics" (section 7.1.3) a permis de renforcer nos arguments en faveur de l'utilisation d'une approche distribuée par rapport à une approche centralisée, pour une mission composée de cinq véhicules autonomes. Le second benchmark (section 7.1.4) a permis de vérifier que la stratégie de réparation locale décrite dans le chapitre 6 était implémentable et que son comportement était celui attendu. Ce benchmark a par ailleurs souligné l'importance de disposer d'une structure hiérarchique pour le HTNi, et non simplement d'une simple séquence de tâches. Le travail de modélisation des tâches abstraites HTN du domaine de la mission doit donc être soigné. HiDDeN a été testé grâce au simulateur MORSE à travers les scénarios I et II du PEA action. Le premier scénario, en plus d'avoir été testé en simulation, a donné lieu à une campagne d'expérimentation

sur le terrain, qui peut être considérée comme un succès de par les retours d'expérience enrichissant qu'elle a fournis, et par le fait que les diverses configurations prévues pour la démonstration officielle du scénario I ont pu être exécutées de façon optimale.

CHAPITRE 8

Conclusions et perspectives

8.1 Conclusions

8.1.1 Synthèse

L'objectif de cette thèse a consisté à concevoir et à mettre en œuvre une architecture décisionnelle qui permette de superviser l'exécution d'une mission par une équipe de véhicules autonomes hétérogènes. A travers ce mémoire, nous souhaitons apporter des réponses à plusieurs problèmes incontournables pour la conception d'un tel système. Tout d'abord, l'hétérogénéité des véhicules dont on veut exploiter les architectures monovéhicules individuelles qui sont déjà fonctionnelles, constitue le premier obstacle. La supervision d'équipe doit aussi tenir compte de l'aspect dynamique de l'environnement dans lequel l'équipe évolue, source d'aléas qui apparaîtront immanquablement dans les missions à caractère opérationnel traitées pas l'équipe. Elle doit également considérer en particulier que les communications ne sont pas toujours disponibles entre véhicules, qui ne peuvent donc pas échanger à chaque instant des informations. L'équipe peut disposer d'un plan de mission, construit hors ligne, dont la forme peut être exploitée pour permettre une supervision d'équipe efficace.

A travers l'état de l'art (chapitre 2, p.17) nous avons pu constater que l'hétérogénéité des véhicules ne semble pas être une préoccupation majeure des travaux décrits. Les véhicules ou agents considérés par les concepteurs d'architectures multivéhicules leur fournissent plutôt des interfaces standardisées pour faciliter les échanges entre la couche décisionnelle d'équipe et la partie commande du véhicule autonome. Il ne semble pas y avoir de solution idéale concernant la gestion des communications, mais plusieurs auteurs parviennent toutefois à réduire les échanges nécessaires entre les véhicules. Un décalage existe entre l'univers généralement symbolique de la planification et le monde réel dans lequel les véhicules autonomes sont plongés.

Grâce à la base de connaissances Koper (chapitre 4, p.53), nous fournissons un outil pour décrire formellement les véhicules autonomes membres de l'équipe, et en particulier leur interface, point crucial pour automatiser des transferts d'information entre une entité décisionnelle d'équipe et l'architecture monovéhicule que nous souhaitons conserver. En plus de la description des interfaces, Koper fournit une base de données exploitable pour l'exécution concrète d'actions décrites dans un plan de mission issu de phases décisionnelles symboliques. Cela contribue à combler le fossé qui

existe entre la planification abstraite et l'exécution d'actions réelles par des vehicules autonomes. En effet, à tout moment la base de connaissances Koper permet de générer des problèmes de planification qui peuvent être interprétés et résolus par un planificateur. Le plan retourné par ce dernier peut à son tour être interprété pour être exécuté par les véhicules autonomes, via la supervision d'équipe qui s'appuie sur Koper.

Pour la gestion de l'exécution de la mission d'équipe, nous proposons une architecture décisionnelle, HiDDeN (chapitre 3, p.45). Celle-ci est distribuée sous forme de superviseurs locaux à bord des véhicules, et permet de doter l'équipe de l'autonomie nécessaire à la coopération de ses membres. Un superviseur local se décompose donc en plusieurs modules logiciels. Un premier module (*ExecMessenger*) assure le dialogue avec l'architecture monovéhicule, en se basant sur les formalisations définies par Koper qui lui autorise une certaine généricité. Un deuxième module (*PlannerManager*) se charge de la même façon des échanges avec les planificateurs embarqués du véhicule. Un troisième (*CoopMessenger*) est consacré à la gestion des échanges entre les superviseurs locaux. Un quatrième module (*DataManager*) abrite une instance de la base de données Koper. Enfin, un module central (*MissionManager*) permet d'organiser les flux de données entre les modules précédents, et gére à proprement parler l'exécution du plan de mission.

La gestion de cette exécution fait également parti de nos contributions (chapitre 5, p.67). Nous proposons de nous appuyer pour la supervision sur un plan issu d'une planification HTN et sous un format HTNi, qui permet de conserver les relations entre tâches ainsi que la structure hiérarchique des HTN. Nous avons montré comment le plan HTNi global d'équipe était distribué à bord de chaque superviseur local en tenant compte des tâches de synchronisation à ajouter et qui sont nécessaires à la coopération des véhicules pour l'accomplissement de leur mission. Nous proposons également des moyens pour détecter les aléas qui sont du ressort de la couche décisionnelle multivéhicule, notamment par une interaction entre la gestion de l'exécution et la base de connaissances Koper.

En effet, dans le contexte opérationnel considéré, des aléas surviendront pendant le déroulement de la mission. Même si nous ne pouvons donner des réponses pour l'ensemble des événements inattendus qui peuvent apparaître durant l'exécution de la mission, nous fournissons des moyens pour doter l'équipe de plans alternatifs lorsque l'aléa relève de la couche décisionnelle multivéhicule (chapitre 6, p.89). Mieux, nous dotons la supervision d'une stratégie de réparation menée aussi localement que possible et qui s'appuie sur la structure hiérarchique de HTNi. Cela procure le double avantage de limiter à la fois le coût de replanification et le nombre de véhicules impliqués dans le processus, et donc le nombre de communications. Les communications peuvent être indisponibles à tout moment, que ce soit durant l'exécution du plan de mission pour des causes connues ou non, ou lors des processus de réparation impliquant plusieurs véhicules. Ces derniers doivent en effet synchroniser leurs données pour permettre la génération d'une requête de planification la plus représentative possible de l'état courant de l'environnement dans lequel ils sont plongés, et recevoir en retour un plan alternatif qui remplacera la branche défaillante de leur plan de mission. Des schémas HTN, exploitables lors de ces phases de réparation, sont utilisés pour remédier à ce genre de contraintes.

Enfin, le système HiDDeN a été implémenté et testé. Deux benchmarks ont permis de vérifier que son comportement était conforme à sa spécification. Des simulations sous MORSE ont également validé son fonctionnement, pour aboutir aux expérimentations sur le terrain, qui vont donner lieu à des démonstrations officielles : HiDDeN a été déployé en mission réelle, comme superviseur, dans les expérimentations du scénario I pour le PEA Action (chapitre 7, p.105) démontrant par là l'atteinte concrète des objectifs fixés. Ces simulations et expérimentations ont permis de mettre

plusieurs points en avant. L'aspect distribué de la supervision s'avère utile sur deux aspects. A aucun moment un véhicule ne se retrouve sans organe décisionnel de haut niveau, même s'il est isolé en terme de communication. De plus, le passage à l'échelle vers des équipes composées de plus de deux véhicules ne devrait pas être contraint par la charge de calcul nécessaire pour la supervision, celle-ci étant répartie à bord des différents véhicules. Il n'en est en revanche pas de même pour les phases de réparation de plan, qui pourraient impliquer plus de véhicules et des problèmes de planification plus complexes à résoudre. Cependant, la stratégie de réparation locale mise en place permet dans une certaine mesure de limiter ces inconvénients. L'expérience de la réalité du terrain reste irremplaçable. Elle introduit des facteurs difficiles à simuler, tel que le stress de l'équipe en charge des démonstrations, les conditions climatiques, ou la quantité et les formes étonnantes que peuvent prendre les aléas de mission réels (conflit entre les opérateurs, soleil éblouissant, panneau routier pris pour cible, réparation de plan différente des attentes de l'opérateur sécurité...)

8.1.2 Contributions

Koper

Avec Koper, nous proposons une base de connaissances pour la planification, l'exécution et la replanification de mission pour une équipe de véhicules hétérogènes. Elle permet de formaliser les services que fournissent des véhicules autonomes hétérogènes, qui n'auraient pas été nécessairement conçus pour effectuer des missions en coopérant avec d'autres véhicules pour former une équipe. Développé à partir d'une ontologie, Koper permet de regrouper à la fois des informations qui concernent les aspects décisionnels, abstraits (comme la planification), et des informations concrètes, plus spécifiques à la phase d'exécution. Koper facilite aussi l'intégration d'un nouveau type de véhicule autonome dans l'équipe, voire d'un véhicule téléopéré ou piloté.

HiDDeN

Centrale dans ce mémoire, l'architecture que nous proposons permet d'exécuter des plans de mission hiérarchiques, les HTNi, et surtout de répondre à des aléas de mission par une stratégie de réparation effectuée aussi localement que possible, principalement pour éviter de communiquer avec des véhicules autonomes qui ne seraient pas impliqués par l'aléa, tout en fournissant des plans alternatifs dont l'exécution permettra d'atteindre les objectifs de mission. Elle permet également de gérer la synchronisation de véhicules qui doivent effectuer une réparation d'équipe lorsque c'est nécessaire.

PEA Action

Une implémentation de l'architecture HiDDeN, appuyée sur une base de données Koper, et pouvant effectuer des requêtes de planification vers le planificateur AcPlan a prouvé lors de simulations puis d'expérimentations sur le terrain qu'elle disposait de bases solides pour assurer le rôle qui lui était destiné, la supervision de mission pour les scénarios du PEA Action.

8.2 Discussions et perspectives

8.2.1 Passage à l'échelle

La nature distribuée de l'architecture devrait tendre à faciliter le passage à l'échelle pour superviser des équipes de plus grande taille. Des simulations sur un benchmark avec cinq véhicules mettent en avant le comportement attendu des architectures distribuées. Mais des simulations avec un plus grand nombre de véhicules seraient nécessaires pour mieux juger de cet aspect. Le mécanisme de réparation locale passe bien à l'échelle en ce qui concerne le nombre d'aléas rencontrés, comme nous avons pu le constater à la figure 7.6 du chapitre 7 (p.117). Il permet également de limiter l'impact d'un aléa aux seuls coéquipiers directement concernés, et réduire ainsi le nombre de véhicules à synchroniser pour une réparation. En revanche, la synchronisation de l'ensemble d'équipes plus larges fait partie des défis à relever, lorsque les aléas ne peuvent pas être résolus à un niveau suffisamment local.

Sur le terrain, et avec un superviseur local embarqué par véhicule, la charge de calcul est divisée entre les véhicules autonomes, ne freinant pas la croissance théorique du nombre de membres de l'équipe. Par contre, il serait intéressant de s'interroger sur l'augmentation des charges de communication que des coopérations de larges équipes de véhicules impliqueraient.

8.2.2 Extensions possibles des mécanismes de supervision

Exécution d'un arbre HTNi temporisé

Lorsque nous avons défini les arbres HTNi (section 5.1.3 du chapitre 5, p.71), nous n'avons pris en compte que deux relations d'ordre pour une méthode donnée (la séquence, et le non ordonnancement des tâches la composant). Nous pourions aussi nous baser sur une relation temporelle, à savoir la date relative d'exécution d'une tâche entre les tâches d'une même méthode. L'algorithme 5 (chapitre 5, p.67) ne permet pas l'exécution d'un tel HTNi. L'appel vers l'action associée à une tâche élémentaire (ligne 12) est bloquant pour l'exécution, ce qui interdit implicitement la réalisation simultanée de plusieurs tâches du HTNi dont les dates de début seraient les mêmes par exemple. L'exécution de deux tâches (ou plus) peut être aussi lancée à des dates différentes, sans que la première tâche lancée soit nécessairement achevée (on se place ici dans un cadre où l'exécution des tâches est contrainte par une date, plutôt que par des relations de précédence qui imposent qu'une tâche ait terminée de modifier l'état du système pour lancer l'exécution de la suivante). On peut distinguer deux cas de simultanéité. Des tâches peuvent être exécutées à la même date (ou dont l'exécution de l'une chevauche la date de lancement des autres) :

- par des véhicules différents ;
- à bord du même véhicule.

Dans notre travail, la modification du HTNi par l'étape de distribution aux superviseurs locaux (section 5.2 du chapitre 5, p.75) autorise différents membres de l'équipe à exécuter simultanément leurs tâches respectives, lorsque celles-ci sont indépendantes les unes des autres. Prenons par exemple une tâche élémentaire *goto*, considérée comme achevée lorsque le véhicule a atteint sa destination. Deux véhicules distants peuvent effectuer deux *goto* respectifs et non ordonnés en parallèle : le premier véhicule parti ne doit pas nécessairement être arrivé à destination pour que le second effectue son action (suppression de la tâche distante lors du processus de distribution du HTNi). Par contre, le second cas n'est pas traité. Par exemple, un même véhicule peut théoriquement "émettre un signal sonore continu" pendant qu'il se déplace avec un *goto*, deux tâches qui correspondraient à deux services différents. Même s'il existe actuellement des moyens détournés d'autoriser de telles

exécutions en pratique (création d'un service fusionnant les deux capacités du véhicule ou utilisation de services dæmon), ce n'est pas formellement satisfaisant. De plus, des exécutions de tâches peuvent imposer des contraintes temporelles fortes, auxquels cas le lancement de l'exécution d'une tâche doit avoir lieu à un moment précis (par exemple, lorsque deux véhicules sont coordonnés pour exécuter une branche du HTNi et qu'ils ne disposent pas de moyen de communication lors de cette exécution).

Une piste pour dépasser ces limites consisterait à généraliser l'algorithme 5 pour autoriser des exécution de tâches en parallèle. Lorsque deux branches du HTNi sont de priorité identique pour $\Re$ (même date de début par exemple), leur exécution doit être simultanée. Un processus parallèle (un thread) peut être créé pour chacune des tâches qui doivent être exécutées. Dans le cas de tâches non ordonnées, le moteur d'exécution peut également lancer des exécution parallèles pour suivre une politique du "dès que possible", ce qui aurait pour effet de réduire la durée de la mission : l'ensemble des tâches doivent être exécutées dès qu'elles peuvent l'être, même si les exécution précédentes ne sont pas terminées. Le mécanisme de réparation du chapitre 6 devrait alors être également généralisé pour tenir compte des exécutions de plusieurs branches du HTNi simultanées.

8.2.3 Contexte où la bande passante est extrêmement réduite

Dans le contexte d'opérations sous-marines, la bande passante autorisée par modem acoustique, pour les communications quand le véhicule est en plongée, est extrêmement faible (de l'ordre de quelques bits par seconde). Nombre d'auteurs [Sotzing et al., 2008, Belbachir et al., 2012, Carlési et al., 2011] placent leurs architectures dans de tels contextes. Il serait intéressant d'étudier les possibilités qu'offre HiDDeN dans ces conditions, et voir comment une flotille d'AUV, d'ASV et d'AAV pourraient coopérer pour des missions aéromaritimes plus complexes.

8.2.4 Autres utilisations potentielles de Koper

Même si nous n'avons que peu exploité cette possibilité en dehors des phases de réparation détaillées au chapitre 6 (p.89), des tâches de communications peuvent être générées pour les besoins de Koper, notamment dans le cas où le service d'un véhicule a une influence sur des données exploitées par un ou plusieurs autres véhicules. Cela n'a pas été implémenté, mais Koper dispose de toutes les informations nécessaires à ce genre de comportement. Par exemple, un simple *goto* modifie la position du véhicule v qui l'exécute. Dans une mission où la localisation des véhicules à tout moment serait une donnée critique, il serait intéressant d'avertir les véhicules distants que la position du véhicule v a changé, à l'instant où il termine son déplacement.

Un travail sur la synchronisation plus fine des données entre les différentes bases de connaissances locales Koper pourrait être une piste de recherche à exploiter, notamment dans le domaine des systèmes distribués informatiques. En effet, lors de l'implémentation, nous avons utilisé des protocoles d'échange sur étagère tel que "rsync" (remote synchronization) [1] en partant de l'hypothèse que la donnée de date la plus récente [2] était celle à prendre en considération. Dans des environnements dont la bande passante des communications est très réduite, il pourrait s'avérer intéressant de limiter les échanges de données aux seules informations nécessaires à la requête de planification par exemple. Et inversement, lors de phases où les communications sont de bonnes qualités, de tirer parti d'une synchronisation opportuniste par exemple.

1. `http://doc.ubuntu-fr.org/rsync`

2. Et en supposant que les horloges des véhicules étaient synchronisées malgré des écarts de quelques millisecondes dans la réalité

8.2.5 Apprentissage du domaine de planification

Il pourrait s'avérer fructueux d'explorer des domaines liés à l'apprentissage en ce qui concerne les phases de replanification. En effet, pouvoir adapter le comportement du véhicule en fonction de son expérience passée pourrait être un atout en terme de robustesse vis-à-vis des aléas de la mission. Par exemple, nous avons vu (section 6.2.2, p.95) que selon leurs types, les aléas n'apportaient pas toujours suffisamment d'informations pour assurer une planification efficace. Suite aux expériences sur le terrain, nous avons par exemple constaté que dans le cas de l'AGV Mana, et lors d'un échec indéterminé de l'action *goto* (le véhicule n'est ni bloqué, ni perdu, et n'a pas détecté de cible), il peut être bénéfique de renvoyer une action identique. En revanche, un tel comportement sur l'AAV ReSSAC serait plus critique, sachant qu'un échec sur un *goto* d'origine inconnue incite systématiquement à l'annulation de la mission pour des raisons de sécurité. Faire en sorte que le domaine de planification puisse évoluer en fonction de l'historique des retours d'exécution pourrait s'avérer bénéfique, afin d'adapter par exemple les replanifications aux effets constatés des actions sur l'environnement, et pas uniquement en se basant sur leur description statique initiale.

8.2.6 Aspects éthiques d'équipe de véhicules autonomes

Dans le cadre du PEA Action, le rôle de l'opérateur est volontairement restreint. L'un des objectifs de HiDDeN est de procurer toute l'autonomie décisionnelle dont a besoin l'équipe de véhicules autonomes pour réaliser sa mission, et ce, théoriquement sans aucune intervention humaine. Même si en pratique cette autonomie reste relative (pilote de sécurité pour l'AAV ReSSAC dont les commandes ont priorité sur tout ordre envoyé au véhicule, confirmation manuelle de certaines commandes...) et s'avère difficile à mettre en œuvre d'un point de vue opérationnel (nombreux essais nécessaires pour obtenir des comportements nominaux du système), elle n'en reste pas moins tout à fait faisable. Dans Koper, l'opérateur humain peut être représenté comme un "véhicule" au sein de l'équipe. Il dispose alors de services qu'il est capable de fournir pour participer à l'accomplissement de la mission d'équipe. Des questions se posent aussi sur la responsabilité des actions des véhicules autonomes réels par exemple et sur le degré de liberté laissé à l'équipe de véhicules autonomes pour décider des actions à accomplir.

8.2.7 Limites des réparations aussi locales que possibles

La notion de contraintes entre parties du HTNi lors des réparations a également été laissée en suspens : des parties d'un plan peuvent être invalidées lors d'une réparation locale (tâches "sœurs" non concernées par une réparation locale), et donc échouer lors de la tentative du superviseur local de les exécuter. Un mécanisme de prévision de ce genre de situation pourrait être bénéfique lors des phases de réparation de plan.

D'autres mécanismes au niveau des interfaces entre planification et supervision peuvent être exploitables. Des stratégies de réparation pourraient être établies entre les deux fonctions suivant la disponibilité des communications, ou des techniques de vérification pourraient être réalisées par le planificateur, après réparation d'une partie du plan.

8.2.8 Plans conditionnels

Choix dynamique de branches HTNi à l'exécution

Dans la définition des HTNi (section 5.1.3 du chapitre 5, p.71), nous donnons la possibilité à une tâche abstraite de disposer de plusieurs méthodes. Cela permet notamment au superviseur local de choisir une branche lors de l'exécution, en fonction de l'état courant du système, la branche étant calculée hors ligne, sans avoir à effectuer nécessairement une réparation de plan. Des aléas de la mission peuvent ainsi être traités lors de la planification initiale, et éviter ainsi des réparations de plan. Par exemple dans le scénario II du PEA action, l'information qu'un bateau puisse se trouver sur une zone z est connue. Elle est pourtant ignorée dans les plans initiaux générés, et implique une réparation lorsque le bateau est effectivement présent. Un plan conditionnel pourrait tenir compte de cette présence éventuelle, et éviter ainsi une requête vers un planificateur. Des pistes de recherche pourraient s'orienter par exemple sur des domaines de planification probabiliste, où les branches retenues pour le HTNi refléteraient donc le plan probabiliste obtenu.

Heuristiques d'exécution dans des plans conditionnels

Dans le cas où les préconditions de plusieurs méthodes sont valides à l'exécution, l'une des méthodes doit être sélectionnée. Nous avons vu à la section 5.3 du chapitre 5 (p.80) que la procédure MISSIONMANAGER::CHOOSEMETHOD laissait la liberté de définir des heuristiques lors de l'exécution du HTNi. Exploiter de telles heuristiques pourrait s'avérer bénéfique en terme d'efficacité car elles permettraient de guider l'exécution de mission suivant des critères adaptés au contexte (disponibilité des communications, date, état du plan...)

Références

[Alami et al., 1998] Alami, R., Chatila, R., Fleury, S., Ghallab, M., and Ingrand, F. (1998). An architecture for autonomy. *International Journal of Robotics Research (IJRR)*, 17(4) :315–337.

[Atay and Bayazit, 2007] Atay, N. and Bayazit, O. (2007). Emergent task allocation for mobile robots. In *Robotics Science and Systems (RSS)*, Atlanta, GA, USA.

[Baclawski and Simeqi, 2001] Baclawski, K. and Simeqi, A. (2001). Toward Ontology-Based Component Composition. In *OOPSLA Workshop on Behavioral Semantics*, Tampa, FL, USA.

[Barbier et al., 2006] Barbier, M., Barrouil, C., Gabard, J.-F., and Zanon, G. (2006). ProCoSA : a Petri net based software package for autonomous system supervision. In *Application and Theory of Petri Nets and Other Models of Concurrency (ATPN)*, Turku, Finland.

[Basu et al., 2008] Basu, A., Gallien, M., Lesire, C., Nguyen, T.-H., Bensalem, S., Ingrand, F., and Sifakis, J. (2008). Incremental component-based construction and verification of a robotic system. In *European Conference on Artificial Intelligence (ECAI)*, Patras, Greece.

[Belbachir et al., 2012] Belbachir, A., Ingrand, F., and Lacroix, S. (2012). A cooperative architecture for target localization using multiple AUVs. *Intelligent Service Robotics*, 5(2).

[Bigot, 2011] Bigot, D. (2011). Planification et supervision pour un ensemble de véhicules autonomes hétérogènes. Master's thesis, Université de Caen Basse-Normandie / Onera, Toulouse, France.

[Bjerknes and Winfield, 2010] Bjerknes, J. and Winfield, A. (2010). On fault tolerance and scalability of swarm robotic systems. In *International Symposium on Distributed Autonomous Robotic Systems (DARS)*, Lausanne, Switzerland.

[Boddy, 2003] Boddy, M. S. (2003). Imperfect match : PDDL 2.1 and real applications. *Journal of Artificial Intelligence Research (JAIR)*, 20 :133–137.

[Boella and Damiano, 2008] Boella, G. and Damiano, R. (2008). A replanning algorithm for decision theoretic hierarchical planning : principles and empirical evaluation. *Applied Artificial Intelligence*, 22(10) :937–963.

[Bonnet-Torrès and Tessier, 2005] Bonnet-Torrès, O. and Tessier, C. (2005). From teamplan to individual plans : a Petri net-based approach. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, Utrecht, The Netherlands.

[Bonnet-Torrès and Tessier, 2006] Bonnet-Torrès, O. and Tessier, C. (2006). Cooperative team plan : planning, execution and replanning. In *AAAI Spring Symposium on "Distributed Plan and Schedule Management"*, Stanford, CA, USA.

[Bratman et al., 1988] Bratman, M. E., Israel, D. J., and Pollack, M. E. (1988). Plans and resource-bounded practical reasoning. *Computational Intelligence*, 4(3) :349–355.

[Bresina et al., 2005] Bresina, J. L., Jónsson, A. K., Morris, P. H., and Rajan, K. (2005). Activity planning for the Mars exploration rovers. In *International Conference on Automated Planning and Scheduling (ICAPS)*, Monterey, CA, USA.

[Brooks, 1990] Brooks, R. A. (1990). Elephants don't play chess. *Robotics and Autonomous Systems*, 6 :3–15.

[Brooks, 1991] Brooks, R. A. (1991). New approaches to robotics. *Science*, 253(5025) :1227–1232.

[Brugali and Scandurra, 2009] Brugali, D. and Scandurra, P. (2009). Component-based robotic engineering (Part I). *IEEE Robotics Automation Magazine*, 16(4).

[Brummit and Stentz, 1998] Brummit, B. and Stentz, A. (1998). GRAMMPS : A generalized mission planner for multiple mobile robots. In *International Conference on Robotics and Automation (ICRA)*, Leuven, Belgium.

[Carlési et al., 2011] Carlési, N., Michel, F., Jouvencel, B., and Ferber, J. (2011). Generic architecture for multi-auv cooperation based on a multi-agent reactive organizational approach. In *International Conference on Intelligent Robots and Systems (IROS)*, San Francisco, CA, USA.

[Chaimowicz et al., 2001] Chaimowicz, L., Sugar, T., Kumar, V., and Campos, M. (2001). An architecture for tightly coupled multi-robot cooperation. In *International Conference on Robotics and Automation (ICRA)*, Seoul, Korea.

[Chanthery, 2005] Chanthery, E. (2005). *Planification de mission pour un véhicule aérien autonome*. PhD thesis, Institut Supérieur de l'Aéronautique et de l'Espace, Toulouse, France.

[Chien et al., 2006] Chien, S., Doyle, R., Davies, A., Jonsson, A., and Lorenz, R. (2006). The Future of AI in Space. *IEEE Intelligent Systems*, 21(4) :64–69.

[Chien et al., 2004] Chien, S., Sherwood, R., Tran, D., Cichy, B., Rabideau, G., Castano, R., Davies, A., Lee, R., M, D., Frye, S., Trout, B., Hengemihle, J., Shulman, S., Ungar, S., and Brakke, T. (2004). The EO-1 autonomous science agent. In *International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, New York, NYC, USA.

[Coles et al., 2009] Coles, A. J., Coles, A. I., Fox, M., and Long, D. (2009). Extending the use of inference in temporal planning as forwards search. In *International Conference on Automated Planning and Scheduling (ICAPS)*, Thessaloniki, Greece.

[Costelha and Lima, 2007] Costelha, H. and Lima, P. (2007). Modelling, analysis and execution of robotic tasks using petri nets. In *International Conference on Intelligent Robots and Systems (IROS)*, San Diego, CA, USA.

[Crestani and Godary-Dejean, 2012] Crestani, D. and Godary-Dejean, K. (2012). Fault tolerance in control architectures for mobile robots : Fantasy or reality ? In *National Conference on Control Architectures of Robots (CAR)*, Nancy, France.

[Dalgalarrondo, 2001] Dalgalarrondo, A. (2001). *Intégration de la fonction perception dans une architecture de contrôle*. PhD thesis, Université Paris-Sud, centre d'Orsay, Paris, France.

[Degroote and Lacroix, 2011] Degroote, A. and Lacroix, S. (2011). ROAR : Resource oriented agent architecture for the autonomy of robots. *International Conference on Robotics and Automation (ICRA)*.

[Deplanques et al., 1996] Deplanques, P., Yriarte, L., Zapata, R., and Sallantin, J. (1996). An ontology for robot modeling and testing. In *CESA Symposium on Robotics and Cybernetics*, Lille, France.

[Dix et al., 2003] Dix, J., Muñoz-Avila, H., Nau, D., and Zhang, L. (2003). IMPACTing SHOP : putting an AI planner into a multi-agent environment. *Annals of Mathematics and AI*, 37(4).

[Doherty et al., 2009] Doherty, P., Kvarnström, J., and Heintz, F. (2009). A temporal logic-based planning and execution monitoring framework for unmanned aircraft systems. *Journal of Autonomous Agents and Multi-Agent Systems (JAAMAS)*, 19(3) :332–377.

[Echeverria et al., 2011] Echeverria, G., Lassabe, N., Degroote, A., and Lemaignan, S. (2011). Modular open robots simulation engine : MORSE. In *International Conference on Robotics and Automation (ICRA)*, Shangai, China.

[Echeverria et al., 2012] Echeverria, G., Lemaignan, S., Degroote, A., Lacroix, S., Karg, M., Koch, P., Lesire, C., and Stinckwich, S. (2012). Simulating complex robotic scenarios with MORSE. In *International Conference on Simulation, Modeling and Programming for Autonomous Robots (SIMPAR)*, Tsukuba, Japan.

[Erol et al., 1994] Erol, K., Hendler, J., and Nau, D. (1994). HTN planning : complexity and expressivity. In *National Conference on Artificial Intelligence (AAAI)*, Seattle, WA, USA.

[Fazil Ayan et al., 2007] Fazil Ayan, N., Kuter, U., Yaman, F., and Goldman, R. (2007). HOTRiDE : hierarchical ordered task replanning in dynamic environments. In *International Conference on Automated Planning and Scheduling (ICAPS)*, Providence, RI, USA.

[Fleury et al., 1997] Fleury, S., Herrb, M., and Chatila, R. (1997). GenoM : a tool for the specification and the implementation of operating modules in a distributed robot architecture. In *International Conference on Intelligent Robots and Systems (IROS)*, Grenoble, France.

[Fox et al., 2006] Fox, M., Gerevini, A., Long, D., and Serina, I. (2006). Plan stability : Replanning versus plan repair. In *International Conference on Automated Planning and Scheduling (ICAPS)*, Cumbria, UK.

[Fua and Ge, 2005] Fua, C.-H. and Ge, S. S. (2005). COBOS : cooperative backoff adaptive scheme for multirobot task allocation. *IEEE Transactions on Robotics*, 21(6) :1168–1178.

[Gancet, 2005] Gancet, J. (2005). *Systèmes multi-robots aériens : architecture pour la planification, la supervision et la coopération*. PhD thesis, LAAS - CNRS, Toulouse, France.

[Gateau, 2009] Gateau, T. (2009). Supervision de mission pour un ensemble de drones. Master's thesis, Institut National Polytechnique de Toulouse, Toulouse, France.

[Gobillot, 2012] Gobillot, N. (2012). Planification de mission pour la robotique autonome prenant en compte des modèles de l'environnement. Master's thesis, Ecole Polytechnique universitaire de Montpellier / Onera, Toulouse, France.

[Hamilton et al., 1996] Hamilton, D., Walker, I., and Bennett, J. (1996). Fault tolerance versus performance metrics for robot systems. In *International Conference on Robotics and Automation (ICRA)*, Minneapolis, MN, USA.

[Hartley and Pipitone, 1991] Hartley, R. and Pipitone, F. (1991). Experiments with the subsumption architecture. In *International Conference on Robotics and Automation (ICRA)*, Sacramento, CA, USA.

[Heger and Singh, 2002] Heger, F. W. and Singh, S. (2002). A hybrid assembly task planning system : Where motion planning helps symbolic planning find good solutions for real-world applications. In *National Conference on Artificial Intelligence (AAAI)*, Edmonton, AL, Canada.

[Heintz and Doherty, 2004] Heintz, F. and Doherty, P. (2004). Dyknow : An approach to middleware for knowledge processing. *Journal of Intelligent & Fuzzy Systems*, 15(1) :3–13.

[Herzog et al., 2008] Herzog, A., Jacobi, D., and Buchmann, A. (2008). A3ME-an Agent-Based middleware approach for mixed mode environments. In *International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM)*, Valencia, Spain.

[Hogg, 2007] Hogg, C. (2007). From task definition and plan traces to HTN methods. In *International Conference on Automated Planning and Scheduling (ICAPS), Doctoral Consortium*, Providence, RI, USA.

[Huang, 2007] Huang, H.-M. (2007). Autonomy levels for unmanned systems (ALFUS) framework : Safety and application issues. In *Performance Metrics for Intelligent Systems Workshop (PerMIS)*, Gaithersburg, ML, USA.

[Huang et al., 2009] Huang, J., Bastani, F., Yen, I.-L., Dong, J., Zhang, W., Wang, F.-J., and Hsu, H.-J. (2009). Extending service model to build an effective service composition framework for cyber-physical systems. In *International Conference on Service-Oriented Computing and Applications (SOCA)*, Taipei, Taiwan.

[Ilghami and Nau, 2001] Ilghami, O. and Nau, D. (2001). A general approach to synthesize problem-specific planners. In *International Joint Conference on Artificial Intelligence (IJCAI)*, Seattle, WA, USA.

[Ingrand et al., 1992] Ingrand, F., Georgeff, M., and Rao, A. (1992). An architecture for real-time reasoning and system control. *IEEE Expert*, 7(6) :34–44.

[Joyeux et al., 2007] Joyeux, S., Alami, R., and Lacroix, S. (2007). A plan manager for multi-robot systems. In *International Conference on Field and Service Robot Systems (FSRS)*, Chamonix, France.

[Kahn, 1962] Kahn, A. B. (1962). Topological sorting of large networks. *Communications of ACM*, 5(11) :558–562.

[Kannan and Parker, 2007] Kannan, B. and Parker, L. E. (2007). Metrics for quantifying system performance in intelligent, fault-tolerant multi-robot systems. In *International Conference on Intelligent Robots and Systems (IROS)*, San Diego, CA, USA.

[Kazz and Greenberg, 2002] Kazz, G. and Greenberg, E. (2002). Mars Relay Operations : Application of the CCSDS Proximity-1 Space Data Link Protocol. Technical report, JPL - NASA, Pasadena, CA, USA.

[Laborie, 1995] Laborie, P. (1995). *IxTeT : une approche intégrée pour la gestion de ressources et la synthèse de plans*. PhD thesis, Ecole Nationale Supérieure des Télécommunications, Toulouse, France.

[Lefevre, 2008] Lefevre, S. (2008). *La robotisation des armées occidentales modernes - Enjeux et perspectives*. PhD thesis, Université Robert Schuman, Institut d'Études Politiques de Strasbourg, Strasbourg, France.

[Legras and Tessier, 2003] Legras, F. and Tessier, C. (2003). LOTTO : group formation by overhearing in large teams. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, Melbourne, Australia.

[Lortal et al., 2011] Lortal, G., Dhouib, S., and Gérard, S. (2011). Integrating ontological domain knowledge into a robotic DSL. In *International Conference on Models in Software Engineering (MODELS)*, Oslo, Norway.

[Luzeaux and Dalgalarrondo, 2001] Luzeaux, D. and Dalgalarrondo, A. (2001). HARPIC, an hybrid architecture based on representations, perception, and intelligent control : A way to provide autonomy to robots. In *International Conference on Computational Science*, London, UK.

[Magnusson et al., 2008] Magnusson, M., Landén, D., and Doherty, P. (2008). Planning, executing, and monitoring communication in a logic-based multi-agent system. In *European Conference on Artificial Intelligence (ECAI)*, Patras, Greece.

[McDermott et al., 1998] McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., and Wilkins, D. (1998). PDDL : the planning domain definition language. Technical Report CVC TR-98-003, Yale Center for Computational Vision and Control, New Haven, CT, USA.

[McGann et al., 2008] McGann, C., Py, F., Rajan, K., Thomas, H., Henthorn, R., and McEwen, R. (2008). A deliberative architecture for AUV control. In *International Conference on Robotics and Automation (ICRA)*, Pasadena, CA, USA.

[Mercier, 2010] Mercier, S. (2010). *Contrôle du partage de l'autorité dans un systéme d'agents hétérogènes*. PhD thesis, Institut Supérieur de l'Aéronautique et de l'Espace, Toulouse, France.

[Merri et al., 2002] Merri, M., Melton, B., Valera, S., and Parkes, A. (2002). The ECSS Packet Utilization Standard and Its Support Tool. In *SpaceOps Conference*, Houston, TX, USA.

[Micalizio and Torasso, 2008] Micalizio, R. and Torasso, P. (2008). Supervision and diagnosis of joint actions in multi-agent plans. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, Estoril, Portugal.

[Miyata et al., 2002] Miyata, N., Ota, J., and Arai, T. (2002). Cooperative transport by multiple mobile robots in unknown static environments associated with real-time task assignment. In *International Conference on Robotics and Automation (ICRA)*, Washington, DC, USA.

[Montano et al., 2000] Montano, L., Garcia, F., and Villarroel, J. (2000). Using the time petri net formalism for specification, validation, and code generation in robot-control applications. *International Journal of Robotics Research (IJRR)*, 19(1).

[Mosteo et al., 2008] Mosteo, A. R., Montano, L., and Lagoudakis, M. G. (2008). Multi-robot routing under limited communication range. In *International Conference on Robotics and Automation (ICRA)*, Pasadena, CA, USA.

[Muscettola et al., 2002] Muscettola, N., Dorais, G. A., Fry, C., Levinson, R., and Plaunt, C. (2002). IDEA : Planning at the core of autonomous reactive agents. In *International NASA Workshop on Planning and Scheduling for Space (IWPSS)*, Houston, TX, USA.

[Nau et al., 2003] Nau, D., Au, T.-C., Ilhami, O., Kuter, U., Murdock, W., Wu, D., and Yaman, F. (2003). SHOP2 : an HTN planning system. *Journal of Artificial Intelligence Research (JAIR)*, 20.

[Nau et al., 1999] Nau, D., Cao, Y., Lotem, A., and Munoz-Avila, H. (1999). Shop : Simple hierarchical ordered planner. In *International Joint Conference on Artificial Intelligence (IJCAI)*, Stockholm, Sweden.

[Nebel and Koehler, 1993] Nebel, B. and Koehler, J. (1993). Plan modification versus plan generation : A complexity-theoretic perspective. In *International Joint Conference on Artificial Intelligence (IJCAI)*, Chambery, France.

[Paolucci et al., 2000] Paolucci, M., Shehory, O., and Sycara, K. (2000). Interleaving planning and execution in a multiagent team planning environment. *Linköping Electronic Articles in Computer and Information Sciences*, 5(18).

[Parker et al., 2004] Parker, L., Kannan, B., Tang, F., and Bailey, M. (2004). Tightly-coupled navigation assistance in heterogeneous multi-robot teams. In *International Conference on Intelligent Robots and Systems (IROS)*, Sendai, Japan.

[Parker and Tang, 2006] Parker, L. and Tang, F. (2006). Building multirobot coalitions through automated task solution synthesis. *Proceedings of the IEEE*, 94(7) :1289 –1305.

[Parker, 1998] Parker, L. E. (1998). ALLIANCE : An Architecture for Fault Tolerant Multi-Robot Cooperation. *IEEE Transactions on Robotics and Automation*, 14 :220–240.

[Parker, 2000] Parker, L. E. (2000). On the development of metrics for multi-robot teams within the ALLIANCE architecture. In *Workshop on Performance Metrics for Intelligent Systems (PerMIS)*, Gaithersburg, MD, USA.

[Parker, 2008] Parker, L. E. (2008). Distributed intelligence : Overview of the field and its application in multi-robot systems. *Journal of Physical Agents*, 2.

[Petit Larousse, 2011] Petit Larousse (2011).

[Puterman, 1994] Puterman, M. (1994). *Markov decision processes : Discrete stochastic dynamic programming*. John Wiley & Sons, Inc.

[Rao and Georgeff, 1995] Rao, A. S. and Georgeff, M. P. (1995). BDI agents : From theory to practice. In *International Conference on Multi-Agent Systems (ICMAS)*, San Francisco, CA, USA.

[Rekleitis et al., 2004] Rekleitis, I., Lee-shue, V., New, A. P., and Choset, H. (2004). Limited communication, multi-robot team based coverage. In *International Conference on Robotics and Automation (ICRA)*, Barcelona, Spain.

[Rooker and Birk, 2007] Rooker, M. and Birk, A. (2007). Multi-robot exploration under the constraints of wireless networking. *Control Engineering Practice*, 15(4).

[Russel and Norvig, 1995] Russel, S. and Norvig, P. (1995). *Artificial intelligence, a modern approach*. Prentice-Hall, Englewood Cliffs.

[Sariel-Talay et al., 2009] Sariel-Talay, S., Balch, T. R., and Erdogan, N. (2009). Multiple traveling robot problem : A solution based on dynamic task selection and robust execution. *IEEE Transaction on Mechatronics*, 14(2).

[Schlenoff and Messina, 2005] Schlenoff, C. and Messina, E. (2005). A robot ontology for urban search and rescue. In *ACM workshop on Research in knowledge representation for autonomous systems*, New York, NYC, USA.

[Sellami et al., 2011] Sellami, Z., Camps, V., Aussenac-Gilles, N., and Rougemaille, S. (2011). Ontology Co-construction with an Adaptive Multi-Agent System : Principles and Case-Study. In *International Joint Conference on Knowledge Discovery, Knowlege Engineering and Knowledge Management (IC3K)*, Funchal, Portugal.

[Sirin et al., 2003] Sirin, E., Parsia, B., Wu, D., Hendler, J., and Nau, D. (2003). HTN Planning for Web Service Composition Using SHOP2. In *International Semantic Web Conference*, Sanibel Island, FL, USA.

[Smith, 1980] Smith, R. (1980). The Contract Net Protocol : high-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, 29(12).

[Soetens and Bruyninckx, 2005] Soetens, P. and Bruyninckx, H. (2005). Realtime hybrid task-based control for robots and machine tools. In *International Conference on Robotics and Automation (ICRA)*, Barcelona, Spain.

[Sotzing, 2007] Sotzing, C. (2007). A multi-agent architecture to increase coordination efficiency in multi-AUV operations. In *OCEANS*, Aberdeen, Scotland, UK.

[Sotzing et al., 2008] Sotzing, C. C., Johnson, N., and Lane, D. M. (2008). Improving multi-AUV coordination with hierarchical blackboard-based plan representation. In *Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG)*, Edinburgh, UK.

[Stancliff et al., 2005] Stancliff, S., Dolan, J., and Trebi-Ollennu, A. (2005). Mission reliability estimation for repairable robot teams. In *International Conference on Informatics in Control, Automation and Robotics (ICINCO), Workshop on Multi-Agent Robotic Systems (MARS)*, Barcelona, Spain.

[Tambe, 1997] Tambe, M. (1997). Towards flexible teamwork. *Journal of Artificial Intelligence Research (JAIR)*, 7 :83–124.

[Teichteil-Königsbuch et al., 2011] Teichteil-Königsbuch, F., Lesire, C., and Infantes, G. (2011). A generic framework for anytime execution-driven planning in robotics. In *International Conference on Robotics and Automation (ICRA)*, Shanghai, China.

[Tenorth et al., 2012] Tenorth, M., Perzylo, A., Lafrenz, R., and Beetz, M. (2012). The RoboEarth language : Representing and Exchanging Knowledge about Actions, Objects, and Environments. In *International Conference on Robotics and Automation (ICRA)*, Saint-Paul, MN, USA.

[Tramutola and Martelli, 2010] Tramutola, A. and Martelli, A. (2010). Beyond the Aurora architecture for the new challenging applications. Technical report, Thales Alenia Space, Torino, Italy.

[Vidal et al., 2002] Vidal, R., Shakernia, O., Kim, H. J., Shim, D. H., and Sastry, S. (2002). Probabilistic pursuit-evasion games : Theory, implementation and experimental evaluation. *IEEE Transactions on Robotics and Automation*, 18 :662–669.

[Vig and Adams, 2006] Vig, L. and Adams, J. (2006). Multi-robot coalition formation. *IEEE Transactions on Robotics*, 22(4) :637 –649.

[Visentin, 2007] Visentin, G. (2007). Autonomy in ESA Planetary Robotics Missions. Technical report, ESA, Noordwijk, The Netherlands.

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

[Witt et al., 2008] Witt, K., Stanley, J., Smithbauer, D., Mandl, D., Ly, V., Underbrink, A., and Metheny, M. (2008). Enabling Sensor Webs by Utilizing SWAMO for Autonomous Operations. In *NASA Earth Science Technology Conference (ESTC)*, Adelphi, ML, USA.

[Zlot, 2005] Zlot, R. (2005). *An Auction-Based Approach to Complex Task Allocation for Multi-robot Teams*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA.

Publications

[Gateau et al., 2010a] Gateau, T., Lesire, C., and Barbier, M. (2010a). Exécution d'une mission par une équipe de véhicules autonomes hétérogènes. In *Journée des Jeunes Chercheurs en Robotique (JJCR)*, Paris, France.

[Gateau et al., 2010b] Gateau, T., Lesire, C., and Barbier, M. (2010b). Local plan execution and repair in a hierarchical structure of sub-teams of heterogeneous autonomous vehicles. In *National Conference on Control Architectures of Robots (CAR)*, Douai, France.

[Gateau et al., 2010c] Gateau, T., Lesire, C., and Barbier, M. (2010c). Local plan execution and repair in a hierarchical structure of sub-teams of heterogeneous autonomous vehicles. In *International Conference on Automated Planning and Scheduling (ICAPS), Doctoral Consortium*, Toronto, Canada.

[Gateau et al., 2012a] Gateau, T., Lesire, C., and Barbier, M. (2012a). Hidden, une architecture décisionnelle distribuée pour la coopération de véhicules individuellement autonomes. In *Conférence Francophone sur la Reconnaissance des Formes et l'Intelligence Artificielle (RFIA)*, Lyon, France.

[Gateau et al., 2012b] Gateau, T., Lesire, C., and Barbier, M. (2012b). Robust strategies for multi-robot team collaboration under uncertain communications. In *International Symposium on Distributed Autonomous Robotic Systems (DARS), Poster session*, Baltimore, MD, USA.

[Gateau et al., 2013] Gateau, T., Lesire, C., and Barbier, M. (2013). HiDDeN, a High-Level Distributed Control Architecture for Multi-Robot Cooperation Under Communication Uncertainty. In *International Conference on Robotics and Automation (ICRA)*, submitted.

[Gateau et al., 2012c] Gateau, T., Severac, G., Lesire, C., Barbier, M., and Bensana, E. (2012c). Knowledge base for planning, execution and plan repair. In *National Conference on Control Architectures of Robots (CAR)*, Nancy, France.

[Gateau et al., 2012d] Gateau, T., Severac, G., Lesire, C., Barbier, M., and Bensana, E. (2012d). Knowledge base for planning, execution and plan repair. In *International Conference on Automated Planning and Scheduling (ICAPS), Workshop on Planning and Plan Execution for Real-World Systems (PlanEx)*, Atibaia, Brazil.

ANNEXE A

HSST - Structure hiérarchique de sous-équipes

Dans la définition des HTNi du chapitre 5, section 5.1.3 (p.71), nous supposons que pour toute tâche τ de l'arbre, l'ensemble des véhicules impliqués sur cette tâche $veh(\tau)$ est connu. En pratique, cet ensemble n'est pas toujours directement accessible, notamment en fonction du format de plan utilisé. Par exemple, avant d'utiliser des HTNi, nous basions nos exécutions sur des plans formellement moins riches, directement issus de réseaux HTN et qui ne disposaient pas toujours d'une information suffisante au niveau des tâches abstraites considérées (l'affectation d'un véhicule n'y est pas toujours explicite à ce niveau). Suivant notre stratégie de replanification d'équipe (chapitre 6, p.89), seuls les véhicules impliqués dans la réalisation de la tâche sur laquelle l'aléa a été détecté doivent être prévenus et engager un processus de réparation. Il peut donc être avantageux de construire hors ligne une structure permettant à chaque instant de savoir quel véhicule est engagé sur quelles tâches. C'est dans cette optique que nous avons construit une structure hiérarchique de sous-équipes, la HSST (Hierarchical Structure of Sub-Teams). Elle fournit une organisation précise et explicite des véhicules au cours de la mission, consultable pour un opérateur à qui une représentation de telles structures peut s'avérer bénéfique pour le suivi de l'évolution de la mission.

La HSST est un DAG (Directed Acyclic Graph), ou graphe acyclique orienté. Les nœuds correspondent à des sous-équipes. La racine est l'équipe formée par l'ensemble de l'équipe. Les feuilles forment des ensembles composés d'un ou plusieurs véhicules. Un même véhicule peut appartenir à plusieurs extrémités différentes. L'équipe commune à tous les véhicules se retrouve en haut de la hiérarchie, à la racine. Nous ajoutons des relations particulières entre les nœuds d'un même niveau du graphe. Les relations *temporelles* correspondent à une évolution temporelle de la sous-équipe dont les fils sont les nœuds de ce niveau. En effet, les sous-équipes sont dynamiques et des véhicules peuvent s'ajouter, ou quitter une équipe pendant l'exécution de la mission, suivant les besoins définis par le plan HTN initial. Les relations *xor* indiquent que durant l'exécution de la mission, la structure de l'équipe sera de la forme d'une branche ou de l'autre, suivant la méthode du HTN choisie en vue de l'exécution. Le choix est exclusif : dès lors qu'une des branches (en relation *xor* avec une autre branche) de l'arbre est choisie, l'autre branche est ignorée. Enfin, les relations *parallèles* entre nœuds autorisent les sous-équipes qui leur sont asociées à réaliser des exécutions non ordonnées. Une première version de la HSST est construite hors ligne, puis est amenée à évoluer localement en cas de réparation du HTNi, suite à l'occurrence d'un aléa.

La HSST est extraite d'un HTNi. Pour cette extraction, nous proposons l'algorithme 7 dans lequel la tâche de plus haut niveau d'un HTNi $\mathcal{H}$ est passée en paramètre. Cet algorithme permet de déduire les véhicules impliqués par des tâches abstraites, même si l'information n'est pas présente au niveau du HTNi Nous supposerons que pour tout τ tâche de $\mathcal{H}$, $veh(\tau)$ connu $\iff \tau \in Te$.

L'algorithme parcourt de façon récursive un HTNi pour construire la HSST. La rencontre d'une tâche élémentaire est une condition d'arrêt (ligne 3). En effet, les véhicules impliqués dans la réalisation de cette tâche vont former une sous-équipe (ligne 4). La rencontre, au sein du HTNi, d'un embranchement entre plusieurs méthodes va créer autant de sous-équipes possibles que de méthodes différentes via un appel à la procédure EXTRACTMETH précisée par l'algorithme 8 (ligne 8). Cette procédure a en paramètre les méthodes de la tâche abstraite et renvoie une HSST. Une relation *xor* est définie entre les sous-équipes obtenues (ligne 9) : pour A et B deux HSST, la procédure $link(A, B, relE)$ (ligne 9) permet en effet d'ajouter la hiérarchie d'équipe de B dans celle de A, en tenant compte de la relation $relE$. Cette dernière sera symbolisée xor pour la relation du même nom entre sous-équipes, par $\sim$ dans le cas où la relation entre sous-équipe est *parallèles* et par $\prec$ dans le cas où la relation entre sous-équipe est *temporelle*.

Algorithm 7: $Extract(t)$

1 $t \in \{Te \cup Ta\}$
2 $A = \emptyset$
3 **if** $t \in \{Te\}$ **then**
4 RETURN $veh(t)$
5 **else**
6 soit $t = (V(t_a), M(t_a), Post(t_a))$
7 **foreach** $m_i \in M(t_a)$ **do**
8 $A_i = ExtractMeth(m_i)$
9 $link(A, A_i, xor)$
10 RETURN A

L'algorithme 8 permet d'extraire d'une méthode instanciée la structure de sous-équipe associée. Après un appel vers l'algorithme 7 pour extraire la HSST de chaque tâche t_i (ligne 5), la HSST obtenue est liée à celle déjà existente. Si les tâches sont séquentielles, une relation *temporelle* est créée entre les sous-équipes chargées de l'exécution de ces tâches. En effet, les sous-équipes formées sur chacune des tâches vont en fait désigner une même sous-équipe évoluant dynamiquement dans le temps. Dans le cas d'une relation non ordonnée, une relation *parallèle* est créée entre les sous-équipes construites.

Algorithm 8: $ExtractMeth(m)$

1 soit $m = (r, Pm, st, rel)$
2 soit $(t_i)_{1 \leq i \leq |st|} \leftarrow st$ tel que $\forall i \in [\![1..|st| - 1]\!], t_i \prec t_{i+1}$
3 $A = \emptyset$
4 **foreach** $t_i \in (t_i)_{1 \leq i \leq |st|}$ **do**
5 $A_i = Extract(t_i)\ link(A, A_i, rel)$
6 RETURN A

L'algorithme 7 termine : le HTNi est un arbre acyclique qui peut donc être vu comme un graphe, un couple (P,Q) de sommets P correspondant à l'union des ensembles des tâches complexes et élémentaires et des méthodes, et les arêtes, Q, à une partie de P x P, correspondant aux arêtes de notre HTN instancié. Le graphe est connexe (de par la décomposition en sous-tâches successives de la tâche racine), et de par le fait qu'il ne comporte pas de cycles. C'est donc une structure d'arbre fini (les nœuds ont pour étiquettes les tâches abstraites et les méthodes, les feuilles correspondent aux tâches élémentaires, la racine à la tâche de plus haut niveau et les arêtes, les liens hiérarchiques entre les tâches). Par induction, il est donc aisé de prouver que l'algorithme se termine (structure d'arbre fini, sans cycle et dont tous les nœuds ne sont parcourus qu'une fois, en largeur d'abord).

La figure A.1 illustre l'application des algorithmes 7 et 8 à la tâche "R" du HTNi de la figure 6.1. Finalement, la construction d'une HSST permet de savoir directement quelle sous-équipe est

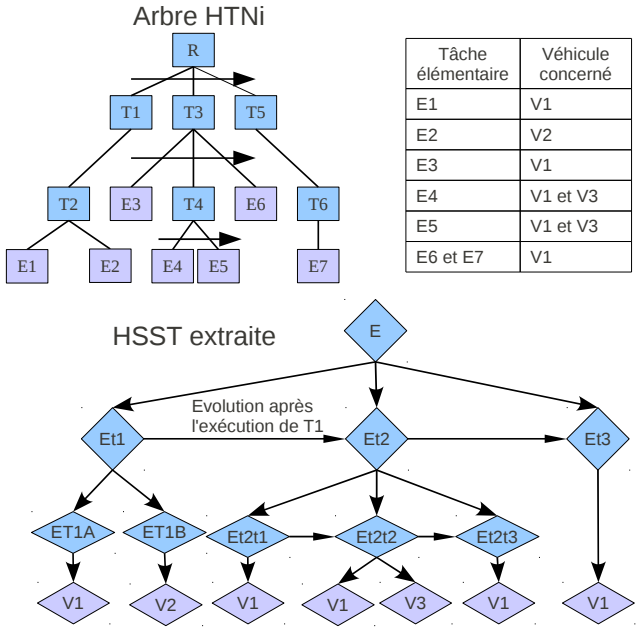


FIGURE A.1 – Exemple de la HSST extraite de l'arbre HTNi de la figure 6.1 du chapitre 6 (p.89), reproduit en haut à gauche. La sous-équipe $Et1$ est chargée des tâches $E1$ et $E2$, respectivement confiées aux véhicules $V1$ et $V2$. Après l'exécution de $T1$, la structure est modifiée en $Et2$: tout 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 $V1$.

responsable de quelle tâche du HTNi, en particulier lors des réparations. Seuls les coéquipiers de la même sous-équipe doivent être prévenus en cas d'aléas, évitant ainsi des communications non fondamentales au processus de réparation vers des véhicules qui ne seraient pas concernés. Son existence nous permet de disposer d'une description plus formelle des sous-équipes qui sont amenées à se créer et à évoluer dynamiquement durant le déroulement de la mission. A chaque réparation, un sous-ensemble de la HSST doit être mis à jour, en fonction des réparations du HTNi. L'implémentation de HiDDeN exploite cette structure, ce qui lui permet d'éviter des parcours complets de l'arbre HTNi en cours de mission.