Distributed Collaborative System for Heterogeneous Swarms of Autonomous Mobile Robots

Vincent Autefage, Serge Chaumette, Damien Magoni

Distributed Collaborative System for Heterogeneous Swarms of Autonomous Mobile Robots

*Vincent Autefage**
Univ. Bordeaux, LaBRI
Talence, France
autefage@labri.fr

Serge Chaumette
Univ. Bordeaux, LaBRI
Talence, France
serge.chaumette@labri.fr

Damien Magoni
Univ. Bordeaux, LaBRI
Talence, France
magoni@labri.fr

***Résumé*—For a few years, several research projects have focused on swarms of autonomous mobile robots (e.g., aircraft vehicles, ground robots). The overall cost (including price, weight, energy, etc.) of the payload required by some missions is sometimes too important to enable the robots of the swarm to embed all the required capabilities (i.e., sensors and actuators). This is the reason why it is more suitable to spread all the capabilities among the robots of the swarm. In this case, it is necessary to implement a collaborative system inside the swarm in order to share the capabilities among them. This collaborative system should provide task allocation as well as management of conflicts and failures which can occur at any moment. In this paper, we present a novel collaborative system for heterogeneous swarms of autonomous mobile robots called AMiRALE (*Asynchronous Missions Relay for Autonomous and Lively Entities*). Our system if fully distributed and relies only on asynchronous communications. We provide a formal description of our proposal based on a graph relabeling system. We also present experimental results obtained through both simulation and emulation. Our evaluation is based on a park cleaning scenario which relies on autonomous and specialized ground robots.**

***Keywords*—Autonomous Swarms ; Heterogeneity ; Mobility ; Graph Relabeling System ; Collaborative System.**

I. Introduction

Un système autonome [1] (*Unmanned System*, UMS) est une entité mobile possédant un degré variable d'autonomie dans ses décisions et actions possibles. Depuis quelques années, un certain nombre d'applications impliquant notamment des robots terrestres (*Unmanned Ground Vehicle*, UGV) ou encore des drones (*Unmanned Aircraft Vehicle*, UAV) ont été développées aussi bien dans le domaine civil que militaire. Certaines de ces applications reposent sur l'utilisation de plusieurs engins, formant ainsi une flotte. Nous pouvons ainsi citer le projet *swarmie* [2] de la NASA qui utilise plusieurs engins terrestres afin d'optimiser l'exploration d'une zone inconnue. Le projet CARUS [3] utilise quant à lui une flotte de plusieurs drones afin d'opérer la surveillance d'une zone.

Toutes ces applications reposent sur des flottes d'appareils homogènes. En effet, les éléments qui composent la flotte ont une configuration et des propriétés identiques. Nous nous intéressons ici aux flottes hétérogènes, i.e. dont les éléments peuvent être différents tant sur le plan matériel que logiciel. En raison des coûts de fabrication, des limites d'espace, de poids ainsi que des contraintes de consommation énergétique, il est bien souvent impossible d'emporter la totalité des capacités (i.e. capteurs et actionneurs) nécessaire à la réalisation d'une mission complexe sur tous les engins [3]. Répartir ces capacités sur l'ensemble des appareils qui composent la flotte est donc une solution naturelle à ce problème. Dès lors, il est nécessaire d'implémenter un mécanisme de collaboration permettant aux engins de la flotte de faire appel aux capacités des autres appareils.

Dans cet article, nous proposons un nouveau mécanisme de collaboration pour flottes hétérogènes appelé AMiRALE (*Asynchronous Missions Relay for Autonomous and Lively Entities*). Notre système est entièrement distribué et repose uniquement sur des messages asynchrones. Cette configuration, comparée à une approche centralisée, procure un certain nombre d'avantages tels que l'exploitation naturelle du parallélisme, la résilience et le passage à l'échelle [4].

Notre contribution s'articule autour des points suivants. Nous proposons en section II un état de l'art sur les différents types de systèmes collaboratifs existants ainsi que leurs architectures respectives. Nous fournissons une description détaillée de notre modèle collaboratif distribué AMiRALE en section III. En section IV, nous présentons à la fois des résultats de simulations et d'expérimentations virtuelles sur un scénario de nettoyage de parc à l'aide de robots terrestres spécialisés. Nous concluons en section V.

II. État de l'art

AMiRALE est un modèle *collaboratif* assimilable à une intelligence artificielle distribuée. L'intelligence artificielle distribuée et plus particulièrement les systèmes multi-agents font référence à un ensemble d'entités (i.e. agents) travaillant conjointement dans le but de résoudre des problèmes plus ou moins complexes [5]. Il existe plusieurs types d'interactions possibles entre ces entités [6].

Un modèle de *coordination* implique que les entités n'aient pas d'objectif commun. Dans ce modèle, chaque entité travaille de manière indépendante et ses actions n'ont pas d'impact sur les autres membres de la flotte. Les agents peuvent néanmoins communiquer afin de réduire leur gêne mutuelle (e.g. robots partageant un espace commun).

Un modèle *collectif* implique que les entités n'interagissent pas directement les unes avec les autres ; i.e. il n'y a pas de communication entre les agents. Leurs actions individuelles

Ce travail est co-financé par la *Direction Générale de l'Armement* et la *Région Aquitaine*.

sont néanmoins bénéfiques pour l'ensemble (ou du moins une partie) de la flotte (e.g. butinage, maintien de formation).

Un modèle *coopératif* implique une communication entre les entités qui composent la flotte. L'ensemble des entités réalise un objectif commun en prenant éventuellement en compte les capacités de chacun des agents (e.g. plusieurs robots poussant un objet).

Un modèle *collaboratif* est une extension du modèle coopératif où les entités ont des objectifs individuels et dont les actions sont bénéfiques pour les autres membres de la flotte. Le modèle *Blackboard* [7] est un exemple de système collaboratif. Ce modèle est particulièrement approprié pour résoudre des problèmes nécessitant une expertise particulière [8]. Il repose sur un groupe d'entités expertes qui travaillent conjointement sur un problème commun (comportement coopératif). Ce problème est subdivisé en une multitude de tâches indépendantes qui peuvent être traitées par les différentes entités en fonction de leurs capacités respectives (comportement collaboratif). Quand une entité estime pouvoir résoudre une des tâches, cette dernière est verrouillée sur le Blackboard ; i.e., seul l'agent ayant déposé le verrou est autorisé à réaliser la tâche. Lorsque celle-ci est accomplie, elle est supprimée définitivement du Blackboard. Ce modèle a été largement étudié et est utilisé dans plusieurs domaines de l'informatique depuis de nombreuses années [9]. Le Blackboard repose sur une architecture centralisée à base de mémoire partagée, ce qui implique que tout événement, action ou connaissance provenant d'un agent particulier est immédiatement connu et pris en compte par l'ensemble des autres entités de la flotte. Cette architecture permet une dissémination parfaite de l'information dans la mesure où les nœuds disposent des mêmes données à tout moment. L'implémentation du Blackboard par mémoire partagée pour les flottes autonomes est néanmoins irréaliste dans la mesure où le modèle ne considère pas les communications entre les nœuds [9].

AMiRALE repose quant à lui sur un modèle collaboratif distribué utilisant uniquement des communications asynchrones entre les nœuds de la flotte.

III. Modèle théorique

AMiRALE est un mécanisme basé sur la notion de mission et qui permet à une flotte d'engins (e.g. drones, robots terrestres) extrêmement dynamiques de réaliser un ensemble de tâches de manière collaborative. Un certain nombre d'évènements pré-enregistrés (e.g. seuil de température dépassé, son anormal, mouvement suspect) peuvent entraîner une action à effectuer par la flotte (e.g. envoi d'informations vers l'extérieur, activation d'une routine). Nous appelons ici *mission* l'ensemble des informations relatives à un (ou des) événement(s) qui vont devoir être échangées afin d'appliquer l'action à effectuer.

Soit un événement de type e (i.e. e représente la nature de l'événement), $Sens_e$ (*capteur*) est une entité capable de capturer l'évènement et de générer la mission $m_e^{n:k}$ où n est l'identifiant du créateur de la mission, k est un numéro de séquence local au nœud n, incrémenté à chaque nouvelle mission créée. $Solv_e$ (*solveur*) est une entité capable de résoudre une mission de type e. Un troisième type de nœuds appelé $Forw_e$ (*relais*) ne fait que transmettre les informations relatives à une mission aux autres entités de la flotte. Pour un type e, si un nœud n'est ni $Sens_e$, ni $Solv_e$, il est par défaut $Forw_e$. La flotte pouvant gérer plusieurs types de missions, un même nœud peut avoir des rôles distincts pour des types différents de missions. En effet, pour deux types d'événements différents *e* et *f*, un nœud peut être $Sens_e$ et $Solv_f$. En outre, un nœud $Sens_e$ n'est pas nécessairement $Solv_e$ et réciproquement. Cette particularité est due aux limitations physiques potentielles des appareils utilisés (e.g. poids, taille, capacité énergétique).

Chaque nœud possède un identifiant unique, une mémoire et sa propre horloge (nous considérons ici que les horloges sont synchronisées, par GPS par exemple). Une mission est représentée par un 7-uplet $\{e, k, n, t, s, n', t'\}$ où e est le type de la mission, k le numéro de séquence de la mission et n l'identifiant du nœud créateur. $\{e, k, n\}$ représente donc l'identifiant unique de la mission. t est la date de création de la mission, n' est l'identifiant du nœud qui a mis à jour la mission pour la dernière fois et t' est la date de dernière modification. s représente l'état courant de la mission et peut prendre 5 valeurs distinctes : *start* (mission en attente de traitement), *will* (mission récupérée par un solveur mais pas encore en traitement), *do* (mission en traitement), *abort* (mission abandonnée), *end* (mission terminée). Ces états sont strictement ordonnés ; i.e. $start < will < do < abort < end$. Une mission $m_e^{n:k}$ ne peut être créée (état *start*) que par un $Sens_e$. L'état de cette mission ne peut être modifié que par un $Solv_e$. Les $Forw_e$ sont des nœuds en lecture seule. Les données contenues dans les missions ne sont pas représentées dans le modèle.

À intervalle de temps régulier, les nœuds diffusent à leurs voisins des parties de leur mémoire ; i.e. un sous-ensemble des missions dont ils ont connaissance (ce sous-ensemble peut dans certains cas représenter l'intégralité de la mémoire). Cela permet aux nœuds d'échanger et de mettre à jour les versions (i.e. avancées) des différentes missions que la flotte doit effectuer. Une mission est diffusée sous la forme d'une *vue* $v_e^{n:k}$ qui est une forme réduite de la mission $m_e^{n:k}$. Il existe plusieurs versions d'AMiRALE dont la différence est le traitement des informations temporelles dans les vues [10]. Nous considérons ici que les nœuds possèdent des horloges synchronisées, par conséquent $m_e^{n:k} = v_e^{n:k}$ dans cette version du modèle. Les vues sont agrégées sous la forme d'un unique message diffusé au travers d'un *broadcast*.

Lorsqu'un nœud reçoit une nouvelle version d'une mission (i.e. état plus avancé) dont il connaît l'existence, il met à jour sa version locale et diffuse la nouvelle version. Lorsqu'un $Solv_e$ a connaissance d'une mission $m_e^{n:k}$ dont l'état est *start*, il modifie cet état en *will* et se prépare à résoudre la mission (e.g. en se déplaçant vers une position précise si nécessaire). Quand il commence à résoudre la mission, il fait évoluer son état en *do*. Si le solveur détecte que la mission a déjà été résolue (e.g. un objet devant être ramassé a disparu), il fixe l'état de la mission en *abort*. Enfin, quand le solveur estime que la mission est terminée, il fixe l'état relatif à *end*. Un solveur ne peut être engagé en *will* ou en *do* que dans une seule mission au même moment. De plus, un solveur ne peut conserver l'état *will* (resp. *do*) qu'un certain laps de temps Ψ^e_{will} (resp. Ψ^e_{do}). Après ce seuil, le solveur abandonne la mission au profit d'un autre solveur s'il est informé que ce

dernier a pris la main sur la mission concernée. Enfin, si un solveur est informé qu'une mission dans laquelle il est engagé, a évolué vers un état supérieur, il abandonne la mission et met à jour sa mémoire locale.

AMiRALE a été formalisé en utilisant une version étendue d'ADAGRS [3] [11] qui est un formalisme de ré-étiquetage de graphes dynamiques basé uniquement sur des communications asynchrones ainsi que des calculs locaux. Nous présentons en figure 1 les différents types de règles possibles du modèle. Ces règles décrivent les interactions entre un nœud et une mission de type e. Pour chacune des règles, les 2 cercles représentent le rôle (i.e. capteur, solveur et relais) du nœud avant et après avoir appliqué la règle. L'ensemble en dessous de chaque cercle représente la version courante de la mission $m_e^{n:k}$. La règle est appliquée uniquement si la condition c est satisfaite. Par simplification, nous avons ajouté un rôle Any_e qui peut s'appliquer à n'importe lequel des rôles précédemment décrits.

1) Cette règle est la seule permettant la création d'une mission. Un capteur de type e nommé n créé la mission $m_e^{n:k}$ si la condition c est vérifiée. Cette condition peut, par exemple, être utilisée pour vérifier si une mission similaire est déjà en cours de résolution.
2) Cette règle permet à un solveur de modifier la version courante d'une mission. Après l'application de cette règle, la mission $m_e^{n:k}$ est mise à jour à la nouvelle version $m_e'^{n:k}$
3) Cette règle permet à un solveur de réagir à un événement applicatif relatif à la mission $m_e^{n:k}$. Un événement applicatif est une indication émise par l'application qui permet de faire évoluer l'état de la mission (e.g. robot prêt à traiter la mission $m_e^{n:k}$, i.e. phase de préparation terminée, qui passe donc de l'état $will$ à l'état do).

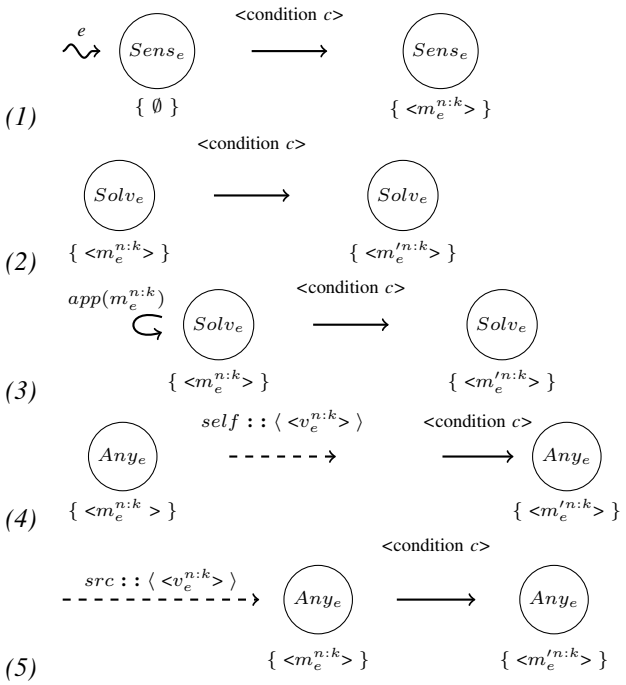

FIGURE 1. Types de règles de ré-étiquetage du modèle AMiRALE.

4) Cette règle permet à un nœud de diffuser sa version locale d'une mission. Cette diffusion est décrite par $self :: v_e^{n:k}$ où $self$ représente l'identifiant du nœud émetteur et $v_e^{n:k}$ est la vue de la mission $m_e^{n:k}$.
5) Cette règle permet à un nœud de mettre à jour une mission suite à la réception d'une version plus avancée. Le message reçu est décrit par $src :: v_e^{n:k}$ où src est l'identifiant de l'émetteur du message.

Certaines décisions internes du mécanisme collaboratif sont dépendantes de l'application. Par exemple, créer ou non une nouvelle mission suite à un événement déjà considéré ou encore ne pas diffuser une mission car celle-ci a déjà été diffusée suffisamment, sont des décisions qui dépendent entièrement de l'application. Par conséquent, AMiRALE inclut des fonctions utilisateurs appelées *filtres* permettant de personnaliser et de configurer finement les mécanismes de décisions internes du modèle en tenant compte des spécificités de l'application. Le comportement de chaque filtre peut être différent pour des types de missions distincts. La liste complète des filtres du modèle est détaillée ci-dessous.

— $f_{ignore}^e(m_e^{n:k})$ permet de décider si la création de la mission $m_e^{n:k}$ doit être avortée lors de la détection d'un événement de type *e* par un capteur $Sens_e$. Ce filtre permet notamment d'empêcher la création d'une nouvelle mission en réponse à un événement déjà en cours de traitement.
— $f_{select}^e(self : m_e^{n:k}, src : v_e^{n:k})$ permet de déterminer localement au solveur *self* si celui-ci doit abandonner le traitement de la mission $m_e^{n:k}$ au profil du solveur *src* qui a diffusé la vue $v_e^{n:k}$ et qui est également en cours de traitement sur la même mission $m_e^{n:k}$. Ce filtre permet ainsi de gérer les conflits sur une même mission entre solveurs.
— $f_{blind}^e(m_e^{n:k})$ permet d'empêcher la diffusion d'une mission. Ceci peut permettre de réduire le volume de trafic réseau généré et donc de limiter les risques de collision et de congestion.
— $f_{pass}^e(m_e^{n:k})$ permet au solveur de ne pas sélectionner la mission $m_e^{n:k}$. Cette fonction permet, entre autres, d'implémenter un ordonnanceur de sélection de missions ou encore de prendre en compte certains éléments (e.g. niveau de batterie, distance avec une cible).
— $f_{check}^e(v_e^{n:k})$ permet d'ignorer la vue $v_e^{n:k}$ reçue. Cette fonction permet d'implémenter un mécanisme de vérification (e.g. signature des messages) ou de politique de sécurité (e.g. certains nœuds ne sont pas autorisés à diffuser certains types de missions).

Nous fournissons dans les figures 2 à 7 l'ensemble des règles de ré-étiquetage du système. Ces règles utilisent le formalisme décrit en figure 1. Nous notons *self* l'identifiant du nœud qui applique la règle, *src* l'identifiant émetteur du message courant, *now* la date courante par rapport à l'horloge du nœud *self*. Cette date est néanmoins la même sur l'ensemble des nœuds dans la mesure où les horloges sont synchronisées. Enfin, *free()* indique si le nœud courant n'est pas impliqué dans une mission (i.e. états *will* ou *do*). Les dates contenues dans les missions sont stockées sous forme d'entiers représentant la durée écoulée depuis un point d'origine prédéfini, le plus courant étant le temps UNIX (i.e. 01/01/1970 à 0H00 UTC).

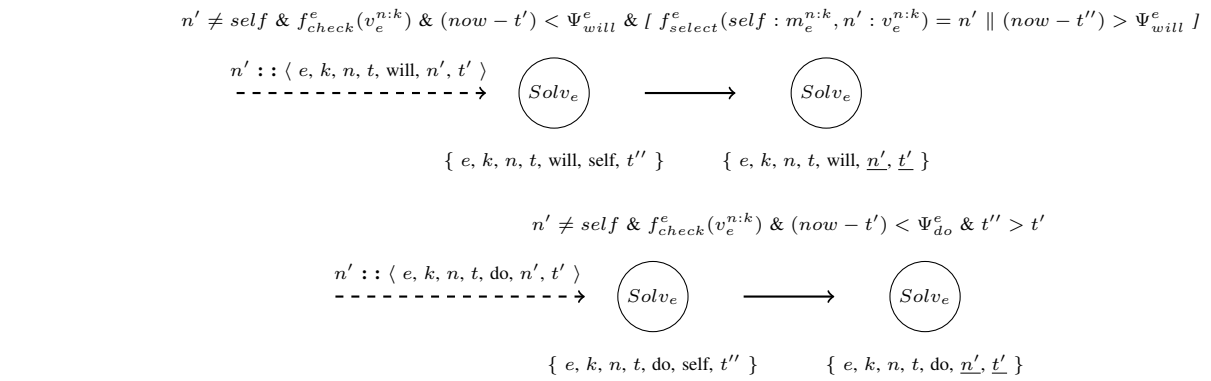

FIGURE 7. Règles relatives au traitement des vues par les solveurs.

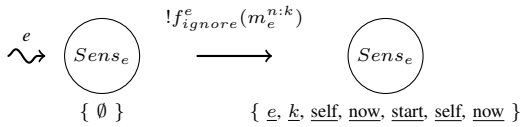

FIGURE 2. Règle relative aux capteurs.

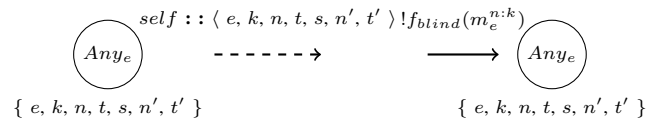

FIGURE 3. Règle relative à l'envoi d'une vue.

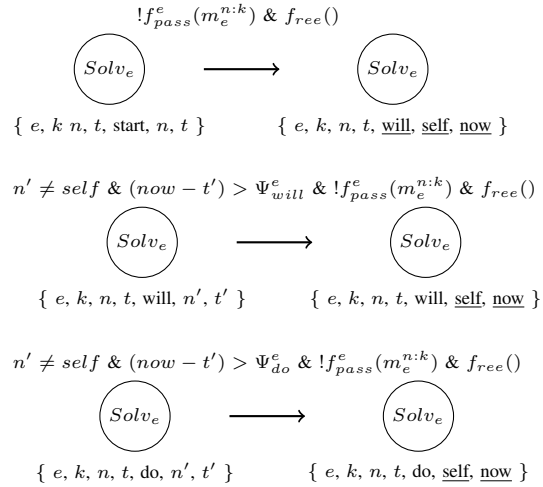

FIGURE 4. Règles relatives aux traitements locaux des solveurs.

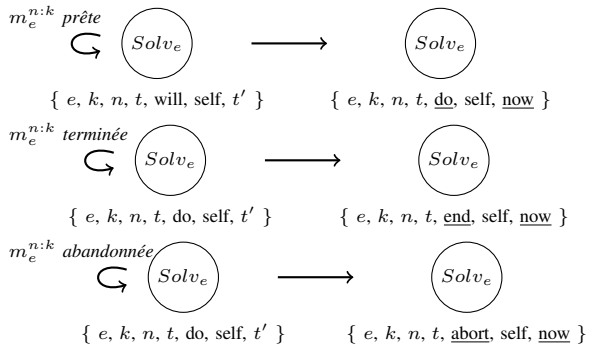

FIGURE 5. Règles relatives aux événements applicatifs des solveurs.

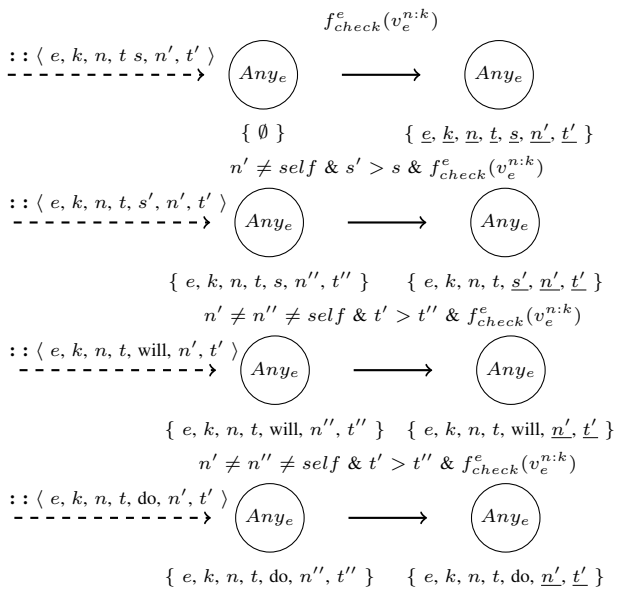

FIGURE 6. Règles relatives au traitement global des vues.

IV. EVALUATION

A. Simulations

Afin d'évaluer les performances de notre modèle, nous présentons dans cette section des résultats de simulations basées sur un scénario de nettoyage de parc [12]. Dans ce scénario, un groupe de robots terrestres est chargé de récupérer les déchets au sol dans un parc ; chaque robot étant spécialisé dans le traitement d'un type de déchets (e.g. verre, plastique, papier, compost). Cette spécialisation des robots caractérise l'hétérogénéité de la flotte. Un robot peut détecter n'importe quel type de déchet mais ne peut en traiter qu'un seul. Les déchets ainsi que les robots sont répartis de manière uniforme dans le parc au début du scénario.

Ces simulations ont été effectuées dans un simulateur de graphes dynamiques appelé JbotSim [13]. C'est une bibliothèque Java permettant de concevoir finement des scénarios utilisant des objets mobiles hétérogènes et communicants. Nous évaluons ici les performances d'AMiRALE en comparaison avec une méthode collective et une méthode collaborative centralisée.

Une méthode collective implique que les robots ne puissent pas communiquer les uns avec les autres. Dans cette solution, chaque robot se déplace de manière aléatoire, ramassant les déchets qu'il est capable de traiter. Cette approche est la pire dans le sens où aucune collaboration n'a lieu entre les entités de la flotte [6]. Nous appelons par la suite ce modèle *Mute*.

La méthode collaborative centralisée à laquelle nous nous intéressons est le Blackboard à mémoire partagée présenté en section II. Dans cette solution, les robots se déplacent également de manière aléatoire ramassant les déchets qu'ils sont aptes à traiter. Lorsqu'un robot détecte un déchet incompatible, la position de ce dernier est référencé dans le Blackboard. Chaque robot consulte périodiquement le contenu du Blackboard à la recherche du déchet compatible le plus proche. Lorsqu'un tel déchet est disponible, il est verrouillé par le robot qui se déplace alors directement vers sa position. Lorsque le déchet est supprimé, l'information correspondante est effacée du Blackboard. La préemption n'est pas possible, ce qui veut dire qu'un déchet verrouillé ne peut être traité que par le robot qui a posé le verrou. Dans le cas d'une panne d'un des robots, le déchet potentiellement verrouillé par ce dernier est immédiatement déverrouillé afin d'éviter tout risque d'interblocage.

AMiRALE est utilisé pour permettre la collaboration entre les différents robots. Le type de mission *e* correspond au type de déchets. Un robot pouvant traiter un déchet de type *e* est donc un $Solv_e$. Pour tout type de mission *i*, chaque robot est $Sens_i$. Des optimisations sont faites par l'intermédiaire des filtres du modèle (présentés en section III) afin d'améliorer l'efficacité du scénario. Quand un capteur détecte un déchet, il ne créé la mission relative que si sa mémoire ne contient pas de mission en cours relative à ce même déchet (filtre f^e_{ignore}). Un solveur sélectionne toujours la mission relative au déchet le plus proche de sa position (filtre f^e_{pass}). Un nœud empêche la diffusion d'une mission terminée depuis plus de 1000 secondes (filtre f^e_{blind}). Si deux solveurs traitent une même mission et que ces derniers sont assez proches pour communiquer directement, le solveur le plus loin du déchet abandonne la mission au profil de l'autre nœud (f^e_{select}). Un solveur est autorisé à être engagé sur une mission pendant 1000 secondes (Ψ^e_{will} et Ψ^e_{do}). Chaque nœud diffuse l'ensemble de ses vues toutes les 5 secondes.

Pour ces trois solutions (Mute, Blackboard et AMiRALE), les robots se déplacent suivant le modèle de mobilité *random waypoint* [14]. Nos paramètres de simulation sont décrits dans le tableau I. Nous évaluons tout d'abord le temps d'exécution nécessaire au nettoyage de la totalité du parc en fonction de plusieurs paramètres tels que le nombre de déchets par type, le nombre de robots par type ainsi que la fréquence des pannes des robots. Une panne consiste en la suppression d'un robot (et donc la perte de sa mémoire locale) et son remplacement immédiat par un robot de même type (à mémoire initiale vide).

La figure 8 montre le temps total de nettoyage du parc pour les différentes solutions (Mute, Blackboard et AMiRALE) en fonction du nombre de déchets par type. Le nombre de types ainsi que le nombre de robots par type sont fixés à 6 (i.e. 36 robots au total). Comme nous pouvions le prévoir, le temps d'exécution est plus important quand le nombre de déchets à traiter augmente. Les valeurs importantes des écart-types présents sur la courbe Mute sont inhérentes à l'aspect

TABLE I. PARAMÈTRES DE SIMULATION

Paramètre	Valeur
Taille du parc	1000 m x 1000 m
Taille des robots	1 m x 1 m
Vitesse des robots	5 m / sec
Rayon de détection des robots	30 m
Rayon de communication des robots	30 m
Nombre d'exécutions par simulation	200

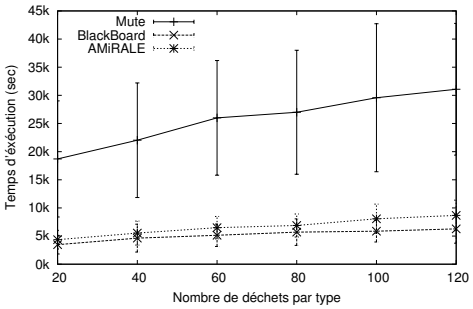

FIGURE 8. Temps total d'exécution pour 6 types et 6 robots/type en fonction du nombre de déchets par type.

purement aléatoire de ce modèle. En effet, dans ce scénario, chaque robot se déplace aléatoirement et indépendamment des autres, ramassant les déchets compatibles lors de son passage. Le temps d'exécution est fortement couplé à la position initiale des robots, la distribution des déchets sur la carte ainsi qu'aux changements de direction induits par le modèle de mobilité. Ces raisons conduisent à des temps d'exécution très fluctuants pour la solution Mute. Nous pouvons également remarquer que les résultats du Blackboard et ceux d'AMiRALE sont similaires. Le Blackboard reste néanmoins plus efficace dans la mesure où aucune communication n'est nécessaire pour disséminer les informations sur les déchets découverts ; ce modèle ne souffre donc pas des effets de distance et de partage d'informations. De plus, le Blackboard n'implémente pas de méthode de synchronisation entre les nœuds. En effet, reposant sur une mémoire partagée, aucune information relative au même déchet ne peut être différente sur deux robots de la flotte. Les résultats du Blackboard sont néanmoins proches de ceux d'AMiRALE bien que ce dernier repose uniquement sur des opérations locales et des communications asynchrones.

La figure 9 montre le temps total de nettoyage du parc pour les différentes solutions en fonction du nombre de robots par type. Le nombre de types ainsi que le nombre de déchets par type sont fixés respectivement à 6 et à 60 (i.e. 360 déchets au total). Nous pouvons constater que l'augmentation du nombre de robots réduit le temps total d'exécution du scénario. Le modèle collectif Mute fournit des résultats médiocres en comparaison de ceux d'AMiRALE et du Blackboard qui sont très similaires.

La figure 10 montre le temps total de nettoyage du parc pour les différentes solutions en fonction de la fréquence des pannes des robots. Le nombre de types ainsi que le nombre de robots par type sont fixés à 6 tandis que le nombre de déchets par type est fixé à 60 (i.e. 36 robots et 360 déchets au total). À chaque panne, un robot choisi aléatoirement est supprimé du parc et remplacé par un robot de même type mais dont la mémoire est vide (i.e., sans information sur les missions en cours ou terminées). Comme on pouvait le prévoir, les effets des pannes sur le Blackboard sont inexistants

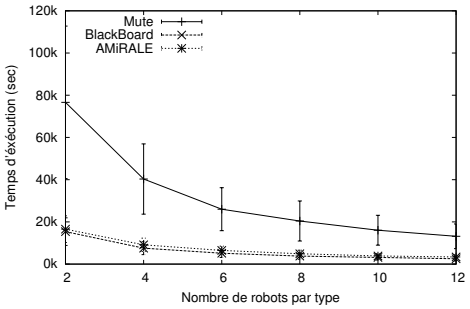

FIGURE 9. Temps total d'exécution pour 6 types et 60 déchets/type en fonction du nombre de robots.

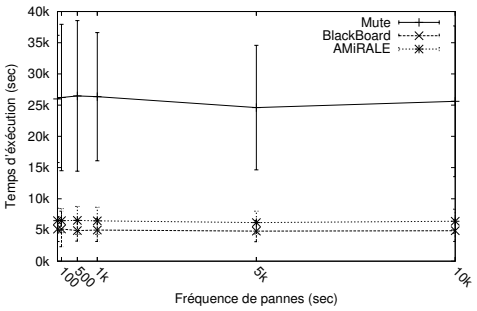

FIGURE 10. Temps total d'exécution pour 6 types, 6 robots/type et 60 déchets/type en fonction de la fréquence de pannes des robots.

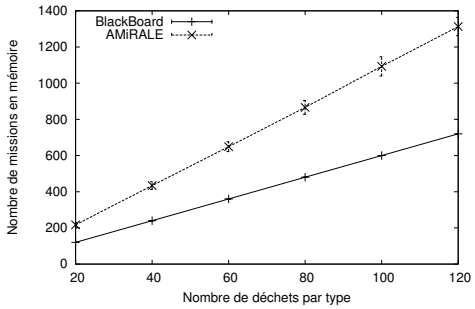

FIGURE 11. Nombre de missions en mémoire par robot en fin de scénario en fonction du nombre de déchets par type.

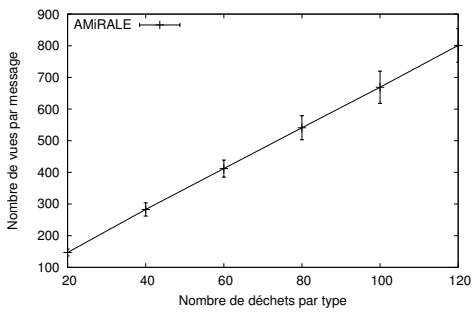

FIGURE 12. Nombre de vues par message en fonction du nombre de déchets par type.

dans la mesure où la mémoire des nœuds est partagée. Ainsi, la perte d'un robot n'implique aucune perte d'informations. Les effets sur la méthode collective Mute sont plus marqués. Les valeurs importantes des écart-types sont dues au caractère purement aléatoire des mouvements des robots. Nous pouvons constater que les pannes n'ont qu'un effet négligeable sur les performances du modèle AMiRALE. Ces résultats sont valables pour une fréquence des pannes aussi bien faible qu'importante (i.e. toutes les 100 secondes). Nous pouvons donc affirmer que notre système AMiRALE est tolérant aux pannes dans le cadre des paramètres de ce scénario.

Nous avons également évalué les performances en termes de consommation mémoire et de volume des messages échangés par les robots dans notre scénario de nettoyage de parc. Ces simulations ne peuvent pas être effectuées sur le modèle Mute car c'est un modèle collectif sans communication ni mémoire.

La figure 11 montre la moyenne du nombre de missions en mémoire en fin de nettoyage du parc pour les solutions Blackboard et AMiRALE en fonction du nombre de déchets par type. Le nombre de types ainsi que le nombre de robots par type sont fixés à 6 (i.e. 36 robots au total). Nous pouvons constater qu'AMiRALE génère environ deux fois plus de missions qu'il y a de déchets ; le nombre de déchets étant rigoureusement égal au nombre de missions du Blackboard. En effet, deux capteurs peuvent potentiellement initier une mission pour un même déchet malgré la présence du filtre f^e_{ignore}. Ce filtre empêche la création d'un mission uniquement si le capteur a connaissance de l'existence d'une mission équivalente. Or, en raison de l'aspect distribué de notre modèle, un nœud ne possède qu'une vision locale de la connaissance globale de la flotte. C'est la raison pour laquelle des missions supplémentaires peuvent être présentes en mémoire.

La figure 12 montre la moyenne du nombre de vues par message en fin de nettoyage du parc en fonction du nombre de déchets par type. La configuration est la même qu'en figure 11. Le Blackboard utilisant une mémoire partagée, nous ne pouvons pas fournir de résultats pour ce modèle. Nous pouvons constater les effets du filtre f^e_{blind} qui empêche la diffusion d'une mission si celle ci est terminée depuis plus de 1000 secondes. Une configuration plus fine de ce filtre à l'aide de techniques de contrôle de congestion [15] (e.g. probabiliste, compteur) peut réduire de manière substantielle le nombre de vues diffusées et donc réduire le risque de congestion réseau [16].

B. *Expérimentations*

En complément des résultats de simulations, nous présentons dans cette section des résultats d'expérimentations en environnement virtuel portant sur la consommation réseau de notre modèle AMiRALE. Nous avons utilisé l'environnement virtuel NEmu [17][18][19][20] qui permet d'émuler un réseau de machines virtuelles mobiles avec un contrôle accru des paramètres réseaux et de mobilité. Nous avons implémenté AMiRALE au travers d'un service écrit en JAVA.

Dans cette expérimentation, chaque machine virtuelle se déplace aléatoirement dans un plan. Chaque machine virtuelle est à la fois capteur et solveur pour tous les types de missions. Les missions sont créées, verrouillées et résolues à intervalles de temps pseudo-aléatoires. Le scénario prend fin lorsque la mémoire de chaque nœud contient un nombre seuil de missions terminées. Les paramètres de notre expérimentation sont décrits dans le tableau II.

La figure 13 présente la moyenne du débit sortant des nœuds (i.e. débit des messages sortants par nœud) en fonction du nombre seuil de missions par type. Le filtre f^e_{blind} empêche la diffusion des missions terminées respectivement depuis plus de 10 secondes pour la première courbe et 60 secondes pour la deuxième courbe. Nous pouvons constater que l'évolution du

débit sortant par nœud augmente linéairement avec le nombre de missions. Nous pouvons également remarquer qu'une configuration plus fine du filtre f^e_{blind} permet de réduire le débit sortant par nœud.

La figure 14 présente la moyenne du débit entrant des nœuds (i.e. débit des messages entrants par nœud) en fonction du nombre seuil de missions par type. Les résultats sont similaires à ceux présentés en figure 13 mais avec des débits plus élevés. Pour un nombre important de missions (i.e. 240 par type, soit 1440 missions au total) le débit entrant avoisine les 160 kbit/s. Ce débit est compatible avec les principales technologies de communication utilisées pour les flottes autonomes (i.e. Bluetooth [21], ZigBee [22] et Wi-Fi [23]). Il est également important de noter que l'implémentation d'AMiRALE utilisée contient des champs d'au moins 1 octet. Une implémentation plus fine, au niveau bit, permettrait de diminuer la taille de certains champs (e.g. état de la mission, identifiant des nœuds) et donc de diminuer le débit nécessaire.

TABLE II. PARAMÈTRES D'EXPÉRIMENTATION

Paramètre	Valeur
Taille du terrain	1000 m x 1000 m
Taille des nœuds	1 m x 1 m
Vitesse des nœuds	5 m / sec
Nombre de nœuds	6
Nombre de types	6
Fréquence des messages	5 sec
Modèle de propagation	*Two-ray ground reflection*
Mémoire vive des nœuds	512 Mo
Carte réseau des nœuds	Intel e1000

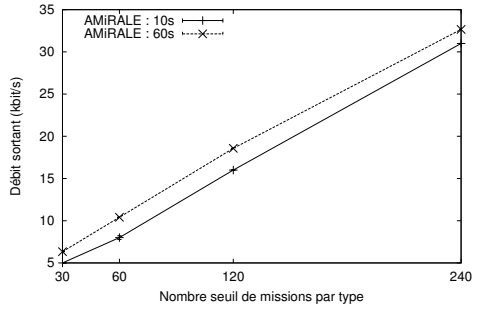

FIGURE 13. Débit sortant par nœud en fonction du nombre seuil de missions par type.

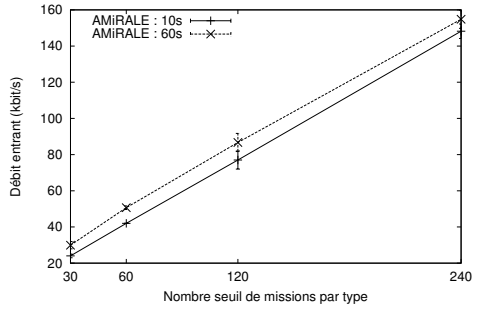

FIGURE 14. Débit entrant par nœud en fonction du nombre seuil de missions par type.

V. CONCLUSION

Nous avons présenté un nouveau système collaboratif appelé AMiRALE uniquement basé sur des opérations locales et des messages asynchrones. Nous avons détaillé les principes du modèle théorique de ce système et fourni des résultats de simulation et d'expérimentation dans un environnement virtuel. Nous avons montré que les performances de notre solution sont proches de celles d'un système collaboratif centralisé à base de mémoire partagée dans le cadre d'un scénario de nettoyage de parc. Nous avons également montré que notre solution est tolérante aux pannes à des fréquences importantes. Nous avons également mesuré les débits moyens induits par notre système. Nous avons ainsi montré que ces débits augmentent linéairement avec le nombre de missions à effectuer mais peuvent être réduits par l'intermédiaire des filtres du modèle.

Nous avons déjà évalué, sur le même scénario de nettoyage de parc, les performances des autres versions d'AMiRALE qui ne requièrent pas de synchronisation entre les horloges des nœuds de la flotte [24]. Nous avons également obtenu des résultats complémentaires en ajoutant des drones aériens à notre scénario de nettoyage de parc dans le but d'améliorer la rapidité de détection des déchets [25].

Notre prochaine étape est de réaliser un démonstrateur réel de ce scénario. Nous avons entrepris une collaboration avec la start-up Mugen [1] dans ce but.

RÉFÉRENCES

[1] NIST, "Autonomy levels for unmanned systems," Tech. Rep., October 2008.
[2] E. Ackerman, *NASA Training 'Swarmie' Robots for Space Mining*, August 2014, iEEE Spectrum Automaton.
[3] S. Chaumette, R. Laplace, C. Mazel, R. Mirault, A. Dunand, Y. Lecoutre, and J.-N. Perbet, "Carus, an operational retasking application for a swarm of autonomous uavs : First return on experience," in *MILCOM*, November 2011, pp. 2003–2010.
[4] Y. Cao, A. Fukunaga, and A. Kahng, "Cooperative mobile robotics : Antecedents and directions," *Autonomous Robots*, vol. 4, no. 1, pp. 7–27, 1997.
[5] P. Stone and M. Veloso, "Multiagent systems : A survey from a machine learning perspective," *Autonomous Robots*, vol. 8, no. 3, pp. 345–383, June 2000.
[6] L. E. Parker, "Distributed intelligence : Overview of the field and its application in multi-robot systems," *Journal of Physical Agents*, vol. 2, no. 1, pp. 5–14, 2008.
[7] D. D. Corkill, "Blackboard systems," *AI expert*, vol. 6, no. 9, pp. 40–47, 1991.
[8] C. Ingram, R. Payne, S. Perry, J. Holt, F. Hansen, and L. Couto, "Modelling patterns for systems of systems architectures," in *IEEE SysCon 8th*, March 2014, pp. 146–153.
[9] D. D. Corkill, "Design alternatives for parallel and distributed blackboard systems," Tech. Rep., 1988.
[10] V. Autefage, "Amirale formal model - a service discovery and collaboration system formalism based on dynamic graph relabeling," LaBRI - University of Bordeaux, Tech. Rep., February 2015, https://hal.archives-ouvertes.fr/hal-01114961/.
[11] R. Laplace, "Applications et services DTN pour flotte collaborative de drones," Ph.D. dissertation, Université Sciences et Technologies - Bordeaux I, December 2012.
[12] V. Autefage, A. Casteler, S. Chaumette, N. Daguisé, A. Dutartre, and T. Mehamli, "Parcs-s2 : Park cleaning swarm supervision system – a position paper," in *9th AIRTEC International Congress*, October 2014.
[13] A. Casteigts, "The jbotsim library," *CoRR*, vol. abs/1001.1435, 2010, http://jbotsim.sourceforge.net.
[14] C. Bettstetter, H. Hartenstein, and X. Pérez-Costa, "Stochastic properties of the random waypoint mobility model," *Wireless Networks*, vol. 10, no. 5, pp. 555–567, September 2004.
[15] B. Williams and T. Camp, "Comparison of broadcasting techniques for mobile ad hoc networks," in *ACM MobiHoc 3rd*, 2002, pp. 194–205.
[16] S. Ni, Y. Tseng, Y. Chen, and J. Sheu, "The broadcast storm problem in a mobile ad hoc network," in *ACM/IEEE MobiCom 5th*, 1999, pp. 151–162.
[17] V. Autefage, *Network Emulator for mobile universes (NEmu)*, http://nemu.valab.net.
[18] V. Autefage and D. Magoni, "Network emulator : A network virtualization testbed for overlay experimentations," in *IEEE CAMAD 17th*, September 2012, pp. 266–270.
[19] ——, "Virtualisation de réseau avec network emulator et application à l'évaluation d'un réseau recouvrant," in *NOTERE/CFIP*, October 2012.
[20] ——, "Virtualisation distribuée de réseaux dynamiques et mobiles avec nemu," *RNTI*, vol. SM-2, pp. 19–38, May 2013.
[21] IEEE, *802.15.1*, 2005.
[22] ——, *802.15.4*, 2011.
[23] ——, *802.11*, 2012.
[24] V. Autefage, S. Chaumette, and D. Magoni, "Comparison of time synchronization techniques in a distributed collaborative swarm system," in *IEEE EuCNC 24th*, June 2015, pp. 460–464.
[25] ——, "Influence des modèles de mobilité sur un système collaboratif pour flottes autonomes hétérogènes," in *ALGOTEL 17th*, June 2015.

1. http://mugen-sas.com