Architecture multi-agent distribuée et collaborative pour l'allocation de tâches à des senseurs : application aux systèmes navals

Paul Quentel

COLLEGES SCIENCES
BRETAGNE POUR L'INGENIEUR
LOIRE ET LE NUMERIQUE

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

THÈSE DE DOCTORAT DE

L'ÉCOLE NATIONALE SUPÉRIEURE
MINES-TÉLÉCOM ATLANTIQUE BRETAGNE
PAYS DE LA LOIRE – IMT ATLANTIQUE

ÉCOLE DOCTORALE N° 648
Sciences pour l'Ingénieur et le Numérique
Spécialité : *INFO : Informatique*

Par
Paul QUENTEL

Architecture multi-agent distribuée et collaborative pour l'allocation de tâches à des senseurs : application aux systèmes navals

Thèse présentée et soutenue à IMT Atlantique, Brest, le 30 août 2024
Unité de recherche : LabSTICC (UMR CNRS 6285)
Thèse N° : 2024IMTA0406

Rapporteurs avant soutenance :

Amal EL FALLAH SEGHROUCHNI	Professeur, Sorbonne Université
Layth SLIMAN	Professeur, EFREI

Composition du Jury :

Président :	Olivier ASSEU	Professeur, Institut National Polytechnique Félix Houphouët-Boigny
Examinateurs :	Amal EL FALLAH SEGHROUCHNI	Professeur, Sorbonne Université
	Layth SLIMAN	Professeur, EFREI
	Ludovic GRIVAULT	Ingénieur, Thales
	Jamal EL HACHEM	Maître de conférences, Université Bretagne Sud
	Pierre LE BERRE	Ingénieur architecte système, Thales
Dir. de thèse :	Yvon KERMARREC	Professeur, Institut Mines-Télécom Atlantique Bretagne Pays de la Loire

Invité(s) :

Yves Henri AUDIC	Responsable Direction Technique, Thales
Laurent SAVY	Référent Technique, Thales

Remerciements

Après trois années de recherche, de défis et de découvertes, mon parcours doctoral touche enfin à sa fin. Ce travail, bien qu'abouti, laisse encore des questions ouvertes, des pistes à explorer et des idées à approfondir, témoignant de la richesse et de la complexité du domaine que j'ai eu la chance d'explorer.

Ce chemin, ponctué de doutes, de réussites et de persévérance, n'aurait pas été possible sans le soutien, les conseils et la présence de nombreuses personnes. C'est avec une profonde gratitude que je souhaite aujourd'hui leur exprimer ma reconnaissance.

En premier lieu, je tenais particulièrement à remercier mon directeur de thèse, Yvon Kermarrec, qui a su me guider pour faire valoriser mes travaux d'un point de vue académique, et sans qui ces recherches n'auraient pas eu la même portée. Ses nombreux retours sur mes présentations et mon manuscrit ont grandement contribué à améliorer la qualité et la compréhension de mes recherches. De plus, nos nombreux débats constructifs ont non seulement enrichi mon travail, mais aussi stimulé ma réflexion critique, me poussant à explorer de nouvelles perspectives et à approfondir ma démarche scientifique. Je le remercie très chaleureusement.

J'adresse également toute ma reconnaissance à mes encadrants, Pierre Le Berre et Ludovic Grivault. Leur rigueur scientifique a a joué un rôle crucial dans l'avancement de mes recherches, en particulier sur les aspects opérationnels. Leur expertise technique a été inestimable, mais plus encore, leur soutien constant, leur encouragement et leur bienveillance m'ont permis de progresser sereinement tout au long de cette thèse et d'affronter la complexité des systèmes logiciels et de simulation. Ils ont su, à chaque étape, être des piliers de confiance, m'aidant non seulement à surmonter les défis intellectuels, mais également à maintenir une motivation et un équilibre indispensables à la réussite de ce travail.

Je tiens également à remercier sincèrement les membres de mon jury. Je remercie mes deux rapporteurs, Amal El Fallah-Seghrouchni ainsi que Layth Sliman pour la lecture et l'évaluation de mes travaux. Mention spéciale pour Oliver Asseu, qui a lu avec intérêt mes travaux, m'a fait des suggestions et qui a présidé cette soutenance avec sa bonne

humeur et sa pointe d'humour, merci. Je remercie également Jamal El Hachem pour son implication en tant que membre du jury et son intérêt porté sur le sujet.

Cette thèse, réalisée en collaboration avec Thales et l'IMT Atlantique, n'aurait pu voir le jour sans le soutien et l'accompagnement des nombreux collègues, membres du personnel, et doctorants que j'ai eu la chance de rencontrer au cours de ces dernières années. Je tiens à exprimer ma gratitude à Louis et Kevin, mes amis doctorants CIFRE, dont la présence a égayé nos pauses et repas en entreprise. Un grand merci également à Gwenolé, mon voisin de bureau, pour sa compagnie et son soutien quotidien. Je souhaite également remercier chaleureusement Julien, Amine, Aymeric, Stanislas, Vincent, Nicolas et Coraline, pour les échanges enrichissants que nous avons eus et pour les moments de convivialité partagés lors de nos sorties. Ces instants de détente ont été non seulement agréables mais aussi essentiels pour maintenir l'équilibre nécessaire au cours d'une thèse. Si par mégarde j'ai omis certains noms, je tiens à m'excuser sincèrement...

Je souhaiterais également exprimer ma profonde gratitude à Yves Henri Audic et Laurent Savy, sans qui cette thèse n'aurait jamais vu le jour. Leur encadrement au cours de la première année a permis d'orienter les travaux de recherche. Yves, je te remercie sincèrement pour la confiance que tu m'as accordée depuis nos premiers échanges en 2018, lors de mon stage d'ingénieur. Tu as activé de nombreux leviers pour me permettre d'arriver là où je suis aujourd'hui, et pour cela, je te suis profondément reconnaissant.

Pour conclure, je tiens à remercier chaleureusement ma mère, mon père et mon frère pour leur soutien et leur présence durant la soutenance. Je souhaite également exprimer toute ma reconnaissance à ma compagne, Lavinia, pour son soutien indéfectible, sa patience et sa compréhension tout au long de cette aventure. Je la remercie de tout mon cœur pour ses encouragements, ainsi que pour son aide précieuse lors de la relecture de mes travaux et sa participation dans la préparation de mes présentations orales.

Définitions

Architecture orientée services L'architecture orientée services (OSA, SOA) est un style d'architecture qui met à disposition des ressources disponibles de telle sorte qu'elles puissent être trouvées et utilisées par d'autres parties qui n'ont pas besoin de les connaître en avance.

Brouillage Le principe du brouillage est d'émettre intentionnellement dans le but de saturer un récepteur avec du bruit ou des données factices. Il existe des brouillages mécaniques qui comprennent les paillettes, réflecteurs radar et les leurres, et des brouillages électroniques comme le balayage ou le barrage.

Interfaces Les interfaces sont des connecteurs qui facilitent les échanges entre des composants de systèmes hétérogènes sans nécessiter une compréhension détaillée du fonctionnement de chaque composant. Un exemple d'interface bien connu est l'IHM (Interface Homme-Machine), qui permet aux utilisateurs d'interagir avec des logiciels ou des systèmes malgré leur complexité interne.

Métriques Les métriques d'un système désignent les données pertinentes que nous surveillons pour garantir son bon fonctionnement. Elles fournissent des indicateurs quantitatifs permettant d'évaluer les performances, la disponibilité, la fiabilité et d'autres aspects du système. Dans le cadre de RabbitMQ, ces métriques peuvent inclure des indicateurs spécifiques tels que le nombre de *sockets* ouverts, le nombre de messages présents dans une file d'attente, ainsi que les taux d'utilisation des ressources système tels que le CPU, la mémoire et le stockage. La latence et le débit moyen permettent d'évaluer la performance globale d'un système d'un point de vue du réseau.

Middleware Un middleware est une couche logicielle positionnée au-dessus du système d'exploitation mais en dessous du programme d'application. Il offre une abstraction de programmation commune à travers un système distribué [Bak01]. Les middlewares gèrent la complexité et l'hétérogéneité inhérentes aux systèmes distribués. Les MOM (Message Oriented Middlewares) sont des exemples de middlewares qui facilitent la communication entre des applications distribuées en utilisant un modèle de messagerie asynchrone.

Monitoring Le *monitoring* consiste à capturer le comportement d'un système au cours du temps en collectant des indicateurs de performances, aussi appelés métriques. Cela permet de détecter et d'identifier des anomalies telles que des congestions dans le réseau ou une répartition inégale de la charge sur les processeurs.

Plan senseur Un plan senseur P_k [Gri18] est composé d'un ensemble de tâches T_i, avec des contraintes les liant entre elles, affectées à une ou plusieurs ressources.

Plateforme Les plateformes sont des entités militaires comme des bâtiments de surface ou des aéronefs. Une plateforme se caractérise par sa position géographique (longitude, latitude et altitude), sa vitesse, ses communications, et les senseurs qu'elle possède définissant ainsi ses capacités d'observation.

Radar de poursuite Un radar de poursuite est un radar qui poursuit une cible grâce à l'estimation des trajectoires en fonction de la position de la cible et de son déplacement. Le radar de poursuite peut être continue mais aussi discontinue, dans ce cas le radar peut poursuivre une multitude de cibles en même temps.

Radar de conduite de tir C'est un radar conçu spécialement pour récupérer les informations de vitesse, de distance, d'azimuth et d'altitude d'une cible dans le but de calculer une solution de tir.

Requête opérationnelle Une requête opérationnelle est issue d'un ordre de mission interne ou externe à la plateforme. Cette requête décrit les besoins durant une opération, elle comprend notamment : le service opérationnel, la zone géographique d'application, la durée effective de la requête, les contraintes, une priorité et la qualité de service désirée.

Senseur Un senseur est un instrument qui détecte et répond à certaines entrées provenant d'un environnement à l'aide de capteurs et de traitements intégrés [TE20]. Dans nos travaux, nous utilisons un modèle simplifié du senseur, que nous caractérisons principalement par sa portée de détection, son type et les services qu'il propose.

Service senseur Un service senseur fournit une capacité (ou une fonction) que proposent un ou plusieurs senseurs. Les senseurs déclarent les services qu'ils peuvent accomplir dans un registre qui sera ensuite partagé entre des utilisateurs de services. Le service senseur est caractérisé par une qualité de service (QoS), des règles, et des ressources qui permettent son utilisation. Il répond à une requête de l'utilisateur sans que celui-ci n'ait besoin de connaître son fonctionnement interne.

Système distribué Un système distribué est un système dans lequel les composants,

situés sur des entités en réseau, communiquent et coordonnent leurs actions en échangeant des messages [CDKB05].

Tâche Une tâche correspond à une unité de travail qui doit souvent être terminée dans un certain délai [TE20]. Dans notre contexte, une tâche correspond à la réservation d'une ressource pendant une période donnée. Une tâche peut avoir des contraintes de précédence si une autre tâche doit être effectuée avant celle-ci. Une tâche peut se retrouver en concurrence avec une autre tâche lorsque la même ressource est demandée simultanément. Dans notre contexte, un service peut nécessiter une ou plusieurs tâches pour fonctionner.

Acronymes

ABMS *Advanced Battle Management System*

AIS *Automatic Identification System*

AMQP *Advanced Message Queuing Protocol*

C2 *Command and Control*

CCN Combat collaboratif naval

CEC *Cooperative Engagement Capability*

CESM *Communication Electronic Support Measure*

CESMO *Cooperation Electronic Support Measures Operations*

CME Contre-mesures électroniques

CMS *Combat Management System*

DGA Direction générale de l'armement

DRIL-P Détection, reconnaissance, identification, localisation, poursuite

EW *Electronic Warfare*

FACE *Future Airbornes Capability Environment*

FDA Frégate de défense aérienne

FDI Frégate de défense et d'intervention

FREMM Frégates multi-missions

GAE Groupe aérien embarqué

GAN Groupe aéronaval

GE Guerre électronique

GPS *Global Position System*

IA Intelligence artificielle

IRST *Infra-Red Search and Track*

JADC2 *Joint All Domain Command and Control*

LDT Liaison de données tactiques

MOM *Message Oriented Middleware*

MOSA *Modular Open System Architecture*

MPE Moyens de protection électronique

MRE Moyen de renseignement electronique

OMS *Open Mission System*

OODA *Observe, Orient, Decide and Act*

OTAN Organisation du traité de l'Atlantique nord

PHA Porte-hélicoptères amphibie

RESM *Radar Electronic Support Measure*

SAR *Synthetic Aperture Radar*

SER Surface équivalente radar

SMA Système multi-agent

SOSA *Sensor Open System Architecture*

TDOA *Time Difference of Arrival*

UHF *Ultra High Frequency*

USAF *United State Air Force*

VCAM Veille coopérative aéro-maritime

VCN Veille coopérative navale

VHF *Very High Frequency*

TABLE DES MATIÈRES

INTRODUCTION

Problématique générale

L'évolution du contexte de défense aéronaval, et de ses besoins en nouvelles fonctionnalités dans le cadre du renforcement des opérations interarmées de la Marine, nécessite une modification majeure de l'architecture des systèmes de senseurs actuels afin de maitriser les futures menaces [QKG+23]. Depuis plus d'une dizaine d'années, la Direction générale de l'armement (DGA) a amorcé des recherches et financé des programmes dans le but d'améliorer les capacités opérationnelles des systèmes de combat des bâtiments de surface. De nos jours, les systèmes de senseurs coopèrent et partagent leurs systèmes d'information avec les autres plateformes par le biais du système de gestion de combat ou *Combat Management System* (CMS). Le CMS de chaque plateforme permet la gestion des senseurs locaux ainsi que le suivi des pistes et des objets de la zone à surveiller, et ce avec autonomie. Les senseurs ne s'associent pas entre eux sur une même plateforme ou avec ceux des plateformes alliées. Chaque plateforme maintient une situation tactique locale à l'aide de ses senseurs, ainsi qu'une situation tactique globale grâce à l'échange d'information avec les autres plateformes par le biais du CMS et des liaisons de données tactiques (LDT). En 2021, la veille coopérative navale (VCN) a fait ses preuves en mer [Gro19], elle permet d'obtenir une situation tactique plus élaborée grâce aux échanges et la fusion instantanée des informations brutes des radars de la flotte.

Depuis quelques années, les senseurs tendent à devenir des systèmes complexes [SG20], capables de partager de la donnée, de communiquer et, bientôt, de collaborer. La collaboration est une notion fondamentale, elle permet aux senseurs d'offrir de nouveaux modes et services au CMS. Dans le cadre de nos recherches, nous nous intéressons particulièrement aux senseurs de Guerre électronique (GE), aux radars et aux capteurs optroniques, nous les détaillerons dans une partie du manuscrit. Le travail collaboratif se définit par une intelligence collective fédérée autour de services opérationnels et reconfigurables en fonction des objectifs à mener. Dans un système collaboratif, plusieurs plateformes peuvent contribuer à la réalisation d'une tâche. La différence avec la coopération réside dans le degré d'interaction et la nature des échanges entre les entités impliquées. Alors que la

coopération implique souvent des entités travaillant ensemble vers un objectif commun avec des interactions limitées et sans nécessairement partager leurs ressources, la collaboration suppose une coordination plus étroite des ressources avec des échanges de données fréquents.

Les nouvelles architectures devront permettre la collaboration des différents senseurs en réseau, avec un besoin crucial d'obtenir et d'exploiter l'information sur l'ennemi plus rapidement afin de prendre l'initiative. Les échanges de données amélioreront la connaissance globale des systèmes, permettant ainsi un meilleur traitement des menaces. De nombreuses problématiques architecturales apparaissent, telles que la mise en réseau des senseurs, les mécaniques de gestion des senseurs, l'optimisation des ressources, l'amélioration des traitements sur la donnée pour la rendre plus rapidement exploitable, l'augmentation du débit des communications, etc.

Objectifs

Ces travaux de thèse présentent les démarches et les choix qui ont permis l'élaboration d'une architecture distribuée et collaborative, avec un besoin applicatif : l'allocation de tâches à des senseurs dans le combat collaboratif naval (CCN). L'architecture proposée doit répondre aux exigences technico-opérationnelles concernant la détection, la localisation et le pistage de menaces, tout en réduisant le temps de prise de décision (cf. la boucle OODA présentée en section 1.6.1). De plus, le contexte opérationnel limite l'utilisation de la bande passante. Cette limitation contraint les choix d'architectures possibles, notamment sur les composants qui pourront être amenés à communiquer et qui devront transmettre une quantité de données limitée. La recherche sur les systèmes multi-agents (SMA) met en avant les agents intelligents comme la métaphore naturelle pour résoudre des problèmes distribués et complexes [BOMJ19]. Dans cette perspective, les agents, dotés de capacités autonomes et de règles de coordination, peuvent interagir et s'adapter de manière décentralisée pour atteindre des objectifs communs. En exploitant les différentes propriétés des agents, l'approche multi-agent facilite la résolution de problèmes d'allocations dynamiques de tâches à des senseurs en réseau dans un contexte de collaboration multi-plateforme, multi-milieu et multi-senseur. L'autonomie du système est améliorée et sa complexité de fonctionnement est réduite du point de vue de l'utilisateur, qui voit diminuer sa charge de travail. Pour concevoir cette architecture, nous devons répondre aux questions de recherche suivantes :

QR1 : Comment les systèmes multi-agents peuvent-ils répondre de manière efficace et évolutive aux défis d'allocation de tâches à des senseurs dans le contexte opérationnel du combat collaboratif, en tenant compte des contraintes spécifiques liées à la collaboration entre les plateformes militaires ou civiles et leurs senseurs, tout en assurant la capacité à passer à l'échelle pour gérer un nombre croissant de senseurs et de tâches ? Comment la conception et la mise en œuvre des agents peuvent-elles répondre à la complexité croissante des environnements militaires, en tenant compte des priorités sur les tâches, des possibles pertes matérielles ou des changements dynamiques de disponibilités des ressources ?

QR2 : Quelles stratégies de communication et d'échange d'informations entre les plateformes peuvent être intégrées dans une architecture distribuée pour garantir une coordination efficace des senseurs, en particulier lorsque les canaux de communication sont limités et que la quantité d'informations échangées est critique ? Comment ces stratégies peuvent-elles prendre en compte les contraintes de bande passante et de latence, ainsi que la nécessité de prioriser les données en fonction de leur pertinence opérationnelle et de leur criticité ?

QR3 : Comment les systèmes multi-agents peuvent-ils être évalués et testés dans des environnements simulés ou réels pour valider leur efficacité et leur performance dans des scénarios opérationnels variés, en utilisant des métriques telles que les latences, la bande passante utilisée et la capacité à répondre aux exigences opérationnelles spécifiques ?

La figure 1 synthétise les éléments de réponses à nos questions de recherche. Nous proposons une nouvelle architecture de collaboration multi-senseur et multi-plateforme. L'aspect orienté services permet de masquer le fonctionnement des senseurs à l'utilisateur du système, ici l'opérateur, qui pourra se concentrer sur les services proposés et les interfaces du système. Une proposition de service pourra se traduire par une demande d'allocation de tâches à des senseurs (section 1.6.2), et le SMA sera investi de la gestion des ressources pour accomplir au mieux les objectifs du système (section 3.4). Ensuite, des scénarios opérationnels soulèveront des métriques d'intérêt qui permettront d'évaluer la nouvelle architecture (section 3.2).

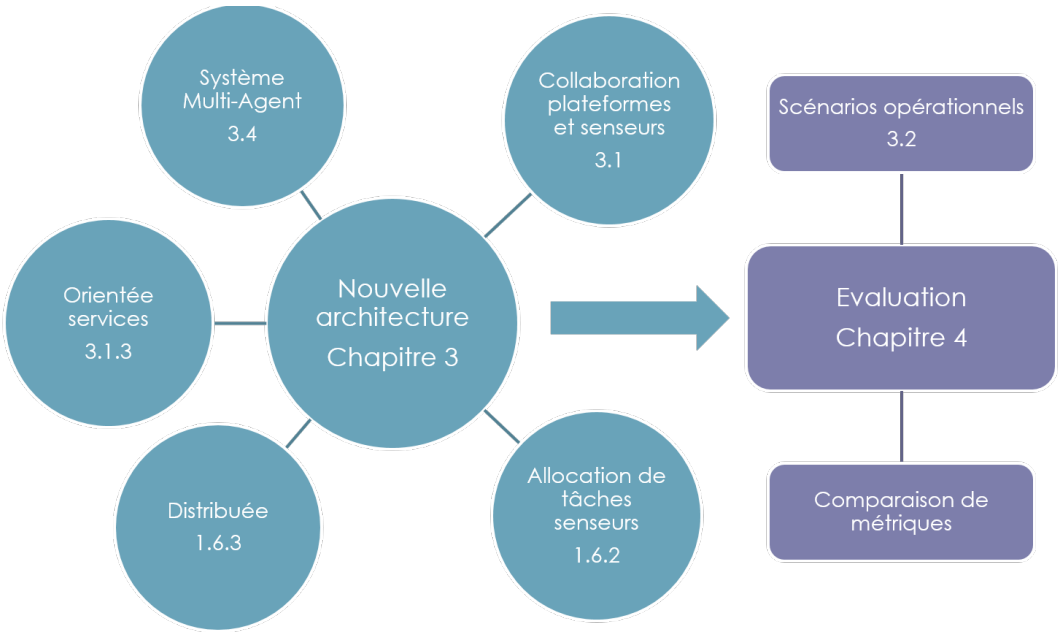

FIGURE 1 – Éléments de réponse aux questions de recherche.

Organisation du manuscrit de thèse

Le premier chapitre du mémoire permettra d'appréhender le sujet. Nous exposerons le contexte d'application de nos travaux dans le cadre du combat collaboratif naval (CCN). Nous définirons les plateformes, les senseurs et leurs utilisations dans un théâtre d'opérations. Nous préciserons les spécificités et contraintes du milieu naval ainsi que les menaces actuelles et futures, cadre d'évolution des systèmes développés. Dans ce contexte, nous présenterons les enjeux qui se posent à nous, à savoir : la conception d'une architecture répondant à des besoins opérationnels et à des exigences industrielles, la spécification de l'allocation de tâches dans le cadre du contexte décrit, et les challenges des systèmes distribués avec les problématiques orientées réseau.

Le deuxième chapitre présentera un état de l'art sur les recherches concernant les grands thèmes abordés lors de cette thèse. Nous regarderons les travaux d'architectures existants qui cherchent à faire collaborer différents systèmes militaires afin d'accomplir un objectif commun. Nous proposerons ensuite de faire un point sur l'utilisation des systèmes multi-agents (SMA), puis nous nous focaliserons sur l'utilisation des SMA pour de l'allocation de tâches en décrivant les différentes approches, plus particulièrement les méthodes à base d'enchères.

Dans un troisième chapitre, nous détaillerons les travaux de conceptions d'architecture. Dans un premier temps, nous présenterons les premiers concepts d'architecture de combat collaboratif naval. Puis, nous définirons un scénario opérationnel afin de préciser les besoins et les nouvelles fonctionnalités auxquels doivent répondre les systèmes. Nous préciserons l'intérêt des systèmes multi-agents pour répondre à la problématique et nous présenterons les réflexions sur l'agentification. Après avoir déterminé les différents rôles nécessaires dans notre SMA, nous présenterons chaque type d'agent en charge d'un rôle. En particulier, nous décrirons les algorithmes et les interfaces de communications des agents, en précisant les enjeux des systèmes distribués de la solution proposée. Enfin, nous évaluerons la complexité de ces algorithmes et interfaces en fonction des paramètres du système, ceux-ci auront un impact sur le passage à l'échelle de la solution.

Le quatrième chapitre mettra en lumière les expérimentations réalisées ainsi que les résultats et les évaluations de l'architecture. Nous avons pour cela observé le fonctionnement d'un système existant en exploitant un simulateur opérationnel interfacé à un logiciel propriétaire Thales. Nous avons ensuite conclu que le système actuel était difficilement adaptable pour la conception d'une architecture distribuée, et que les plateformes multi-agents existantes ne remplissaient pas les besoins définis en phase d'étude. Pour ces différentes raisons, nous avons développé un banc d'essai pour simuler des échanges entre des agents de manière asynchrone et analyser les performances. Plusieurs versions de ce banc d'essai ont été réalisées dans le but de démontrer la validité des concepts de l'architecture multi-agent proposée en chapitre trois. Ces expérimentations nous permettront d'évaluer l'architecture et les mécanismes répondants aux différents besoins susmentionnés.

Enfin, nous conclurons ce manuscrit en résumant la proposition d'architecture permettant l'allocation de tâches à des senseurs distribués et en répondant aux différentes questions de recherche adressées. Nous ouvrirons sur des perspectives pour la suite de ces travaux.

Chapitre 1

Contexte applicatif et industriel

Comme précisé dans l'introduction, le contexte de défense aéronaval nécessite de concevoir une nouvelle architecture de collaboration entre les senseurs de différentes plateformes. Dans ce premier chapitre, les plateformes aéroportées et navales ainsi que les types d'instruments embarqués à leur bord seront présentés et définis. L'objectif de la collaboration entre ces éléments hétérogènes consiste à collecter et partager de l'information sur les acteurs (avions, navires, civils ou militaires, alliés ou ennemis) présents dans l'environnement que nous appelons théâtre d'opérations. Cela nécessite des actions collaboratives impliquant l'utilisation de senseurs complémentaires. Des exemples de plateformes et de senseurs sont présentés dans ce premier chapitre et en annexe.

Definition 1 (Plateforme) Les plateformes sont des entités militaires comme des bâtiments de surface ou des aéronefs. Une plateforme se caractérise par sa position géographique (longitude, latitude et altitude), sa vitesse, ses communications, et les senseurs qu'elle possède définissant ainsi ses capacités d'observation.

1.1 Présentation des plateformes

Le contexte de défense aéronaval requiert l'utilisation conjointe de plateformes aériennes et de surfaces [Nata]. En effet, lors de mission en mer, un ou plusieurs bâtiments de surface sont déployés, et accompagnés d'aéronefs. De plus, certaines plateformes navales contiennent : des avions pour le porte-avions Charles de Gaulle, des hélicoptères pour les porte-hélicoptères amphibies (PHA) ou les Frégates multi-missions (FREMM), et des drones pour certaines frégates. Le ministère des armées présente l'ensemble des bâtiments de la Marine nationale et des porteurs de l'aéronautique navale [ble21]. Dans les sous-parties suivantes, nous détaillerons une plateforme navale et une plateforme aérienne. L'étude actuelle ne porte pas sur la partie sous-marine. Nous présentons des plateformes en complément dans l'annexe. L'utilisation conjointe de plateformes autour d'un porte-avions s'appelle le Groupe aéronaval (GAN).

1.1.1 Une plateforme navale : Le porte-avions Charles de Gaulle

Le porte-avions Charles de Gaulle (figure 1.1) est un bâtiment à propulsion nucléaire de la Marine nationale française, c'est d'ailleurs la pièce maîtresse de la flotte. Parmi ses missions principales, nous pouvons citer la maîtrise de l'espace aéromaritime et l'appui aérien aux opérations à terre. Le porte-avions opère généralement au sein d'un Groupe aéronaval (GAN) qui comprend : le Groupe aérien embarqué (GAE), une frégate anti-sous-marine, une Frégate de défense aérienne (FDA), un sous-marin nucléaire d'attaque et un bâtiment de ravitaillement.

Le GAN est un vecteur de coopération militaire, il est capable d'intégrer des navires étrangers, ce fut notamment le cas avec le destroyer américain USS Ross qui a assuré la défense aérienne du porte-avions lors de la Task Force 473 [ble20]. Le GAN peut contribuer à plusieurs objectifs de missions en simultanés grâce à une combinaison de capteurs offrant des capacités diverses et variées.

FIGURE 1.1 – Porte-avions Charles de Gaulle (2019).

Le porte-avions communique à l'aide de transmissions satellites, de liaisons de données tactiques ou de liaisons radio permettant entre autres le partage de pistes radar [Natb]. Une quarantaine d'aéronefs compose le groupe aérien embarqué : une trentaine de Rafales marines, deux Hawkeyes et des hélicoptères (deux Caïmans Marine, deux Panthers et deux Dauphins Pedro), chacun permettant d'obtenir une vue du théâtre. Les aéronefs ont chacun des rôles spécifiques et définis durant les missions. Le Hawkeye est chargé de la

veille lointaine, du commandement et du contrôle. Le Dauphin Pedro est garant de la sécurité d'appontage et de décollage des Rafales sur le porte-avions, il surveille également l'espace rapproché. Les plateformes sont complémentaires les unes des autres, le Caïman Marine peut par exemple guider une frappe du Rafale augmentant ainsi sa précision.

Le porte-avions possède plus d'une dizaine de systèmes de senseurs [Jan24], capables de percevoir l'environnement, de suivre des menaces et de protéger le batiment des radars ennemis :

- Un ensemble de radars tels que le ARABEL ou le SMART-S Mk2. Le premier aide à la conduite de tir du système, tandis que le second assure les missions de veilles aérienne et surface moyenne portée.
- Des systèmes de guerre électronique tels que le détecteur radar pour les mesures de soutien électronique RESM ARBR-21. Il s'agit d'un système passif d'écoute du spectre électromagnétique dans les bandes radars assurant les fonctions d'alerte, de tenue de situation tactique et de renseignement.
- Plusieurs systèmes optroniques comme par exemple le PASEO XLR qui est un système d'observation à très longue portée, capable d'observer de jour comme de nuit et dans des conditions climatiques diverses.

En complément, chaque senseur possède des modes (ou profils) de fonctionnement différents, par exemple le SMART-S peut activer un mode surveillance ou un mode défense. Il s'interface avec le Système de Combat et intègre des consoles de visualisation et de maintenance. Nous pouvons donc remarquer que chaque plateforme, à l'instar du porte-avions, embarque de multiples senseurs avec des fonctions spécifiques, ce qui démontre l'hétérogénéité des systèmes et les possibilités d'actions collaboratives, comme par exemple un suivi multi-cible qui combine les informations des différents capteurs. D'autres plateformes aériennes et navales, présentées en annexe, possèdent des rôles précis dans le GAN. La variété des senseurs et leurs conditions d'utilisation sur chaque plateforme améliorent le succès des missions.

1.1.2 Une plateforme aérienne : Le Rafale Marine

FIGURE 1.2 – Rafale Marine.

Le Rafale Marine est un avion de combat français polyvalent capable d'accomplir diverses missions, telles que l'attaque en mer ou la défense aérienne, pour n'en citer que quelques-unes [Natc]. Il est embarqué sur le porte-avions et possède un armement diversifié ainsi que de nombreux équipements électroniques et optroniques. Nous retiendrons par exemple l'équipement du système GE de contre-mesures électroniques (CME) Spectra, le radar RBE-2 ou le capteur optronique OSF. La diversité des dispositifs du Rafale Marine, couplée à sa haute vélocité, permet au GAN d'intervenir rapidement sur les zones de conflits et de sécuriser l'espace aérien, malgré les contraintes et les limitations inhérentes aux systèmes embarqués et aux communications actuelles. Ces contraintes incluent la disponibilité limitée d'espace et de poids pour l'intégration des équipements, ainsi que les défis liés à la bande passante restreinte, à la latence et à l'interopérabilité des communications dans des environnements opérationnels complexes.

1.2 Présentation des senseurs

Les plateformes exposées précédemment sont dotées de multiples senseurs. Ceux-ci peuvent, comme leur nom l'indique, percevoir l'environnement en recueillant des informations électroniques et permettent aux plateformes de naviguer et d'agir lors de détection de menaces. Dans un premier temps, il conviendra de définir ce qu'est un senseur, ensuite, nous présenterons un exemple de senseur embarqué sur le Rafale. D'autres modèles de senseurs sont également décrits en annexe. Étant donné le caractère confidentiel des senseurs mentionnés, ces informations proviennent uniquement de la littérature ouverte et de sources en ligne.

1.2.1 Définition d'un senseur

L'étymologie du mot provient de l'anglais « sensor » qui signifie « capteur » en français. Selon le dictionnaire Larousse, un senseur correspond à « Tout équipement de détection inclus dans un système d'arme (radars, caméras infrarouges, télémètres lasers). » De manière générale, c'est un appareil qui transforme une grandeur physique en un signal, le plus souvent électrique, ce signal est ensuite exploité après un traitement. Un capteur possède un sens commun plus simple que ce qu'on nomme senseur. Un senseur combine un ensemble de capteurs et de traitements intégrés. Dans le cas des senseurs embarqués dans les plateformes, le senseur recueille l'information électromagnétique qu'il a sur les éléments de l'environnement. Ces senseurs sont composés pour la plupart d'émetteurs et/ou de récepteurs ainsi que d'une partie servant au traitement de l'information brute récupérée.

Definition 2 (Senseur) Un senseur est un instrument qui détecte et répond à certaines entrées provenant d'un environnement à l'aide de capteurs et de traitements intégrés [TE20]. Dans nos travaux, nous utilisons un modèle simplifié du senseur, que nous caractérisons principalement par sa portée de détection, son type et les services qu'il propose.

Deux groupes de senseurs se distinguent : les senseurs actifs et les senseurs passifs. Les senseurs actifs émettent des signaux afin de récupérer de l'information par la suite. Par exemple, un système radar génère une onde électromagnétique puis attend un retour du même signal atténué. Les senseurs passifs sont plus furtifs, car ils n'émettent pas, ils ne font que capter l'information provenant de l'environnement. Un senseur de guerre

électronique (GE) est un senseur passif qui cherche à recueillir un maximum de données pour compléter sa situation tactique, et ceci sans être découvert.

Nous considérons trois types de senseurs différents, proposant des capacités propres. Les senseurs de GE comme le RESM (*Radar Electronic Support Measure*) peuvent détecter des signaux radar en restant discrets, puis localiser et identifier les cibles correspondantes aux signatures. Les radars comme les FCR (*Fire Control Radar*) sont des senseurs actifs capables de détecter, de localiser et de classifier des cibles aériennes ou de surfaces en fournissant des données de distance, de direction et de vélocité [KNF10]. Enfin, les senseurs optroniques accomplissent de la veille infrarouge (ou IRST : *Infra-Red Search and Track*) qui consiste à repérer et localiser des menaces par leur chaleur. La complémentarité des senseurs leur permet de compléter et de confronter les renseignements concernant une cible. Ainsi, le radar fournit la position et la mesure de distance avec la cible. La GE produit des droites de visée et collecte des informations de distance par défilement (déplacement de la plateforme réceptrice). Enfin, l'optronique ne mesure pas la distance, mais peut recueillir les informations de position et d'identification.

La fusion et le partage des données provenant de tous les senseurs permettent de construire ce qu'on appelle la situation tactique. Cette dernière correspond à l'état du théâtre des opérations du point de vue d'une plateforme, avec la connaissance sur les éléments de l'environnement. Elle peut être représentée par une carte de la zone d'opération avec une identification des éléments singuliers (amis, neutres, ennemis ou autres) et les données connues sur chacun comme sa vitesse, sa position ou son type. La situation tactique peut être enrichie en échangeant des données entre les plateformes et en déduisant de nouvelles caractéristiques avec les données connues. Par exemple, la vitesse d'une cible peut être calculée grâce aux mesures de sa position à des instants différents, sa direction peut aussi être déterminée.

1.2.2 La guerre électronique

La guerre électronique (GE) cherche à préserver, au profit de son porteur, l'intégrité du spectre, pour y conduire les actions qui concourent à la préparation, à l'exécution et à l'évaluation des opérations navales, terrestres ou aériennes. Elle vise aussi à dégrader, à l'insu ou non de l'adversaire, ce même spectre, pour le contraindre à n'utiliser ses capacités qu'en modes altérés. La GE est divisée en trois branches [Spe02] :

➤ L'attaque électronique, ou la contre-mesure électronique (CME), comprend les opé-

rations de brouillage, de leurrage et de déception dans le but d'agir contre les menaces et d'interdire l'utilisation du spectre aux adversaires.

- ➤ Le soutien électronique, ou moyen de renseignement électronique (MRE), vise à contrôler le spectre électromagnétique en utilisant les émissions électroniques de l'adversaire afin de détecter, localiser et identifier les cibles.
- ➤ La protection électronique, également connue sous le nom de moyens de protection électronique (MPE), vise à assurer la sécurité des plateformes alliées. Par exemple, elle peut réduire la surface équivalente radar (SER) lors de la conception des porteurs, mettre en place des mesures de silence radio, recourir à des technologies de saut de fréquence ou utiliser des moyens de communication sécurisés par chiffrement. Elle est également désignée sous le terme de CCME, la contre contre-mesures électroniques.

Dans le cadre de nos travaux, nous nous intéressons plus particulièrement à la seconde branche, celle du soutien électronique. Les systèmes MRE, ou plus communément appelés ESM (Electronic Support Measures), jouent un rôle crucial dans la guerre électronique en offrant une capacité de surveillance du spectre électromagnétique. Leurs forces résident dans leur capacité à détecter, localiser et identifier les émissions électromagnétiques adverses, ce qui permet aux forces alliées de mieux comprendre les intentions et les mouvements de l'ennemi. Cependant, les ESM présentent également des faiblesses, notamment en termes de capacité à discriminer les signaux et à distinguer les émissions adverses des signaux parasites ou non pertinents.

Pour pallier ces limitations, la collaboration entre les systèmes ESM et d'autres senseurs s'avère essentielle. Par exemple, les radars fournissent une vue plus précise de l'environnement en détectant les cibles de manière active, tandis que les ESM offrent une compréhension plus large du spectre électromagnétique en détectant les émissions de manière passive. Ensemble, ces systèmes permettent une surveillance plus efficace contre les menaces de l'environnement et renforcent ainsi les capacités opérationnelles des forces armées lors de missions.

1.2.3 Fonctionnement d'un radar

Comme mentionné précédemment, le radar est un senseur actif. Il émet une onde électromagnétique dans une direction donnée (aussi appelé impulsion radar) qui se déplace à une vitesse proche de celle de la lumière. Un objet se situant dans la direction d'émission

de l'onde radar réfléchira celle-ci permettant au système de réception du radar d'obtenir un signal retour bruité et atténué. L'objet détecté possède des propriétés de réflexions spécifiques (SER) qui dépendent de sa forme, de sa composition et de la nature de ses matériaux. De plus, le radar est composé, entre autres, d'une antenne et de circuits électroniques intégrés qui impactent les propriétés de puissance et de directivité de celui-ci.

Le radar utilise l'écho qu'il reçoit pour déterminer la distance et la direction de l'objet détecté. La précision, la résolution et l'ambiguïté sur les mesures effectuées dépendent du radar, de sa puissance en émission, de la directivité et de la taille de son antenne. Selon les besoins de la mission, un mode particulier de l'un des radars embarqués pourra être sélectionné en fonction du rôle de celui-ci. Par exemple, un mode peut permettre d'obtenir plus de précisions sur la position d'une cible et de réduire l'ambiguïté, ou lorsque la cible ne peut être localisée que par un radar de longue portée.

Le système collaboratif vise à optimiser la sélection des équipements en fonction des exigences spécifiques de la situation rencontrée. Cette approche permet de tirer parti des forces et des faiblesses respectives des différents radars et des systèmes de guerre électronique. Certains radars peuvent offrir une portée étendue avec une bonne capacité de détection, mais peuvent être plus vulnérables aux brouillages électromagnétiques. D'autres radars, bien que moins sensibles aux brouillages, pourraient avoir une portée plus limitée. En intégrant les capacités des radars avec celles des systèmes de guerre électronique, il devient possible de compenser les faiblesses de chaque système et de renforcer la connaissance globale des objets du théâtre des opérations. En général, les récepteurs de guerre électronique possèdent une portée de détection plus importante que les radars. Le signal d'un radar doit faire un aller-retour, tandis que le récepteur de GE attend un aller simple de celui-ci. La GE détecte également plus rapidement les cibles ennemies, et ce de manière passive.

1.2.4 Les systèmes optroniques

L'optronique peut être utilisée pour accomplir différentes fonctions dans le domaine militaire [Mey15]. Par exemple, l'autodirecteur d'un missile est équipé d'un pointage laser qui lui permet de suivre sa cible et de guider le missile de façon autonome. En fonction du domaine de fréquence, les autodirecteurs peuvent recourir à des techniques télévision, infrarouge ou laser.

Dans le cadre de nos travaux, les senseurs optroniques sont principalement déployés pour faire de la veille infrarouge ou IRST (*Infra-Red Search and Track*) qui consiste à

repérer et localiser des menaces par leur chaleur. Le système optronique de veille IR n'utilise que le domaine visible des infrarouges en observant les émissions de chaleurs, ce qui le rend indétectable par l'ennemi en comparaison du radar. L'utilisation d'une faible longueur d'onde rend le senseur plus précis et moins encombrant, ce qui permet une meilleure reconnaissance des cibles. Le senseur optronique possède une portée de fonctionnement moins importante que le radar, et il ne permet pas d'obtenir d'information sur la vitesse de la cible. Cela confirme l'intérêt de l'emploi conjoint de l'optronique avec le radar dans le but de compléter et d'améliorer les informations de pistages.

1.2.5 Exemple d'un senseur : le radar RBE2 AESA

FIGURE 1.3 – RBE2 AESA.

Le radar RBE2 [Tha] (radar à balayage électronique 2 plans, figure 1.3) équipe le Rafale. La troisième version du radar RBE2 utilise une antenne active AESA (Active Electronically Scanned Array) composée de plusieurs milliers de modules de transmission/réception appelés TRM. Cette évolution a de nombreux avantages en comparaison à son prédécesseur passif PESA (*Passive Electronically Scanned Array*), notamment la flexibilité grâce à la possibilité de diviser les modules afin d'accomplir de multiples tâches en parallèle. Cette propriété importante démontre que le Rafale, avec son équipement,

peut multiplier les tâches et le nombre de cibles à poursuivre en profitant de plusieurs modes radar en simultané. Un mode permet de remplir une fonctionnalité différente en modifiant les paramètres du radar. De plus, le radar possède une résistance au brouillage, car il peut utiliser différentes fréquences à la fois, et il a une fiabilité augmentée du fait de la redondance des antennes.

1.2.6 Conclusion

Nous avons présenté les types de senseurs pouvant être embarqués sur les plateformes. Les plateformes ont des missions complémentaires au sein du GAN, et les senseurs possèdent des conditions de fonctionnement et des rôles spécifiques pour le développement de la situation tactique. Le positionnement des plateformes, les conditions sur l'environnement (pluie, nuit, brouillard, etc.), les caractéristiques des menaces à pister ou les portées de fonctionnement des senseurs sont des contraintes orientant l'utilisation des équipements des plateformes. Le nombre de paramètres à prendre en compte par les opérateurs augmente à mesure que les systèmes de senseurs évoluent. Les actions collaboratives ajoutent des contraintes supplémentaires que le système doit prendre en compte afin d'aider l'opérateur dans sa prise de décision, sur le choix d'utilisation distribuée des senseurs en fonction du contexte dans lequel ils évoluent.

1.3 Spécificités et contraintes du milieu naval

1.3.1 Spécificités de l'environnement

Les radars navals doivent être capables de détecter et de pister des cibles de natures très variées, aériennes ou navales, de petites ou de grandes tailles. Les formes d'ondes et le traitement du signal varient selon les menaces détectées. La chaine de réception doit pouvoir éviter la saturation du récepteur pour détecter des porteurs de petite taille lorsqu'au même moment une cible volumineuse apparait [BCC+14].

De plus, l'environnement induit de fortes contraintes évoluants selon le milieu. L'environnement est défini par ses frontières naturelles, la propagation et la géographie des milieux, par exemple, la présence de montagnes près des côtes induit de forts échos de sols. Cet espace partagé requiert la distinction des fréquences électromagnétiques pour coordonner les activités militaires et civiles, tout en détectant les événements inhabituels au sein du flux de mouvements alliés et neutres. Les plateformes navales sont amenés à

agir en pleine mer, mais également près des côtes où des menaces terrestres peuvent apparaitre dues à la proximité du continent. Par exemple, les radars de défense sol-air comme le GM200 peuvent surveiller les menaces aériennes de la force aéronavale et engager un aéronef grâce à un guidage de missile depuis le continent.

1.3.2 Spécificités de la propagation des ondes électromagnétique

Plusieurs phénomènes de propagation des ondes ont lieu en raison de la singularité de l'environnement maritime. En pleine mer, peu d'obstacles existent dans la plupart des scénarios, dans ce cas, seules l'eau et l'atmosphère peuvent éventuellement provoquer une réflexion des signaux émis. Les variations de pression atmosphérique, de température et d'humidité influent sur la propagation des signaux radio, ce qui peut avoir des répercussions sur la détection des cibles en fonction de différents facteurs tels que l'altitude, la distance et la présence de l'horizon radio. Ce dernier est dû à la courbure de la Terre, au-delà de laquelle une zone d'ombre peut se former. L'équation 1.1 définit la distance de l'horizon radio d'un radar situé à une altitude h avec la constante du rayon de la terre R_f égale à 8500 km :

$$d = \sqrt{2 * R_f * h} \tag{1.1}$$

Si on prend par exemple un radar situé sur un mât de navire à 40 m de hauteur, la distance de détection d'un objet à la surface de l'eau ne pourra pas dépasser 26 km (en considérant que la portée radar soit supérieure). La collaboration avec des plateformes aériennes pourraient donc apporter une vision plus étendue du théâtre et la position de la plateforme pourra être considérée dans le choix des actions à mener.

Un autre phénomène apparait aussi en raison de la réflexion du signal radar avec l'eau ou avec des obstacles comme du relief. Cet évènement introduit des variations de phases entre les trajets réfléchis de l'onde et crée ce qu'on appelle des franges d'interférences, pouvant être constructives ou destructives. Les radars navals exploitent ces propriétés de l'environnement pour dépasser l'horizon radio.

1.3.3 Comparaisons entre les plateformes navales et aériennes

Les plateformes navales et aériennes présentent des différences fondamentales dans leur conception, leurs missions et leurs contraintes opérationnelles. D'une part, les radars navals assurent la détection de cibles de surface, mais aussi la détection de missiles ou cibles aériennes. Cette polyvalence des radars navals implique des défis supplémentaires, car les

cibles peuvent varier considérablement en termes de vitesse et de surface équivalente radar (SER), nécessitant une adaptabilité accrue des systèmes de détection. En complément des radars de navigation et des radars de poursuite, certaines plateformes navales embarquent deux radars de veille. Le premier est destiné à la détection à longue portée des menaces potentielles, tandis que le second est davantage orienté vers la défense ou la poursuite de tirs.

Outre les différences fonctionnelles, les missions des plateformes navales sont souvent caractérisées par leur longue durée, s'étendant parfois sur plusieurs mois, et par des distances importantes entre les différentes plateformes, pouvant aller de quelques dizaines de mètres à des kilomètres. Ces missions prolongées exigent une logistique complexe pour assurer le ravitaillement, la maintenance et le soutien des équipages sur de longues périodes en mer. De plus, l'hétérogénéité des plateformes navales, comprenant divers types de navires et de senseurs, entraîne des défis d'interconnexion et d'interopérabilité, nécessitant une collaboration étroite pour garantir le succès des opérations.

D'autre part, les missions aériennes, notamment celles des Rafales, sont souvent caractérisées par leur flexibilité et leur capacité à être rapidement déployées sur des théâtres d'opérations variés. Les missions aériennes peuvent être de courte durée et nécessitent une haute réactivité pour répondre aux menaces changeantes et aux situations d'urgence. La distance entre les plateformes aériennes, telles que les escadrons de Rafales, est généralement plus courte que celle entre les navires, ce qui peut avoir un impact sur la rapidité de communication entre ces plateformes.

En résumé, les plateformes navales et aériennes présentent des différences significatives en termes de missions, de contraintes opérationnelles et de logistique, ce qui complexifie la conception d'une architecture commune. Nous pouvons résumer les différences entre les domaines aéronautique et naval dans le tableau suivant :

	Domaine aéronautique	Domaine naval
Dimension	Volume 3D	Surface 2D
Vitesse	Rapide ($\sim$ 300 m/s)	Lent ($\sim$ 15 m/s)
Manœuvrabilité	Importante	Faible
Taux rafraichissement données	Fréquent	Moins fréquent
Portée de détection	Jusqu'à 1000 km	Jusqu'à 500 km
Espace équipement	Très restreint	Large

Table 1.1 – Différence entre les domaines naval et aéronautique

1.4 Les menaces en mer, aujourd'hui et demain

Les menaces et les détections dans le domaine naval sont diverses [Ger06] :

— Des cibles aériennes comme des avions, des hélicoptères, des drones ou des missiles ;

— Des menaces potentielles en surface comme des bateaux de pêche, des porte-conteneurs ou des frégates ;

— Des dangers sous la mer comme des sous-marins, des mines marines, des torpilles ou des systèmes sonar.

Les nouvelles menaces tendent à réduire leur SER afin de diminuer leur probabilité de détection. Les drones sont des aéronefs autonomes ou pilotés à distance, leur faible coût de fabrication permet d'en déployer plusieurs sur chaque plateforme. Leur caractéristique d'autonomie nécessite une connaissance de l'environnement complétée par des capteurs et des échanges par le biais de liaisons de données. Ces informations déterminent la meilleure stratégie pour atteindre leurs objectifs. Ils peuvent également effectuer des missions de reconnaissance dangereuses, limitant les pertes humaines alliées. Les bâtiments ennemis pourront aussi en être équipés et les utiliser pour des missions d'attaques en intégrant des charges explosives à leur bord. La menace balistique est aussi présente avec des missiles pouvant être lancés à des milliers de kilomètres. Il convient donc de concevoir des systèmes de défense rapides et performants pour s'en protéger.

Certaines missions amènent des menaces spécifiques, ces missions incluent la lutte contre l'immigration clandestine, la surveillance des territoires, la police des pêches, la lutte contre la pollution ou le narcotrafic et peuvent évoluer dans un futur proche. Les navires accomplissent des missions différentes et ne gèrent pas les mêmes types de menaces. Par exemple les patrouilleurs de service public comme le Flamant ou le Cormoran luttent contre la pollution des mers et surveillent la pêche. Les frégates de défense aérienne FDA comme le Forbin ou le Chevalier Paul s'occupent de la lutte antiaérienne et commandent la défense aérienne depuis la mer. Les contraintes lors de leurs missions diffèrent donc en fonction des caractéristiques des navires. Un navire de guerre de grande taille possèdera une polyvalence accrue avec une meilleure autonomie en mer (carburant et vivres), un équipage plus important, une capacité de défense plus grande et une résilience aux conditions météorologiques améliorée.

Outre les menaces au-dessus de l'eau, de nombreuses menaces proviennent de sous la mer. C'est un environnement complexe et dense en termes de perturbations, qui peuvent provenir de l'homme comme de l'animal : faux échos de cétacés, génération de lumière

avec les algues, la pêche, les câbles sous-marins, les épaves, les bruits sismiques, la pluie, le vent, l'orage, le courant, les marées, la pression, la température, etc. Pour les plateformes navales, les menaces proviennent des mines marines et des sous-marins. La guerre des mines désigne le cadre des missions relatif à la recherche et la neutralisation des menaces de type mines, pouvant endommager les sous-marins et les frégates alliées. De nombreux moyens de détection des menaces sous-marines existent comme les sonars : le sonar de coque, le sonar trempé ou le sonar remorqué (le Captas-4 équipe les FREMM).

Les navires sont par ailleurs amenés à effectuer des missions toujours plus proches des côtes. Les menaces y sont supérieures, car c'est un milieu où les détections sont omniprésentes et peuvent cacher des menaces asymétriques. Par définition, une menace de ce type exploitera les faiblesses de l'opposant en utilisant des méthodes non conventionnelles. Par exemple, certains actes de pirateries amènent à détourner des navires ou utiliser des bateaux rapides à dévoilement tardif, dont la classification se voit complexifiée. D'autres menaces asymétriques peuvent être citées, comme les nouveaux missiles hypervéloces toujours plus rapides et furtifs ou les missiles *seaskimmers* (au ras de l'eau) qui poussent les systémiers à améliorer les systèmes de défenses des navires.

Ces évolutions amènent par ailleurs le monde maritime à se numériser de plus en plus [dAM16], la technologie impacte les systèmes de communication, de navigation et de gestion des navires. Le réseau des senseurs pourrait par exemple être attaqué si des moyens de protection ne sont pas appliqués comme des pare-feux ou des systèmes de détection d'intrusion (IDS). D'autres questions peuvent se poser, comme l'accès à internet, à la Wifi et l'introduction d'équipements personnels connectés par les membres de l'équipage au réseau à bord et à internet. Les différents systèmes présents sur un navire doivent être cloisonnés et les besoins d'interconnexions détaillés. Quelques exemples de vulnérabilités de systèmes peuvent être présentés :

- ➔ Le système AIS (*Automatic Identification System*) [YWW+19] présent sur chaque navire est obligatoire et gère l'envoi et la réception des positions GPS, des vitesses, des caps, des types, des lieux et de l'heure d'arrivée des navires, vers et depuis les navires environnants. L'accessibilité des données induit une potentielle vulnérabilité face au brouillage, à l'envoi de fausses informations et à la transmission de virus informatiques. Par exemple, un faux signal de détresse pourrait être transmis pour ensuite tendre une embuscade au bateau qui viendra aider.
- ➔ Les signaux GPS (*Global Position System*) des navires civils ne sont pas protégés par chiffrement et peuvent donc être interceptés et dupliqués. De plus, le brouillage

intentionnel et l'interférence volontaire sont d'autres vulnérabilités du système.

L'aspect cyberdéfense de l'architecture ne sera pas considéré dans ces travaux de thèses. Ces sujets font l'objet de travaux dans le cadre de la Chaire de cyberdéfense des systèmes navals à Brest et sont au cœur de cette problématique complexe [Nao22, Sul20, ML17, Cos18].

1.5 Contexte industriel et opérationnel

Des travaux préliminaires étatiques ont porté sur le domaine coopératif, centré sur le partage d'informations d'intérêt radars entre frégates, avec la veille coopérative navale (VCN) qui permet de détecter plus vite et plus loin (voir figure 1.4). Ces travaux sont poursuivis par la veille coopérative aéro-maritime (VCAM) qui étend le concept aux autres senseurs de types GE ou optronique ainsi qu'aux plateformes aériennes et terrestres. Ce sont des travaux en cours qui se poursuivront par le Combat collaboratif naval (CCN) qui ajoutera la couche d'engagement réparti entre les plateformes navales [ble19]. Les forces alliées se doivent de coopérer et de partager de l'information rapidement dans un environnement qui évolue en permanence et où les menaces peuvent impacter le comportement des systèmes.

Les systèmes actuels ont cependant des limites, la quantité de données échangées augmente plus rapidement que la taille des canaux de transmissions, le système cherche donc à partager de gros volumes de données sans pour autant avoir le débit nécessaire aux échanges. En conséquence, les nouveaux développement doivent considérer la réduction des besoins en bande passante et catégoriser l'importance des informations à transmettre.

Actuellement, la donnée, provenant des senseurs de la plateforme ou de l'extérieur, est transmise au système de combat selon une architecture centralisée. Dans les architectures futures, les senseurs pourraient sélectionner les données à transmettre aux autres plateformes pour ne pas surcharger la bande passante. Ces données pourraient être enrichies en modifiant l'architecture ou en améliorant les canaux de transmissions.

Pour développer de nouveaux concepts, il est primordial de relever les enjeux liés à la connectivité entre les plateformes et les systèmes, les temps de réponse peuvent avoir un impact sur les choix de conception selon l'échelle de temps où la conception se situe [TLL17] :

- Le « temps réfléchi » réfère aux temps de traitement nécessitant réflection avant décision, l'humain intervient largement à ce niveau.

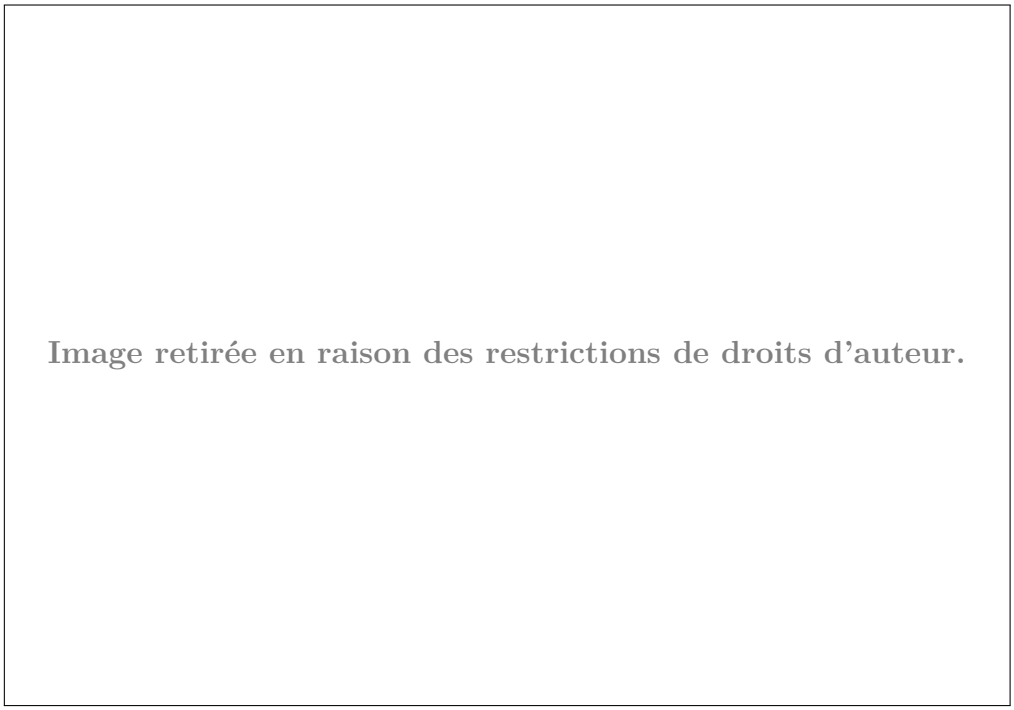

FIGURE 1.4 – Partage des pistes senseur avec les plateformes alliées.

- Le « temps réflexe » permet à l'humain de traiter l'information qu'il reçoit avec une latence de quelques secondes, jusqu'à une minute au maximum.
- Le « temps réel » doit faire preuve d'une réactivité importante avec un besoin de traitement temps-réel pour respecter les échéances d'engagement, par exemple pour se défendre face à une menace hyper véloce. La latence doit être au maximum de quelques millisecondes et le traitement de l'information puis l'action qui en découle doivent se faire dans la seconde.
- Le « temps quasi réel » est un concept proche du temps réel, sauf qu'un retard lié au traitement automatique autorise un traitement de l'information et une prise de décision de l'ordre de la dizaine de secondes.

La donnée est omniprésente, elle alimente les systèmes d'information de mission et peut provenir de diverses sources : le centre opérationnel de commandement, les bases de données techniques de GE et de Cyber, les systèmes embarqués sur la plateforme (système de mission, système de navigation, etc.), des senseurs après une synthèse des données brutes, du web, d'images satellites, etc. Ces données peuvent englober les caractéristiques des plateformes et des senseurs connus, ainsi que les événements ou l'environnement. La gestion de ces données doit prendre en compte leur hétérogénéité (signaux bruts/traités,

images, vidéos, etc.), leur volume, leur débit, leur qualité, leur validité, leur fiabilité et leur niveau de classification.

Le développement des technologies à base d'intelligence artificielle (IA) s'est accéléré ces dernières années. L'IA permet de prendre des décisions de défense, là où un être humain n'aurait pas le temps nécessaire à la réflexion [MBL+20]. La spécificité de l'environnement, la diversité des plateformes et de leurs senseurs, ainsi que la complexité croissante des missions nécessitent une coopération entre les navires et leurs partenaires qui ne peut plus être accomplie efficacement par un humain seul. L'IA, en traitant les données qu'elle reçoit via les senseurs, pourra réagir rapidement, partager des informations dans le réseau et proposer des actions coordonnées aux opérateurs. Le système multi-agent (SMA) s'inscrit comme une technologie d'IA distribuée et peut, par exemple, aider l'humain dans sa prise de décision en proposant de planifier les actions des senseurs de la plateforme grâce à la connaissance des senseurs, des ressources disponibles et de l'environnement. Cette technologie fera l'objet d'un point de notre état de l'art et nous monterons comment elle répond à notre problématique de l'allocation de tâches à des senseurs.

1.6 Enjeux de la thèse

1.6.1 Conception d'une architecture système

Etapes de conception d'une architecture

La conception d'une architecture de système est un processus complexe qui implique plusieurs étapes à suivre pour garantir un fonctionnement cohérent avec les besoins de son utilisateur. La première étape consiste à identifier les besoins et les exigences de l'utilisateur du système qui seront répondus par des fonctions et des contraintes. Les besoins fonctionnels et non-fonctionnels de l'architecture doivent être étudiés, afin de proposer des solutions applicatives permettant d'y répondre. Ensuite, le matériel et les composants de l'architecture sont identifiés : par exemple, des serveurs ou des unités de calculs. Les interactions et les interfaces entre les composants sont également définies. Une première version simplifiée voit le jour avant de passer à l'étape de conception détaillée de l'architecture. Enfin, l'architecture doit être validée et vérifiée à l'aide de simulations, de tests ou d'analyses pour confronter les besoins et les exigences, définis en amont, aux performances réelles du système.

Nous pouvons prendre l'exemple des travaux d'architecture du projet européen CA-

MELOT [UPT+18], où des ingénieurs R&D conçoivent une architecture de type C2 (*Command and Control*). L'objectif est d'exploiter une flotte de drones hétérogènes dans l'optique de surveiller les frontières extérieures de l'Union européenne, qu'elles soient terrestres, aériennes ou maritimes. Dans le cadre de l'analyse fonctionnelle du système, des entretiens ont été menés avec les utilisateurs, tels que la marine et les forces armées. Ces entretiens ont permis d'explorer divers aspects de l'architecture, notamment les besoins en plateforme mobile, en interopérabilité, en coopération et en intelligence ouverte. Cette étape d'analyse met en lumière l'importance de comprendre les besoins des utilisateurs pour concevoir un système adapté.

À la suite de ces entretiens, les besoins fonctionnels et non-fonctionnels ont pu être définis. Les besoins fonctionnels spécifient les opérations et activités que l'architecture doit être capable de faire. Dans le cas de CAMELOT, cela inclut des descriptions de données d'entrée du système, les activités effectuées par le système, la description des flux de travaux du système et d'autres sorties, et la manière dont le système répond aux exigences réglementaires. Les besoins non-fonctionnels spécifient le comportement et la performance des opérations et activités, ce sont des critères qui permettent de juger les opérations d'un système. Des exemples typiques de besoins non-fonctionnels sont : la performance (temps de réponse ou débit), la scalabilité, la capacité, la disponibilité, la confiance, la sécurité ou l'interopérabilité. Un exemple d'exigence non-fonctionnelle pourrait être : « *dans le système, l'opération X doit être effectuée dans un délai maximum de 5 secondes* ». Un autre exemple sur l'exigence d'interopérabilité serait : « *le système échange ses données de manière transparente avec les autres plateformes ou composants* ».

Répondre aux besoins opérationnels

Les besoins opérationnels correspondent aux exigences identifiées par les utilisateurs du système. En général, les utilisateurs sont les différents opérateurs de la plateforme. Ces besoins sont définis en discutant avec ces personnes qui ont des problématiques en mer auxquelles l'architecte pourrait ne pas penser. De nombreuses documentations internes à l'entreprise étudient et expriment déjà ces exigences.

Le travail d'architecture est un travail difficile, le contexte des systèmes des plateformes de combat est incertain et les architectures évoluent rapidement. Des standards d'architecture sont parfois établis, puis largement diffusés par la suite (par exemple les architectures ouvertes présentées en section 2.2). La conception d'éléments d'architecture doit considérer leur possible réutilisabilité dans d'autres systèmes. De plus, la récupéra-

tion de concepts d'architectures existantes permet dans certains cas de réduire les coûts de conception [LMS+08]. C'est dans cette optique que nous énumérons la liste des caractéristiques importantes pour l'architecture d'un point de vue opérationnel :

- ➤ Le système doit être commandé par des ordres provenant d'un opérateur et ses instructions amènent l'utilisation physique des senseurs par le système. Des technologies, des méthodes et des outils doivent pouvoir traduire un ordre opérationnel en demandes d'actions à réaliser par les senseurs. Par exemple, l'utilisateur peut formaliser son ordre via une IHM (Interface Homme/Machine) [Cha07] afin que le système comprenne la requête et utilise les senseurs en conséquence.
- ➤ L'affectation de tâches à un senseur doit se faire dynamiquement. L'architecture doit pouvoir réduire le temps de prise de décision (voir sous-partie suivante sur la boucle OODA). Les mécanismes d'allocations de tâches senseurs doivent se faire en un temps limité pour montrer une amélioration par rapport aux systèmes actuels.
- ➤ L'architecture doit être générique, le contexte comprend des plateformes navales multiples et variées ainsi que des porteurs aériens qui ajoutent un niveau d'hétérogénéité important.
- ➤ Le système doit pouvoir passer à l'échelle, car nous travaillons dans un contexte multiplateforme avec des fonctions de collaboration amenées à évoluer. Un GAN est composé d'une dizaine de plateformes, mais de futures opérations pourraient voir l'apparition d'essaim de drones ou de collaborations internations, ayant pour conséquence d'augmenter la taille du réseau et le volume des échanges.
- ➤ La communication entre les systèmes augmentera en raison des échanges collaboratifs, le besoin en bande passante doit être quantifié et ajusté.
- ➤ L'importance de l'IA est présentée en section 1.5 et fait partie des évolutions voulues dans les prochaines architectures. L'IA peut apporter une aide à l'opérateur dans ses prises de décisions, et peut améliorer l'autonomie du système [MBL+20].
- ➤ La topologie centralisée de l'architecture actuelle [CLL05] doit être revue, les points faibles de celle-ci ne sont plus à démontrer (point unique de défaillance, difficulté de passage à l'échelle, goulot d'étranglement, etc.). Des choix d'architectures décentralisées, distribuées ou hybrides sont à étudier pour satisfaire les nouveaux besoins mentionnées dans cette section.
- ➤ L'architecture se doit d'être robuste, ce qui signifie qu'elle doit garantir la disponibilité et la sécurité du système, assurant ainsi un fonctionnement continu et fiable.

Dans le contexte de la guerre navale, le système doit pouvoir continuer de fonctionner malgré une attaque de l'ennemi qui pourrait conduire à la perte d'un senseur en activité, ou pire, du navire entier.

➤ Les résultats, lors du déroulement d'un scénario, doivent être reproductibles en raison de la criticité du système, ils ne doivent pas changer entre plusieurs exécutions dans des conditions similaires.

➤ Le système doit gérer l'ordre de priorité des tâches en considérant également les demandes prioritaires d'utilisation des ressources du système par l'utilisateur, c'est-à-dire que le système ne doit pas bloquer l'opérateur s'il souhaite effectuer une tâche en particulier.

La boucle OODA

L'article [BU20] soulève un enjeu important, celui de réduire au maximum le temps de prise de décision en s'inspirant du modèle de la boucle OODA (*Observe, Orient, Decide and Act*).

« Face à la furtivité et à la montée en gamme des adversaires dans le domaine aérien, les axes principaux de raccourcissement de cette boucle décisionnelle sont :

- La détection collaborative (affinement de la *Common Operational Picture*) par répartition et orientation coordonnées des senseurs passifs, radars, optroniques ;
- L'engagement collaboratif : guidage missile par répartition et orientation coordonnées des senseurs et des effecteurs ;
- Le combat collaboratif défensif et de survie (manœuvres, brouillage, leurrage, tirs).

Cette boucle décisionnelle optimisée exige :

- Le décloisonnement de l'usage des senseurs et effecteurs ;
- Un réseau local autonome, discret et robuste ;
- La supervision locale, en temps réel, des moyens (Interaction Homme Système adaptée) ;
- Un haut niveau d'interopérabilité. »

D'autres besoins opérationnels apparaissent dans le cadre des études sur le CCN, ce sont les besoins liés à la détection, la localisation, la poursuite et le pistage de cibles, l'engagement et l'architecture des systèmes. La maîtrise du spectre électromagnétique repose sur la gestion et la coordination des émissions alliées dans le temps et la mise en œuvre

d'actions contre le spectre ennemi, cette maîtrise est cruciale dans l'élaboration des stratégies actuelles. Ensuite, le traitement des nouvelles menaces doit être rapide. Lorsqu'une menace est détectée, elle doit ensuite être classifiée et identifiée en minimisant les fausses classifications et le nombre de menaces inconnues. Le pistage doit se faire de manière continue malgré l'environnement complexe et incertain. Les plateformes de la flotte ont besoin de communications rapides et fiables pour partager au plus tôt les renseignements sur les menaces et agir au plus vite. Au niveau de l'architecture du système, les besoins principaux sont liés aux systèmes d'arme, aux senseurs et aux systèmes de communication (réseau). Ils doivent pouvoir s'intégrer au système et communiquer l'information aux uns et aux autres afin d'établir une situation tactique globale qui pourra améliorer les capacités de chaque partie prenante.

Répondre aux exigences industrielles

La conception de l'architecture d'un système nécessite d'étudier les exigences industrielles qui encadrent les processus. Cela peut aller de normes à respecter à des pratiques considérées par l'entreprise. Dans ce contexte, la modularité dans l'architecture peut être interprétée comme la capacité à diviser le système en modules distincts et interconnectés, chacun responsable d'une fonction spécifique. Cette approche favorise la réutilisation des modules existants et facilite l'ajout, la suppression ou la modification de fonctionnalités sans affecter l'ensemble du système. La modularité peut être réalisée grâce à l'utilisation de standards et à l'établissement d'interfaces capables d'interconnecter les sous-systèmes actuels et futurs. La communication entre les sous-systèmes peut se faire grâce à des *middlewares* du style MOM (*Message Oriented Middleware*), qui fournissent une couche d'abstraction et améliorent l'interopérabilité, les performances, la tolérance aux pannes et la sécurité.

De plus, le système pourra évoluer dans son développement, que ce soit par la mise à jour de logiciels ou par la modification du matériel. Les senseurs peuvent avoir de nouvelles fonctionnalités à intégrer, ainsi que de nouveaux algorithmes de traitement. Le réseau peut évoluer, les câbles de connectivité peuvent par exemple être remplacés par de la fibre sur un réseau local, bien plus rapide, augmentant le volume de données pouvant être échangées. Un senseur de la plateforme doit pouvoir être remplacé ou amélioré en réduisant l'impact sur le système, par cet exemple nous évoquons la capacité d'adaptation du système. L'architecture doit s'adapter à une évolution des senseurs, que ce soit par leur nombre, leur taille, leur puissance ou leurs interactions.

Pour répondre à ces exigences, le système doit fournir une capacité « Plug-and-Play », ce qui signifie que tout module développé selon des normes et des standards d'interopérabilité peut s'intégrer au système. Le module doit être en mesure d'échanger des informations avec les autres modules du système en encodant les données pour simplifier leur transport sur le réseau. Ce processus est appelé sérialisation.

Ensuite, l'architecture, conçue pour du combat collaboratif multi-plateforme, devra être compatible avec le système existant, tout en garantissant son fonctionnement en état nominal en cas de défaillance. Si le navire se trouve isolé en mer, il pourra encore compter sur la collaboration de ses propres senseurs, même si cette collaboration reste plus limitée. Le système est amené à fonctionner dans un environnement où le nombre de moyens de transports et d'embarcations diverses peut être élevé, englobant les transports aériens et maritimes réguliers tels que les avions et les cargos, ainsi que les navires de pêche et les voiliers. Au niveau du système, les limites capacitaires sur l'utilisation des ressources contraignent celui-ci à faire des choix de tâches à assigner, en fonction de critères tels que la priorité ou la criticité. Ainsi, la détection et la poursuite de toutes les cibles de l'environnement ne sera pas toujours possible.

Enfin, le travail de conception d'une architecture d'un système est influencé par un nombre important de facteurs et de perspectives. L'architecture évolue en fonction des exigences, des besoins et des contraintes définis en amont. L'architecte cherche à atteindre une stabilité entre ces objectifs parfois contradictoires. Chaque personne intervenant sur le système voit son architecture sous un angle différent appelé « point de vue ». Dans ces conditions, l'architecture n'aboutira qu'à force de compromis et d'équilibrages dans la compréhension des points de vue.

1.6.2 L'allocation des tâches senseurs

La problématique de l'allocation de tâches [Jia15], dans notre contexte à des senseurs, consiste à faire correspondre des tâches à un ensemble de senseurs, dans le but de remplir des actions, collaboratives ou non, en maximisant l'utilité globale dans un environnement limité en ressources. L'utilité globale se traduit par la somme des valeurs ou des bénéfices de chaque tâche contribuant à répondre aux objectifs et besoins du système. Par simplification, l'utilité d'une tâche k est notée u_k et se calcule en fonction de différents paramètres p, par exemple l'efficacité ou la qualité de service des senseurs. La formule

suivante définit l'utilité globale :

$$U = \sum_{k=1}^{n} u_k(p) \tag{1.2}$$

Definition 3 (Tâche) Une tâche correspond à une unité de travail qui doit souvent être terminée dans un certain délai [TE20]. Dans notre contexte, une tâche correspond à la réservation d'une ressource pendant une période donnée. Une tâche peut avoir des contraintes de précédence si une autre tâche doit être effectuée avant celle-ci. Une tâche peut se retrouver en concurrence avec une autre tâche lorsque la même ressource est demandée simultanément. Dans notre contexte, un service peut nécessiter une ou plusieurs tâches pour fonctionner.

De manière générale, le terme « ressource » englobe les senseurs, les capacités de calcul, la mémoire ou l'énergie. Les ressources servent à l'exécution de tâches et peuvent avoir besoin d'un temps de préparation avant d'être utilisées. Par exemple, une antenne radar peut nécessiter un changement de fréquence ou d'orientation selon la tâche qui lui est assignée. Nous utiliserons souvent ce terme pour définir la ressource spécifique qu'est le senseur. Le senseur nécessite généralement d'autres types de ressources pour mener à bien ses tâches et nous supposons qu'il y a accès sans collisions avec les autres senseurs.

Il existe des tâches mono-senseurs et multi-senseurs ainsi que des allocations instantanées ou planifiées dans le temps. De plus, un senseur peut effectuer une ou plusieurs tâches simultanément. Nous nous limiterons aux allocations de tâches instantanées dans le cadre de la thèse, les ressources sont allouées dans un délai très court et le plus rapidement possible à partir du moment où les tâches sont demandées. L'allocation de tâches sur un axe temporel long correspond à une problématique d'ordonnancement déjà étudiée dans de précédents travaux de thèse [Gri18, Gua21].

Les senseurs peuvent accomplir des tâches, plus précisément des services collaboratifs, comme le TDOA (*Time Difference of Arrival*) ou l'imagerie SAR (*Synthetic Aperture Radar*). Une tâche d'imagerie SAR sur une zone peut utiliser plusieurs senseurs appartenant à des plateformes différentes pour « scanner » la zone plus rapidement. Des ressources sont alors réservées pour accomplir des tâches sélectionnées selon plusieurs paramètres et critères. L'utilisation des ressources senseurs peut être impactée par le déplacement des plateformes, une modification d'un objectif opérationnel, un changement lors de la mission, la mise à jour de pistes sur la situation tactique ainsi que par un changement de propriétés ou de disponibilité sur les ressources du système. Toutes ces données sont les entrées de l'organe au sein de l'architecture qui sera en charge de la réservation des

ressources. Son rôle est donc de proposer une allocation de tâches cohérente avec les ressources disponibles, tout en composant avec un environnement instable pouvant impacter les tâches établies. Nous appelons « plan senseur » un ensemble de tâches permettant de réaliser une fonction senseur.

Definition 4 (Plan senseur) Un plan senseur P_k [Gri18] est composé d'un ensemble de tâches T_i, avec des contraintes les liant entre elles, affectées à une ou plusieurs ressources.

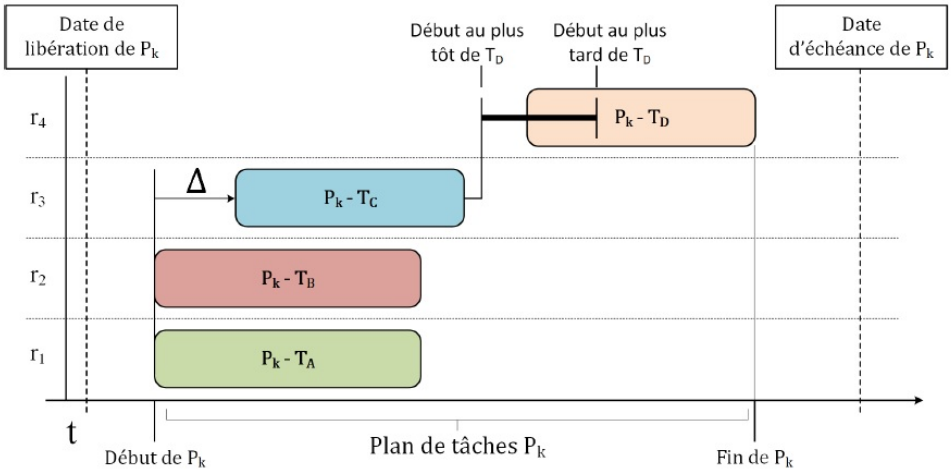

FIGURE 1.5 – Schéma d'un plan senseur, de [Gri18].

Concrètement, un plan senseur sera proposé à un instant donné et devra être réalisé le plus tôt possible avec une date d'échéance au-delà de laquelle le plan ne sera plus valide (en raison de l'instabilité de l'environnement). Pour donner un exemple d'un plan senseur complexe comprenant plusieurs tâches avec une contrainte de précédence, nous pouvons reprendre l'exemple du SAR (dont le fonctionnement est détaillé en annexe). Le plan senseur du SAR nécessite une phase d'acquisition et une phase de traitement. La première phase comprend un ensemble de tâches à effectuer en parallèle sur deux types de ressources : l'antenne active du radar r_1 et la fréquence d'émission r_2 (pour ne pas brouiller d'autres équipements qui utiliseraient cette ressource). Pour compléter le plan senseur du SAR, la phase de traitement intervient après la phase d'acquisition et utilise de la ressource de calcul r_4 pour exploiter les signaux capturés.

Nous avons présenté un exemple de plan senseur complexe, mais, dans le cadre de la thèse, nous supposons que la ressource senseur utilise l'antenne active et la fréquence sous une seule et unique tâche. La phase de traitement, s'il y en a une, sera également comprise

dans la durée du plan de tâches sans contraintes de concurrences avec d'autres ressources senseurs. Ainsi, nous posons l'hypothèse simplificatrice qu'un plan senseur comprend une unique tâche sur la ressource senseur, et que le senseur englobe plusieurs autres ressources.

L'allocation de tâches à des senseurs peut être complexe. Dans notre contexte, plusieurs contraintes se posent. D'une part, le besoin de réactivité du système, face à des événements critiques et prioritaires qui peuvent intervenir à tout moment, peut affecter les plans de tâches en cours par la nécessité de préempter des ressources en cours d'utilisation. Ensuite, une ressource peut être utilisée pour réaliser plusieurs tâches. Par exemple, un radar à balayage électronique peut exécuter trois tâches que sont : imager une zone, surveiller l'environnement et poursuivre un objet. Les tâches doivent donc se succéder sur un plan senseur pour savoir quel taux d'occupation de la ressource elles peuvent obtenir. Pour donner un autre exemple de l'impact d'une tâche sur l'utilisation de la ressource, la poursuite d'une cible demande un taux de rafraichissement plus ou moins court selon la vitesse de la cible, si la cible se déplace lentement, la ressource sera très peu utilisée laissant place aux autres tâches.

Pour répondre à cette problématique, nous pouvons agir sur la notion de services et de qualité de services, cela permettra dans un premier temps de refuser des plans senseurs qui ne sont pas prioritaires au profit de plans vitaux pour la coalition. Un travail sur la définition d'une qualité de services appliquée à la planification de nos tâches pourra donc être une piste pour répondre à nos besoins.

1.6.3 Défis et enjeux des systèmes distribués pour la conception de l'architecture

Le contexte multi-plateforme de nos problématiques implique des enjeux dans le cadre des systèmes et des réseaux distribués. Ces challenges s'appliquent donc à la résolution de la problématique d'allocation de tâches à des senseurs distribués.

Definition 5 (Système distribué) Un système distribué est un système dans lequel les composants, situés sur des entités en réseau, communiquent et coordonnent leurs actions en échangeant des messages [CDKB05].

Challenges des systèmes distribués

Les systèmes distribués apportent leur lot de challenges :

- L'hétérogénéité des composants : il y a un besoin en protocoles de communication, car les types de données et la représentation de chaque système doivent être pris en compte lors des échanges. Des middlewares comme CORBA ou Java RMI permettent de masquer l'hétérogénéité des réseaux, du matériel et des systèmes opérants.
- L'ouverture du système : elle vise à intégrer ou à modifier des composants dans le système. Cela peut également impliquer l'ajout de nouveaux services et de ressources supplémentaires pour les utilisateurs. Cette ouverture ne peut être achevée que par le partage de documentation détaillée sur l'utilisation des interfaces. Par exemple, les spécifications des protocoles Internet sont communément publiées dans des RFC (*Requests For Comments*).
- La mise à l'échelle : le système doit vérifier le coût de la ressource physique (serveurs) en fonction du nombre d'utilisateurs, contrôler la perte de performance, empêcher d'atteindre l'épuisement de la ressource, éviter les goulots d'étranglement. Les caches et la réplication sont des exemples de solutions lorsque les ressources sont fréquemment utilisées, ce qui entraine le problème de synchronisation et de cohérences entre les réplications.
- Gestion des pannes et échecs : les tâches dans les systèmes peuvent produire des résultats incorrects, voire aucun résultat, en cas de défaillance. Les tâches en cours peuvent également s'arrêter sans avoir fini, bloquant potentiellement des ressources non libérées. Pour résoudre certaines parties des problèmes de pannes ou d'échecs, plusieurs méthodes peuvent être utilisées. Par exemple, la technique de détection de fautes par *checksum* permet de vérifier les données corrompues dans un message ou un fichier. Les pannes peuvent aussi être cachées en retransmettant des messages ou en créant de la redondance par duplication de la donnée sur des disques ou serveurs différents. Le *rollback* permet de remettre le système dans le dernier état cohérent suite à un échec. De plus, la réplication des processeurs ou des services peut être envisagée pour améliorer la tolérance aux pannes. Cependant, il est important de noter que ces solutions ne sont que partielles et qu'elles ne garantissent pas une résolution complète des problèmes de pannes ou d'échecs.
- La concurrence : les entités peuvent vouloir accéder à une même ressource en même temps, ce qui peut bloquer la ressource (*deadlocks*).
- La transparence : elle fait référence à la capacité du système à dissimuler à l'utilisateur des détails d'implémentation afin de lui offrir une perception unifiée. Six différents types de transparence existent : l'accès, la réplication, la localisation, la

migration, la concurrence et la défaillance.

- La qualité de service : le système doit fournir des garanties au niveau des temps de réponse. La qualité de service implique que le système propose les ressources de calcul et de communication nécessaires à l'exécution d'une tâche dans les temps.

Internet ou les réseaux mobiles illustrent des exemples de systèmes distribués. La proximité géographique ou la simulation locale d'un système distribué ne réduit pas l'importance des composants communicants par messages. Les mêmes défis et enjeux s'appliquent, qu'il s'agisse d'interactions entre des serveurs distants ou de processus locaux sur une seule et même machine. Dans nos travaux, des processus sont en concurrence et communiquent entre eux pour l'utilisation de la ressource partagée dans le but d'accomplir leurs tâches. En l'absence d'horloge globale, les programmes rencontrent des difficultés à déterminer si les ressources qu'ils demandent sont en cours d'exécution sur d'autres processus. Les composants peuvent être défaillants individuellement, alors que la communication avec d'autres composants se poursuit. En cas de pannes partielles ou de dégradation du système, la gestion des ressources et le maintien de la cohérence du système sont des défis. Dans un contexte comme celui présenté dans cette thèse, un fonctionnement en mode dégradé doit être considéré, le système doit rester opérationnel malgré des fonctionnalités et des capacités réduites.

Middlewares

La communication par message entre les entités du système distribué implique une latence dans les échanges entre processus, pouvant parfois entrainer des échecs lors de l'envoi des messages. Les processus communicants dans un réseau sont appelés des noeuds et communiquent avec plusieurs paradigmes de communication : la communication interprocess (*message passing, multicast, socket*), l'invocation à distance (*remote invocation*), la communication directe, le *request/reply*, le *publish/subscribe*, la file d'attente de message et la mémoire partagée distribuée. Les middlewares servent à masquer l'hétérogénéité du réseau.

Definition 6 (Middleware) Un middleware est une couche logicielle positionnée audessus du système d'exploitation mais en dessous du programme d'application. Il offre une abstraction de programmation commune à travers un système distribué [Bak01]. Les middlewares gèrent la complexité et l'hétérogéneité inhérentes aux systèmes distribués. Les MOM (Message Oriented Middlewares) sont des exemples de middlewares qui facilitent

la communication entre des applications distribuées en utilisant un modèle de messagerie asynchrone.

Différentes catégories de middleware existent : les objets distribués (CORBA, Java RMI), les composants distribués (OpenCOM, JBoss), les systèmes *publish/subscribe* (Scribe, JMS), les files d'attente de message (Websphere MQ, JMS), les services Web (Apache Axis) et les systèmes *peer-to-peer* (Gnutella, Pastry). Durant nos travaux, nous avons utilisé le middleware RabbitMQ qui se situe dans la catégorie des files d'attente (*message queuing*), il permet également d'utiliser le paradigme *publish/subscribe*.

Exemple concret de système distribué : Un MMOG

Un MMOG (Massively Multiplayer Online Games) est un jeu vidéo en ligne caractérisé par son nombre important de joueurs se connectant à un serveur de jeu en simultané. Dans les MMOGs, les joueurs doivent coopérer et communiquer, ce qui représente un challenge en raison du nombre de joueurs qui utilisent des outils sociaux et qui influent sur le système économique du jeu simultanément. Il y a un besoin de réponse rapide pour garder l'expérience des utilisateurs du jeu. Le monde partagé entre les joueurs doit rester cohérent, c'est-à-dire que les données du jeu doivent être identiques sur tous les nœuds (donc pour tous les joueurs). En effet, dans les MMOGs, quand les joueurs coopèrent pour combattre un monstre puissant aussi appelé « *boss* », la vision de celui-ci doit être la même pour chacun des joueurs. Lorsque le « *boss* » lance une attaque de groupe, chaque joueur doit pouvoir réagir en conséquence, et si jamais un nœud est incohérent, tous les joueurs ne verront pas la même chose, pouvant mener à la mort du personnage et donc à une dégradation de l'expérience du joueur. Par exemple, l'architecture de EVE Online repose sur une architecture client/serveur centralisée où chaque joueur fait appel à la base de données originale pour obtenir des informations sur le monde. La cohérence est assurée par l'unique copie de l'état du monde localisée sur un serveur, mais demande au serveur de nombreux calculs pour satisfaire la charge. D'autres types d'architectures existent, plus distribuées, notamment avec des serveurs géographiquement distribués qui permettent aux utilisateurs de se connecter là où la latence est la plus faible, typiquement le plus proche serveur géographique (mais ce n'est pas toujours le cas). Le type d'architecture qu'utilise EVE Online est extensible par l'ajout de serveurs.

Problème de synchronisation

L'interaction entre les processus dans les systèmes distribués résulte dans le besoin de communication et de coordination, c'est-à-dire que les processus doivent échanger de l'information et que les activités ou tâches doivent être synchronisées ou ordonnées dans leur exécution. La latence des communications doit être prise en considération, car c'est ce délai qui impactera et limitera la coordination des processus. Une synchronisation temporelle entre les processus permet également au système de prendre les décisions dans le bon ordre. Le temps de transmission d'un message comprend la latence entre les deux noeuds et la taille du paquet transmis en fonction du débit disponible.

Coordination des processus

Des processus distribués doivent souvent coordonner leurs activités, surtout lorsqu'ils partagent une même ressource. Dans ce cas, l'exclusion mutuelle permet d'éviter les interférences et de garder un système cohérent [RB86]. Dans notre contexte, si un radar partage un fichier d'une piste puis décide de le modifier pendant qu'un autre processus utilise cette donnée en entrée de ses algorithmes, alors la piste et les calculs effectués ne seront plus cohérents. Le fichier d'une piste doit être bloqué pour un seul utilisateur à la fois.

Des algorithmes d'exclusion mutuelle existent pour éviter que des ressources partagées ne soient utilisées simultanément (Lamport, Raymond, Naimi-Tréhel [Lej14]). Nous considérons un système asynchrone composé de plusieurs processus sans échec qui utilisent des ressources communes et qui délivrent des messages uniques et sans erreur. Un algorithme d'exclusion mutuelle gère l'accès à des sections critiques qui sont des zones du programme pouvant être accédées par les différents processus. L'entrée à une section critique est bloquée lorsqu'un autre processus y est déjà, il accède à la ressource puis quitte la section critique quand il a terminé. Cependant, ce modèle pourrait provoquer des interblocages ou des famines, c'est-à-dire que les processus pourraient être bloqués dans leur tentative d'accès à la ressource, ou qu'ils pourraient être privés continuellement des ressources nécessaires à leur fonctionnement. Afin d'éviter de telles situations, les algorithmes d'exclusion mutuelle sont conçus et mis en œuvre avec des mécanismes intégrés pour détecter et prévenir les interblocages et les famines.

D'une part, la famine survient lorsqu'un processus n'est jamais en mesure d'obtenir les ressources nécessaires pour terminer ses tâches en cours. Cela peut se produire si

l'algorithme de partage de ressource n'est pas équitable.

D'autre part, l'interblocage se produit lorsque deux processus, A et B, demandent simultanément des ressources R1 et R2 pour accomplir un service. Si le processus A demande d'abord la ressource R1 puis la ressource R2, pendant que le processus B fait l'inverse, une situation d'interblocage peut survenir. Chaque processus détient une ressource nécessaire à l'autre processus pour poursuivre son exécution. Bien que les processus puissent attendre la libération des ressources demandées, ou bien annuler leurs requêtes si les ressources sont déjà utilisées, ils risquent de continuer à se bloquer mutuellement s'ils redemandent les ressources simultanément. Des méthodes pour éviter ces problèmes existent [Sin89], et imposent par exemple un ordre d'acquisition des ressources. Ainsi, un processus devra obligatoirement récupérer la ressource R1 avant la ressource R2. Ce n'est pas toujours une solution optimale, car elle ne prend pas en considération les latences qu'il pourrait y avoir dans le réseau.

Partage d'un état global

En l'absence d'horloges communes et de communications instantanées, l'observation simultanée de l'état local de chaque processus afin d'obtenir un état global est impossible dans un système distribué [HMR93]. L'état local d'un composant ou d'un noeud dans le système fait référence aux informations observées et perçues par ce composant et lui seul. Les interactions amènent l'état local d'un composant à être modifié, en général pour se rapprocher le plus possible de l'état global qui est perçu par l'ensemble du système. L'état local est crucial pour la prise de décision autonome de chaque composant.

Dans un système distribué, la gestion et la mise à jour efficaces de l'état local des composants sont essentielles pour assurer la cohérence, la fiabilité et les performances du système dans son ensemble. Dans notre contexte, les plateformes ont des connaissances sur les ressources disponibles des autres plateformes à un instant T. Le partage de ces informations passe par un canal dont le temps de transfert varie et n'est pas connu, ce qui fait que l'état réel de la ressource peut avoir été modifié entre le moment de l'envoi du message et la mise à jour de l'état local. Le processus en charge d'allouer des tâches à des senseurs doit avoir une connaissance de l'état global proche de la réalité pour éviter de prendre des décisions incohérentes. Des mécanismes appropriés de synchronisation, de communication et de coordination peuvent garantir la cohérence des états locaux avec l'état global du système.

Conclusions

Ce chapitre a permis d'exposer le contexte et le cadre d'application de nos recherches. Dans un premier temps, nous avons présenté des plateformes aériennes et navales, et les senseurs qui peuvent être intégrés à leur bord. Ces senseurs sont les ressources essentielles qui permettront la collaboration multiplateforme et l'allocation de tâches présentées dans le manuscrit. Dans un second temps, nous avons présenté les particularités et les contraintes de l'environnement opérationnel des plateformes navales et aériennes, en mettant en lumière les diverses menaces auxquelles elles sont confrontées. Nous avons également examiné le rôle de chaque équipement et navire dans la gestion de ces menaces, en tenant compte de leurs caractéristiques individuelles, ce qui met en évidence un besoin stratégique de collaboration. Ensuite, nous avons exposé l'état actuel des recherches sur les systèmes et l'intérêt des armées de gagner en préavis par la proposition d'une nouvelle architecture mettant en avant le combat collaboratif naval. Nous avons soulevé le besoin d'utiliser des technologies à base d'intelligence artificielle pour accélérer les processus de prise de décision, tout en tenant compte des limitations des réseaux en termes d'échange de données.

Le contexte d'application de nos recherches impose de nombreuses contraintes pour la collaboration entre les systèmes de senseurs. La diversité des plateformes, de leurs missions, et de leurs senseurs introduit une hétérogénéité qui nécessite une communication efficace au sein d'une architecture distribuée. Cette architecture doit être conçue pour répondre aux besoins et aux exigences des utilisateurs du système, tout en proposant une prise de décision rapide grâce à une allocation dynamique des tâches aux senseurs. De plus, il est crucial de prendre en compte la généricité et la mise à l'échelle de la solution. En considérant l'aspect multi-plateforme de la problématique, nous sommes confrontés aux défis inhérents aux systèmes distribués, aux communications inter-systèmes et à la gestion de la concurrence pour l'accès aux ressources partagées.

Chapitre 2

Etat de l'art

Dans ce chapitre, nous allons présenter un état de l'art des recherches sur les architectures dans le domaine militaire. Nous proposerons de détailler l'orientation des recherches américaines dans l'élaboration d'une architecture dans le cadre du combat collaboratif. Ensuite, nous présenterons des types d'architectures utilisées dans des projets militaires, à savoir les architectures ouvertes et les architectures orientées services (SOA). Enfin, nous présenterons un état de l'art sur les systèmes multi-agents (SMA), notamment leurs utilisations pour la problématique d'allocation de tâches à des agents.

2.1 Architectures du domaine militaire

2.1.1 Combat collaboratif américain

L'armée américaine est préoccupée par son développement technologique militaire face à ses besoins stratégiques. Elle doit à tout prix garder sa suprématie et l'a montré en développant le *Joint All Domain Command and Control* (JADC2). Ce sont de nouveaux concepts de C2 (*Command and Control*) pour la guerre multi-milieu qui visent à contrer la stratégie « déni d'accès et interdiction de zones » (A2AD) [MSK+22]. La proposition d'offrir une capacité offensive à un maximum de navire permet d'augmenter la force de dissuasion. La stratégie « A2AD », quant à elle, vise à empêcher l'adversaire de rentrer dans une zone et de manœuvrer. Des îlots artificiels militarisés sont positionnés avec des systèmes antiaériens et antinavires afin de contrôler la zone maritime. D'autres armes peuvent être utilisées dans le domaine naval, par exemple des mines dans la zone désignée comme non accessible, ou de l'artillerie au niveau des côtes pour empêcher l'approche d'un port. JADC2 a pour but de fournir un avantage par rapport aux adversaires grâce à une prise de décision plus rapide et mieux informée. Pour accomplir cela, dans un environnement où le nombre de capteurs et la quantité de données échangées augmentent, des capacités de fusion et de traitement des données sont nécessaires. Le rapprochement

des réseaux tactiques et stratégiques permettra d'accélérer la fusion et le transport de données entre les domaines opérationnels. La résilience du C2 sera améliorée et l'allocation de tâches dans les systèmes se fera de manière dynamique. Ces concepts développés dans le cadre du JADC2 soulèvent l'intérêt de la COP (*Common Operational Picture*) qui est un état global de la situation tactique partagé entre les commandements. La mise en œuvre de ces concepts introduit les réseaux et les systèmes distribués, notamment les problèmes de latences et de bandes passantes qui sont des limitations pour l'échange des volumes de données dans l'élaboration de la COP.

L'ABMS (*Advanced Battle Management System*) est un exemple de projet JADC2 Américain qui utilise l'IoMT (*Internet of Military Things*). Ce système a pour but de rendre la donnée générée disponible à n'importe quel endroit, le traitement de la donnée et les actions associées sont effectués rapidement. Les Américains utilisent l'environnement cloud, tout comme les Français, ainsi que les nouvelles méthodes de communications pour partager les données en toute transparence avec l'utilisateur en utilisant l'IA et ainsi améliorer le temps de prise de décisions [ABM22]. En septembre 2020, l'ABMS a effectué des démonstrations et réussi à détecter puis détruire un missile de croisière simulé lancé en direction des États-Unis en utilisant une défense hypervéloce.

Le concept de JADC2 est donc large dans sa définition et propose une solution pour le combat collaboratif en connectant des systèmes d'armes et de senseurs entre eux, plus rapidement et avec plus de résilience que ce qui est fait actuellement. Les notions d'interconnexion des systèmes et de résilience découlent de la nécessité de garantir une communication fluide et continue entre les différentes composantes des forces armées, ainsi que d'assurer une capacité d'adaptation rapide face aux perturbations et aux menaces adverses. De nombreux pays dirigent des recherches qui vont dans le même sens, les solutions étudiées sont variées en raison de l'aspect confidentiel qui restreint le partage des connaissances classifiées.

2.1.2 *Cooperative Engagement Capability* (CEC) [Hop95]

Le but de l'approche du CEC est de rendre la flotte de combat plus réactive face aux différentes menaces de l'environnement où elle évolue. Les systèmes de combats partagent les données de senseurs associés aux pistes rapidement et avec précision afin que l'ensemble des navires, avions et unités au sol, présents sur le théâtre d'opérations, puissent agir comme une seule et même entité. Le théâtre complexe de la défense aérienne comprend l'environnement naturel et ses effets sur les senseurs (météo, reliefs, obstacles, etc.), mais

aussi des vols commerciaux, des bateaux de plaisances et de pêches dont la classification entre alliés et ennemis est difficile. CEC tente de prendre parti de chaque navire de guerre localisé sur le théâtre possédant des senseurs et armes divers. Cette approche requiert un partage des données de tous les senseurs pour que chaque plateforme de combat ait accès aux mêmes informations.

Le CEC permet l'échange de données radar entre toutes les plateformes, les situations tactiques des plateformes sont donc identiques. Il permet aussi l'engagement coopératif entre les plateformes, une plateforme peut engager la cible sans avoir obtenu de détection de pistes d'une cible avec ses propres radars, augmentant de ce fait sa portée d'action. Le CEC est en interface avec le CMS de la plateforme, il est par exemple intégré sur les systèmes de combat de classes AEGIS et LHA (Destroyer et porte-hélicoptère).

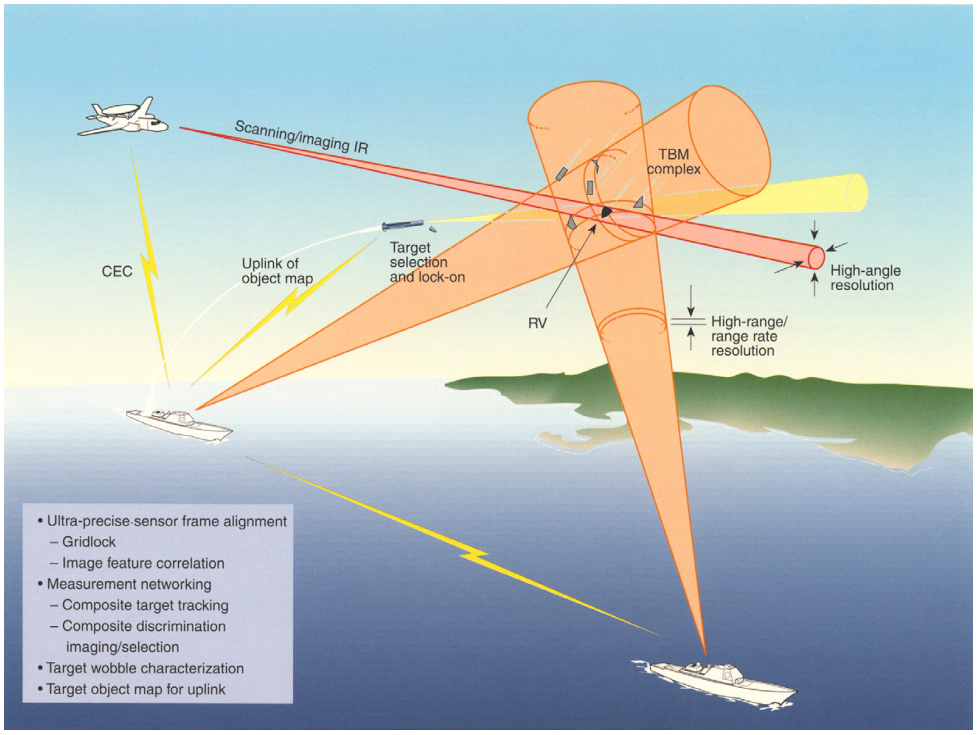

FIGURE 2.1 – Illustration de l'amélioration de la résolution d'un missile balistique gâce à l'approche du CEC [Hop95].

Dans la figure 2.1, le CEC montre le potentiel d'interception d'un missile balistique. L'ensemble des senseurs des différentes plateformes permet de maintenir une unique piste de meilleure qualité. L'amélioration de la précision de localisation de la menace, avec une fréquence de rafraîchissement élevée, permet de caractériser précisément le mouvement

d'une cible et de créer une cartographie 3D autour de la menace. Le système partage également en temps réel le statut d'engagement de la cible et des recommandations sur le choix de l'unité qui a la plus grande probabilité de détruire le missile ennemi.

2.1.3 MOSAIC Warfare [DPSG19]

Le concept de MOSAIC Warfare de la DARPA (*Defense Advanced Research Projects Agency*) propose un réseau de senseurs à bas coût rapidement configurable, avec des noeuds multi-domaine C2 et des systèmes coopératifs. Ce concept vise à répondre aux situations de crise où l'incertitude règne, en imposant à l'ennemi une complexité accrue dans ses prises de décisions. Cette complexité peut être atteinte en distribuant les senseurs et les systèmes d'armes plutôt que de les rassembler sur une seule plateforme, de manière dynamique et rapide de telle sorte que l'ennemi ne puisse pas s'adapter aux nouvelles configurations possibles. Contrairement à une approche traditionnelle où les forces militaires utilisent des plateformes spécialisées pour des missions spécifiques, le *MOSAIC Warfare* cherche à intégrer un large éventail de capacités militaires pour former un ensemble interconnecté et agile.

La « mosaïque » est un art décoratif qui assemble des tuiles pour former une figure ou un motif. Par analogie, une tuile peut représenter un senseur, une arme, un système ou un autre élément qui peut être réarranger avec d'autres tuiles pour former un motif pouvant correspondre au système opérationnel. Le système peut modifier sa configuration en déplaçant les tuiles sur la mosaïque. La coordination des systèmes est possible grâce aux avancées scientifiques dans les réseaux, les liaisons de données, l'IA et le *machine learning*. MOSAIC crée ainsi un réseau composé de nombreux éléments hétérogènes, avec des fonctions et des capacitités qui peuvent collaborer de façon dynamique au cours du temps. Toutes les plateformes sont décomposées en leurs plus petites fonctions opérationnelles, créant ainsi des nœuds collaboratifs au sein d'un réseau global.

La figure 2.2 présente les communications dans le cadre de MOSAIC où chaque acteur du théâtre d'opérations peut être à la fois « producteur » ou « consommateur » de données. La figure montre une configuration possible du réseau avec des possibilités de d'adaptation. Avec ces concepts, la perte d'une plateforme ne mettrait plus en péril la mission, le système pourrait se reconfigurer et la multitude de combinaisons empêcherait l'adversaire de connaître exactement le fonctionnement de l'architecture, ce qui assurerait une forte résilience du système. Ces études ne sont encore que préliminaires et de nombreuses problématiques émergent : la standardisation des interfaces, le besoin en dé-

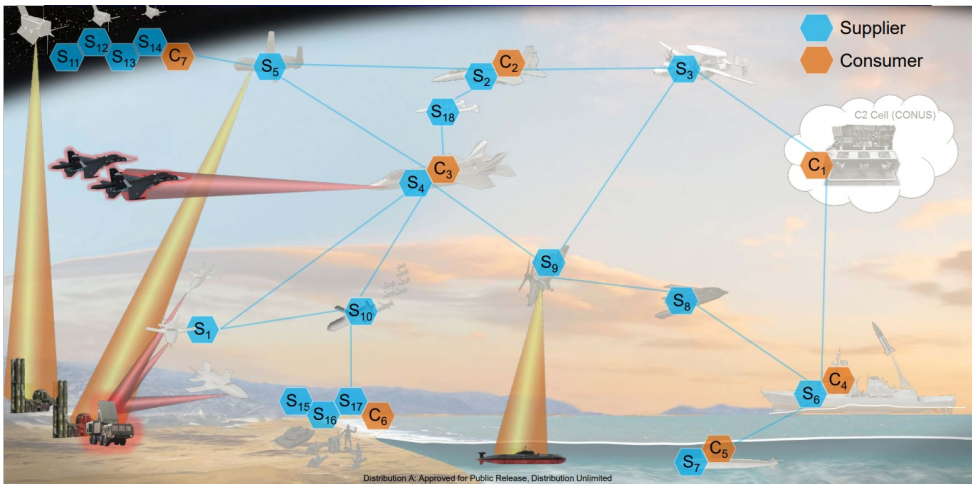

FIGURE 2.2 – Présentation des producteurs et consommateurs de données dans le cadre de MOSAIC [Gra18].

bit pour les nombreux échanges générés, la compréhension entre les acteurs qui sont des systèmes parfois très différents, l'optimisation de l'utilisation des ressources, etc.

2.2 Les architectures ouvertes

L'architecture de systèmes ouverte comprend de nombreux modules pouvant être assemblés ou composés grâce à des interfaces standardisées et bien définies, favorisant ainsi l'intégration de nouvelles fonctionnalités ou de systèmes tiers de manière transparente et efficace [Vog13]. L'analogie d'un jeu de Lego permet de présenter le concept des architectures ouvertes, les pièces peuvent être imbriquées les unes dans les autres selon des spécificités propres à chacune d'entre elles. Les avantages de ces architectures sont multiples. Dans un premier temps, les entreprises expertes dans une partie du système, mais dans l'incapacité de développer le système dans son entièreté peuvent développer des équipements standardisés à intégrer. La compétitivité apporte du choix et de la qualité dans ces modules qui sont alors portables, reconfigurables, maintenables, et de nouvelles technologies peuvent y être intégrées grâce à des mises à jour des fabricants. En outre, ce modèle architectural présente divers avantages, tels que l'interopérabilité, la réutilisabilité et la mise à l'échelle, ce qui le rend facilement interopérable grâce à l'utilisation de normes ouvertes bien établies au sein de la communauté. Cependant, toutes ces propriétés peuvent apporter un coût supplémentaire non négligeable dans la conception des modules.

L'architecture « *Modular Open System Architecture* » (MOSA) [Hen09] suit une approche d'architecture ouverte avec une conception modulaire, des interfaces définies et ouvertes, des standards connus et de la donnée accessible. Ces bénéfices se manifestent de différentes manières. La compétitivité est favorisée grâce aux nombreux modules à développer. De plus, la modification d'un composant du système n'implique pas nécessairement la redéfinition de l'ensemble. La configuration et la reconfiguration des modules existants sont simplifiées lors de modifications ultérieures. La réutilisation des modules permet d'économiser, mais nécessite des coûts de conception plus élevés. Enfin, les modules logiciels et matériels peuvent être remplacés indépendamment. Des standards américains de l'*United State Air Force* (USAF) sont décrits dans [Tok17], ce sont des architectures basées sur les concepts d'architectures ouvertes, et plus particulièrement MOSA. Ces architectures sont présentées dans les sous-sections suivantes.

2.2.1 *Open Mission System* (OMS)

Un des objectifs d'OMS, comme bien d'autres architectures ouvertes, est d'identifier les nouvelles approches architecturales afin de réduire les coûts de développement. L'architecture utilise des concepts existants et employés dans des systèmes vendus à un large public. L'architecture est définie par des interfaces entre les sous-systèmes (charges et senseurs) et des services connectés à un bus ASB (*Avionics Service Bus*). Ce bus utilise des protocoles spécifiques permettant d'interconnecter les différents sous-systèmes avioniques, comme les systèmes de navigation ou de vol. Les fonctionnalités du système sont caractérisées par un ensemble de services et de clients qui entrent et quittent le système par l'ASB. Le bus prend en charge des capacités de communications ainsi qu'un mécanisme d'enregistrement des interfaces afin de permettre au client de faire appel aux services. Les composants insérés dans le système doivent se décrire auprès du système afin d'échanger des données avec les autres composants. L'architecture OMS se sert des concepts de services, d'interfaces et de modules. Chaque composant s'insère dans le système via le bus ASB. En se décrivant lui-même, le composant fournit aux autres composants les informations nécessaires pour interagir avec lui de manière standardisée. Cette description détaillée favorise l'interopérabilité entre les sous-systèmes en permettant une communication cohérente et harmonieuse entre les différents composants.

2.2.2 *Future Airbornes Capability Environment* (FACE)

L'architecture FACE a été conçue pour promouvoir l'interopérabilité, la portabilité et la modularité au sein de l'industrie de la défense. Une explication détaillée du standard FACE est fournie dans un article portant sur l'utilisation de *middlewares* dans un système partitionné distribué [GVDPTL18]. Ce standard définit un environnement de développement, des guides de conception et des interfaces conçues pour un déploiement de composants portables dans le domaine de l'avionique [BDS]. Son objectif principal est d'établir des points d'interaction entre différentes technologies afin de faciliter leur intégration dans le système. Dans l'architecture FACE, les interfaces sont regroupées en deux catégories principales : les interfaces verticales et les interfaces horizontales. Les interfaces verticales sont conçues pour soutenir une couche logicielle spécifique, facilitant ainsi la communication et l'interaction entre les différentes couches du système. Les interfaces horizontales permettent de connecter les applications entre elles ou avec des sources de données externes. L'architecture de référence est organisée en plusieurs segments distincts :

- Le « *Portable Components Segment* » (PCS) regroupe les applications ou composants FACE eux-mêmes.
- Le « *Transport Services Segment* » (TSS) utilise des technologies *middlewares* pour assurer l'interopérabilité du système.
- Le « *Platform-Specific Services Segment* » (PSSS) englobe les services spécifiques à la plateforme et aux appareils.
- Le « I/O *Services Segment* » (IOSS) constitue la couche de bas niveau, comprenant des adaptateurs et des pilotes pour accéder aux éléments périphériques tels que le réseau et le matériel.

Cette architecture permet une modularité accrue, ce qui signifie qu'un changement dans l'un des segments aura un impact minimal sur l'ensemble de l'architecture. Cela se traduit par une réduction des coûts d'intégration de nouvelles capacités et une plus grande flexibilité dans l'évolution du système. De plus, l'architecture FACE prévoit un interfaçage avec divers matériels (Ethernet, MIL-STD ou autre) ainsi qu'avec les équipements (capteurs, écrans, radios, transports, etc.), assurant ainsi une intégration fluide avec les environnements matériels et logiciels existants. La figure 2.3 offre une vue de l'architecture et de ses segments.

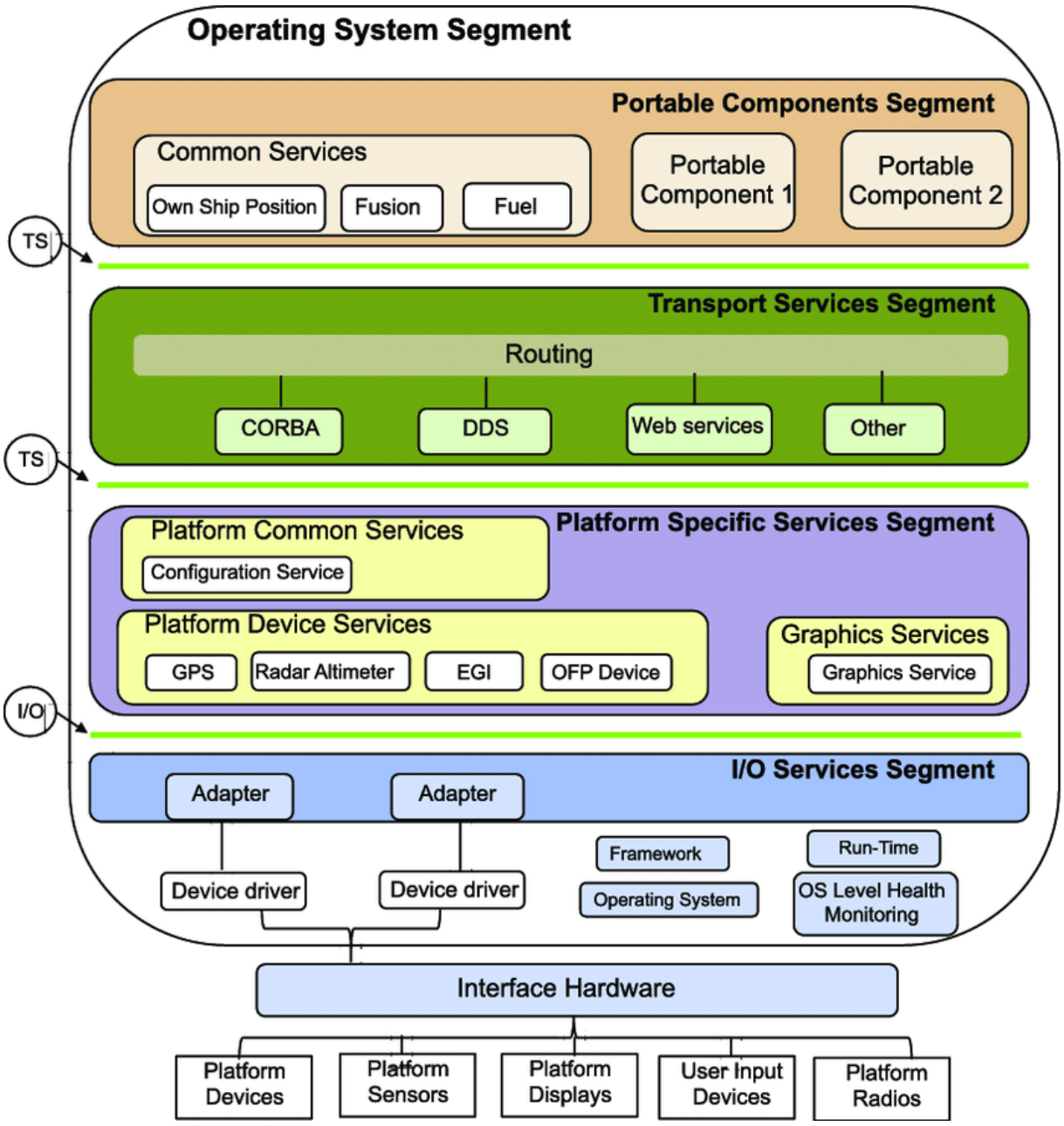

FIGURE 2.3 – Architecture FACE [GVDPTL18].

2.2.3 *Sensor Open System Architecture* (SOSA) [SCL19]

Le groupe SOSA est une initiative qui regroupe les besoins architecturaux de 3 départements militaires américains : l'Armée de Terre, l'Armée de l'Air et la Marine. Les travaux de SOSA s'inspirent des architectures MOSA existantes, telles que FACE et OMS,

dans le but de développer une nouvelle architecture intégrant les aspects logiciels et matériels, la gestion des systèmes et des senseurs, ainsi que les aspects de sécurité et de communication entre modules. Les fondements de l'architecture sont : une architecture de référence du système ouverte, une standardisation des modules (logiciels, matériels et électro-mécanique), l'adoption de standards ouverts existants ou émergents, une efficacité des coûts et une adaptabilité rapide aux évolutions souhaitées par les utilisateurs.

Le consortium SOSA collabore avec le gouvernement et les partenaires industriels pour façonner la prochaine génération de senseurs. Ces senseurs seront dotés d'interfaces permettant l'intégration avec les composants logiciels, physiques et mécaniques/électriques. Les objectifs et points forts de l'architecture SOSA sont multiples :

— **Interopérable** : capable de fonctionner et d'opérer avec d'autres systèmes.

— **Sécurisé** : avec une surface d'attaque réduite et des mécanismes d'authentification renforcés.

— **Modulaire** : composée de modules standardisés et testables individuellement.

— **Compatible** : avec des versions antérieures du standard.

— **Portable** : au niveau des logiciels et matériels pour les utiliser dans d'autres environnements SOSA.

— « ***Plug-and-play*** » : capable de reconnaître l'insertion et le remplacement de modules et de comprendre leurs services et capacités.

— **Évolutive** : permettant la mise à jour des modules et leur intégration.

— **Multi-senseurs** : capable de gérer une multitude de senseurs.

— **Hétérogène** : fonctionnant sur des porteurs de différentes tailles.

— **Résiliente** : capable de maintenir ses opérations en cas de dommages physiques, d'interférences électromagnétiques ou d'attaques cybernétiques.

La figure 2.4 présente l'architecture SOSA du senseur, décomposée en 25 modules. Les modules, identifiés par différentes couleurs, couvrent un ensemble de fonctions : la gestion du senseur, la collecte des données, le traitement des signaux et des pistes, et l'analyse et l'exploitation des données. Le module en gris clair achemine la donnée analysée sous forme de rapport, et les modules gris foncé gèrent les opérations de support du système. Le but de SOSA est de permettre de remplacer n'importe quel module présenté sur la figure par un module développé par un concurrent par exemple, favorisant ainsi l'évolution des logiciels et du système dans un environnement concurrentiel.

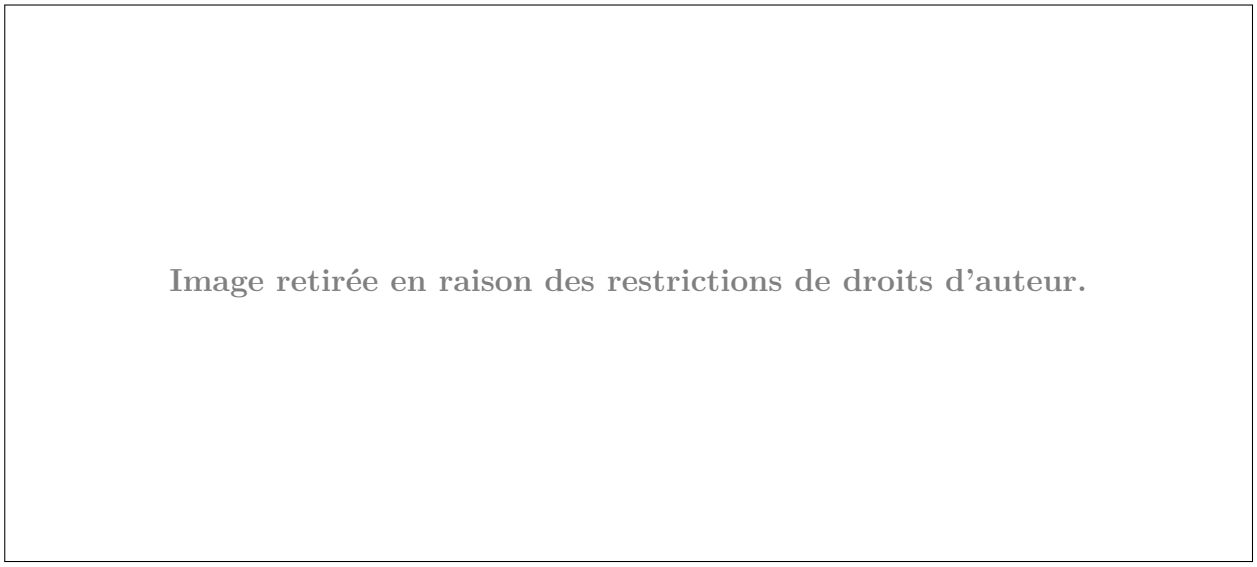

FIGURE 2.4 – Modules d'une architecture SOSA.

2.3 Les architectures orientées services

De nombreuses définitions de l'architecture orientée services existent, notamment selon le domaine d'application de celle-ci. Dans un contexte militaire, l'article [SHR+08] en propose une :

Definition 7 (Architecture orientée services) L'architecture orientée services (OSA, SOA) est un style d'architecture qui met à disposition des ressources disponibles de telle sorte qu'elles puissent être trouvées et utilisées par d'autres parties qui n'ont pas besoin de les connaître en avance.

Un service est décrit formellement à l'aide d'une interface, par exemple en utilisant le langage WSDL (*Web Services Description Language*) dans le cadre des services Web. Le langage WSDL décrit le protocole de communication, le format des messages requis par le service, les fonctions disponibles du service et la localisation du service. L'utilisateur du service est donc indépendant de l'implémentation du service, le service est alors modifiable sans impact sur l'utilisateur tant que l'interface reste inchangée, ce qui crée un fort découplage. Un utilisateur n'a donc pas besoin de connaître le fonctionnement d'un service, et plusieurs services pourraient répondre à un même besoin formulé par l'utilisateur, ce qui permet de choisir le meilleur service ou d'avoir des alternatives.

Dans les domaines complexes, civils ou militaires, l'architecture orientée services offre une approche flexible et modulaire pour la conception de systèmes complexes. Elle permet une intégration efficace des applications et des services, favorisant ainsi l'interopérabilité entre les différents composants d'un système. Cette approche permet également de réutiliser les services existants, réduisant ainsi les coûts de développement et de maintenance des systèmes. En outre, l'architecture orientée services facilite l'évolution et la mise à jour des systèmes, en permettant le remplacement ou la modification des services individuels sans perturber le fonctionnement global du système.

2.3.1 Un standard OTAN : CESMO (*Cooperation Electronic Support Measures Operations*)

L'OTAN en tant qu'organisation collective entre plusieurs pays ne dispose pas de force militaire propre, elle dépend des armées des différentes nations dont elle est composée. La problématique est donc de pouvoir gérer et coordonner les capacités de l'ensemble des nations lors des opérations. Dans ce contexte, chaque capacité joue un rôle clé et peut être vue comme un service. En orchestrant les services, l'ensemble de la force alliée trouvera de nouvelles capacités qui n'existeraient pas sans collaboration. Les services, selon l'architecture SOA proposée ici, ne sont pas de simples services web, dans le cadre de CESMO, le service de géolocalisation en est un exemple. La planification des services, c'est-à-dire le choix de l'activation de chaque service pendant un intervalle de temps, est importante. Pour qu'elle puisse se faire, il faut avoir connaissance de ce que fait le service, de sa disponibilité, de la possible composition de services (un service plus général qui fait appel à d'autres services pour fonctionner), de l'ordre d'appel des services (un service qui génère une image ne sera appelé qu'après un service de balayage de zone), ainsi que de la capacité du service à se décrire lui-même, ce qui permet aux utilisateurs de savoir comment l'utiliser de manière efficace. Par exemple, des services comme l'analyse d'images ou la poursuite de cibles peuvent être disponibles.

Le standard OTAN CESMO (*Cooperation Electronic Support Measures Operations*) [Wit20] permet le recueil de données RF en temps réel pendant les opérations afin d'aider à l'évitement rapide de menaces ou à l'engagement. Une nouvelle liaison de données n'est pas nécessaire au partage de l'information CESMO, car n'importe quelle liaison utilisant IP peut gérer du trafic CESMO. CESMO utilise la triangulation et la TDOA pour localiser les émetteurs ennemis, que ce soit des radars ou des appareils de communica-

tions. CESMO récupère la liste des plateformes participantes avec leurs senseurs correspondants, notamment leurs RWRs (Radar Warning Receivers) et leur ESMs (Electronic Support Measures). Une liste d'émetteurs hostiles est fournie grâce à une récupération intelligente durant les missions. Le besoin de CESMO découle du fait que les actions des différents porteurs de guerre électronique n'exploitent pas pleinement la présence des alliés dans une même zone d'opérations, en raison du manque de standardisation et de la propriété industrielle sur les interfaces des plateformes. Cela conduit soit à une intégration de l'homme dans le processus de collecte des informations de surveillance électronique, soit à l'adaptation des systèmes à des interfaces personnalisées pour prendre en charge les formats propriétaires des messages. Pour résoudre ces problèmes, des normes sont mises en place, des formats de messages interopérables sont développés et des pratiques de travail communes avec les alliés sont partagées.

2.3.2 CAMELOT

CAMELOT (*C2 Advanced Multi-domain Environment and Live Observation Technologies*) est un projet européen ambitieux qui vise à concevoir et à mettre en œuvre une architecture distribuée innovante reposant sur des services [ASD18]. Son objectif central est de développer un système de surveillance et de contrôle des frontières européennes, spécifiquement conçu pour détecter et contrer les activités telles que l'immigration illégale et le trafic de stupéfiants. En combinant des technologies de pointe et des approches collaboratives, le projet CAMELOT cherche à offrir une solution intégrée et efficace pour renforcer la sécurité des frontières de l'Europe.

Les travaux d'architecture de CAMELOT [UPT+18] mettent en évidence les architectures existantes, comprenant une dizaine de références parmi lesquelles figure FACE, et les critères qui ont conduit au choix de certaines technologies. Le système vise à assurer le contrôle des UxVs (Véhicules sans pilotes), ainsi que des capteurs pour fournir des services complexes. La standardisation permet d'intégrer facilement de nouveaux modules ou services. L'architecture utilise un *middleware* et le paradigme *publish/subscribe* pour les interactions entre les modules, services, équipements ou tâches. Des critères supplémentaires impactent la validité d'une architecture : la facilité d'utilisation, l'interopérabilité, la réutilisation d'anciens composants, la modularité sur les fonctionnalités proposées, la maintenance des composants, l'extensibilité de la solution et l'utilisation de standards ouverts. La figure 2.5 présente l'architecture finale proposée pour CAMELOT.

La documentation sur l'architecture établit une liste d'équipements à intégrer. Chaque

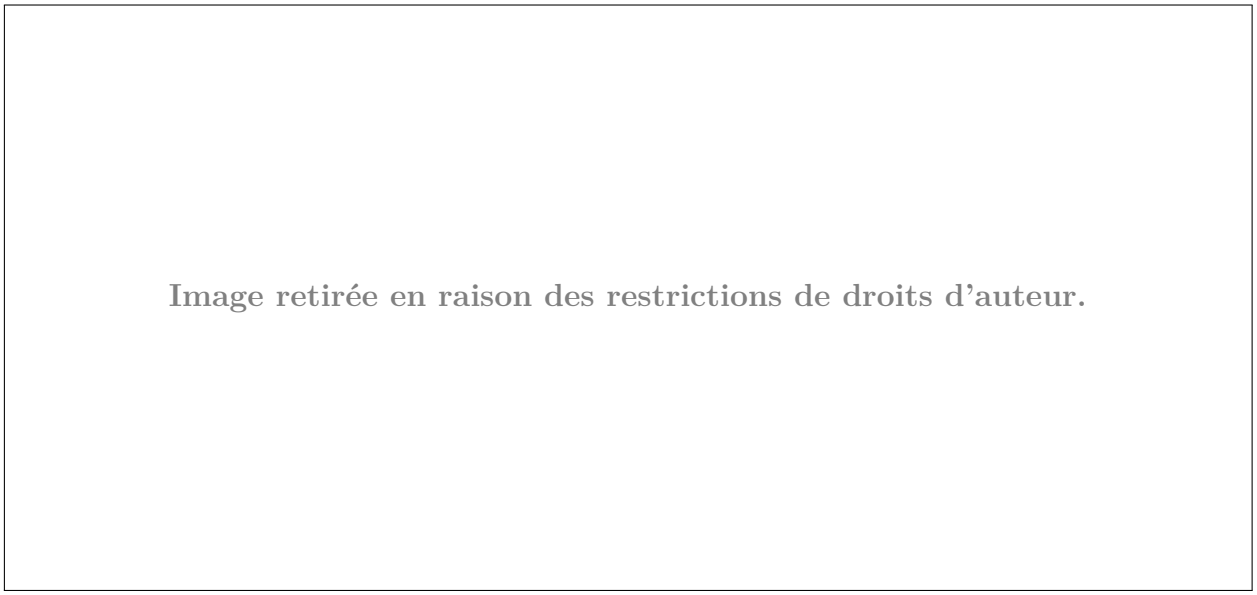

FIGURE 2.5 – Achitecture CAMELOT, [ASD18].

équipement décrit ses fonctionnalités, ses interfaces avec le système, ses entrées, ses sorties. Par exemple, une caméra thermique permet d'observer les menaces en conditions de noir total et de mauvaise météo. La documentation décrit les performances d'imagerie, les spécifications de l'équipement, les interfaces (TCP/IP, RS-422), les portées de détection, de reconnaissance et d'identification. Les équipements peuvent également être des drones, des radars ou des logiciels.

CAMELOT suit les concepts des architectures orientées services (SOA). Les fonctionnalités, les équipements et les applications de CAMELOT sont enveloppés et exposés comme des services Web. Le *pattern publish/subscribe* délivre des messages au format XML acheminés par un bus ESB (*Enterprise Service Bus*) entre des producteurs et des consommateurs de services. Le bus ESB permet la description et la découverte de ces services, qui sont des composants logiciels faiblement couplés. Le bus peut être considéré comme un *backbone* dans lequel les services peuvent se connecter et se déconnecter facilement sans impacter les autres services et sans nécessiter un redémarrage ou un arrêt des applications en fonctionnement. Les services sont capables d'interagir entre eux grâce à ce bus. Dans CAMELOT, le bus est en charge de trois tâches : la communication, l'intégration et l'interaction entre les services. La figure 2.6 présente le modèle SOA. Le fournisseur de services est une entité du système (un calculateur, un logiciel, un équipement, etc.) qui publie ses services et la façon de les utiliser (description de services) à un registre de

services. Ensuite, le consommateur de service est une entité (logicielle, applicative, etc.) qui recherche un service dans le registre pour des besoins définis. Une fois le service adéquat trouvé, le consommateur va faire une demande d'utilisation du service, le fournisseur acceptera ensuite la requête, selon ses disponibilités.

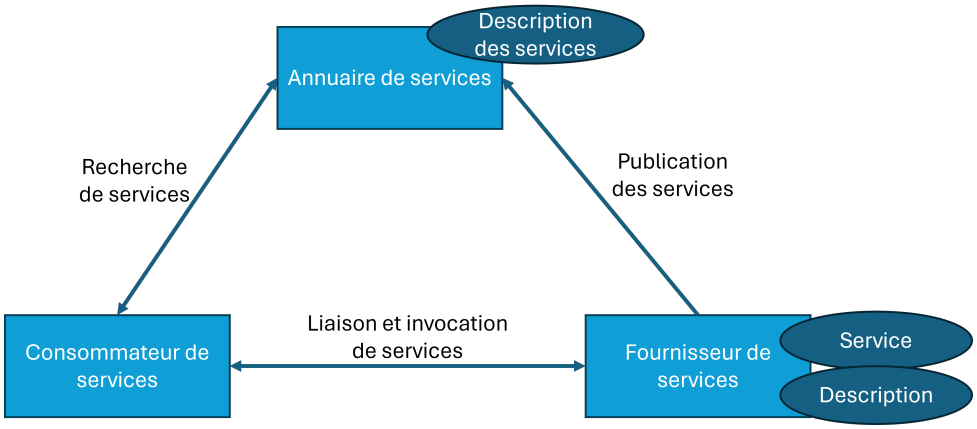

FIGURE 2.6 – Les échanges de services dans une architecture orientée service.

2.3.3 RAMSES II

Les travaux de thèse actuels s'inscrivent dans la lignée de précédentes recherches qui ont exploré des aspects similaires dans le domaine aéronautique. L'objectif de la thèse de Ludovic Grivault, intitulée « Architecture multi-agent pour la conception et l'ordonnancement de systèmes multi-senseur embarqués sur plateformes aéroportées » [Gri18], était de proposer une architecture à base d'agent, pour l'ordonnancement de tâches senseurs sur une plateforme unique.

L'architecture proposée par l'auteur cherche à concilier des besoins opérationnels et industriels dans un contexte aéronautique. Du côté opérationnel, l'autonomie du système est un enjeu majeur, avec le maintien et l'analyse d'une situation tactique. L'opérateur doit pouvoir utiliser des ressources senseurs s'il le demande, la trajectoire et les actions senseurs peuvent être modifiées. La reproductibilité est également un critère important. C'est-à-dire que, dans des conditions d'expérience similaires, les performances, les résultats et les fonctionnalités peuvent être reproduits à l'identique. Au sein de l'industrie, il est essentiel de concevoir l'évolution future de l'architecture de manière à faciliter sa modification, notamment en la rendant plus modulaire. Le retravail doit être minimisé lors des

modifications ou d'un changement sur un senseur dans le système. Les senseurs peuvent être amenés à évoluer (nombre, taille, puissance, fonctionnement), ce qui nécessite une flexibilité quant aux évolutions matérielles et logicielles.

Le partage des ressources au sein du système multi-senseur s'effectue grâce à un ordonnancement qui gère les concurrences d'utilisation des ressources en temps réel. L'ordonnanceur prend en entrée des plans à ordonnancer et délivre en sortie les dates de début des tâches ordonnancées ainsi que les plans non ordonnancés. L'ordonnancement est évalué selon deux critères, le nombre de plans ordonnancés et le temps d'occupation des ressources. Le modèle choisi divise les tâches en parts élémentaires, il implique de fortes complexités au niveau de la mémoire et des calculs, en raison des différences sur la durée des tâches, mais il peut aussi résoudre le problème de micro-ordonnancement des senseurs.

L'agentification est le processus de répartition des agents sur les différentes entités du système. L'agent est une entité physique ou virtuelle, qui agit de façon autonome et proactive, qui est capable de communiquer, d'agir, de percevoir et qui essaie d'accomplir un ou plusieurs objectifs. Il se différencie de l'artéfact qui lui n'est pas autonome (base de données par exemple).

Le développement de l'architecture a été fait avec la méthodologie Arcadia [Voi17] qui comprend plusieurs niveaux d'analyse : l'analyse opérationnelle (besoins utilisateur), l'analyse fonctionnelle (fonctions pour répondre aux besoins), l'analyse logique (fonctions réalisées par le système) et enfin l'analyse matérielle. La première étape de conception de l'architecture consiste à énumérer l'ensemble des composants au contact du système. Dans ce cas, nous avons le gestionnaire de missions, la plateforme aéroportée, les bases de données, les têtes senseur et autres ressources, le *track merger*, le gestionnaire des senseurs, l'ordonnanceur et l'environnement. Ensuite, une réflexion sur les données échangées entre les entités est réalisée, comme par exemple : les requêtes opérationnelles, les données tactiques, les politiques (des contraintes sur le système : restrictives, autonomie, comportement), les connaissances fonctionnelles et opérationnelles, la description des plans (description de l'utilisation des ressources dans le temps), les données fusionnées. La figure 2.7 présente l'architecture finale retenue.

Les travaux de thèse ont donné lieu à un brevet [GHSVE22]. De plus, des études internes à Thales sont en cours pour étendre les concepts proposés à une problématique multiplateforme aéroportée. L'architecture de RAMSES II proposée étant centralisée et monoplateforme, nos travaux de thèse se concentreront sur le développement d'une ar-

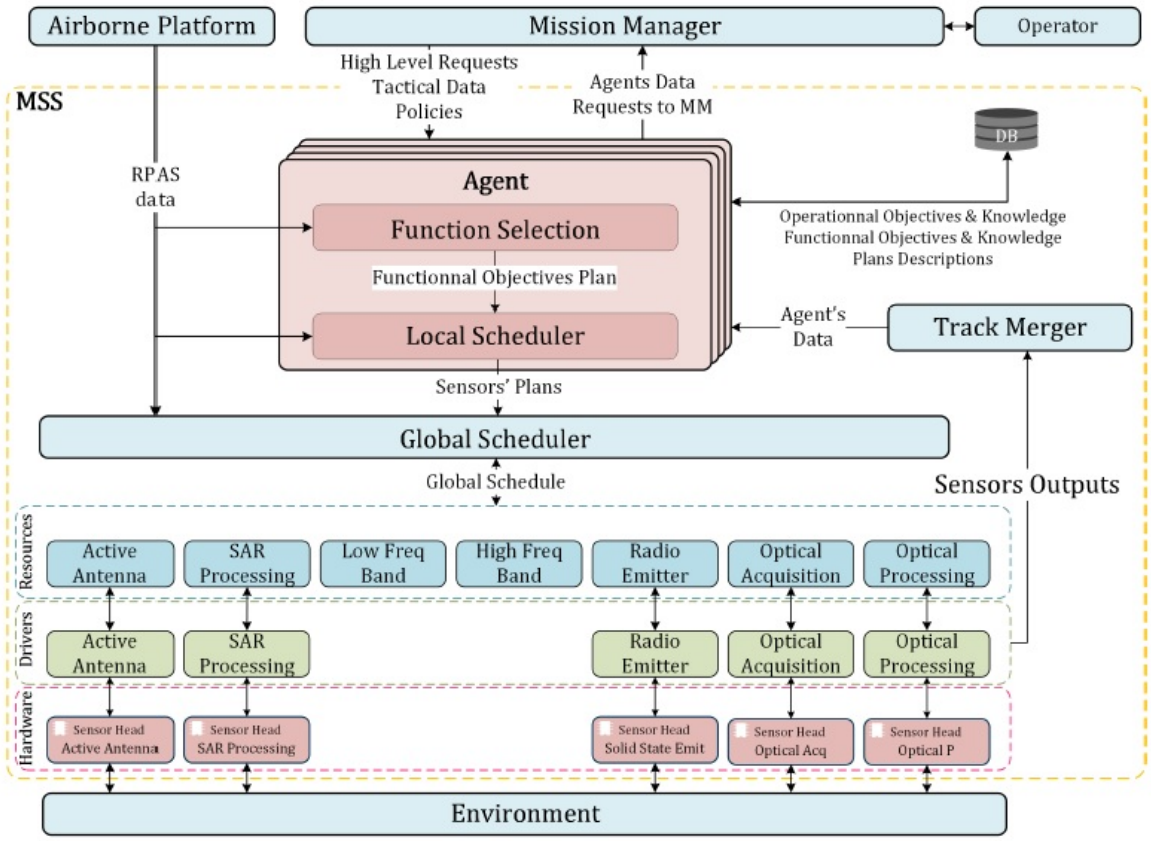

FIGURE 2.7 – Schéma de l'architecture système du SMS multi-agent, [Gri18].

chitecture multiplateforme et multi-milieu. Cette architecture nécessitera une approche distribuée avec une prise en compte des contraintes de communication et de la cohérence entre les systèmes distribués.

2.3.4 Conclusion

L'état de l'art sur le combat collaboratif Américain et les architectures ouvertes et orientées services dans le domaine militaire offre une vue d'ensemble des différentes approches et études en cours. Ces travaux d'architectures fournissent un cadre structuré pour la conception et le développement de systèmes complexes dans le domaine du combat collaboratif.

Nous avons vu que chaque architecture présente des caractéristiques et des qualités différentes, chacune offrant des réponses distinctes à la gestion de la complexité. Certaines

architectures se concentrent sur des aspects orientés services, favorisant la modularité et la réutilisabilité des composants, tandis que d'autres cherchent à masquer les détails d'implémentation pour faciliter l'interopérabilité et l'intégration. Le tableau 2.1 propose une synthèse de ces architectures.

Nous avons également présenté une architecture multi-agent qui cherche à ordonnancer les actions des senseurs d'une plateforme aéroportée, démontrant ainsi le potentiel des systèmes à base d'agents dans la coordination et la gestion de tâches complexes dans des environnements dynamiques et incertains, tels que les opérations militaires. Dans le contexte du combat collaboratif, les architectures à base d'agents offrent une approche prometteuse pour la conception de systèmes adaptatifs et réactifs. Les agents sont capables de prendre des décisions rapidement en fonction du contexte ou des objectifs de missions. Les architectures à base d'agents favorisent l'autonomie, la flexibilité, la modularité et la robustesse, ce qui les positionne comme de bons candidats pour répondre aux exigences opérationnelles du combat collaboratif.

Systèmes	**Types d'architecture**	**But**	**Qualités et caractéristiques**	**Origines**
CEC	Distribuée	Augmenter la réactivité face aux différentes menaces ; partager l'information ; engagement coopératif	Réactivité, interopérabilité, adaptabilité, amélioration de situation tactique	Américain
MOSAIC	Distribuée	Architecture conceptuelle qui vise à répartir les senseurs et systèmes d'armes ; proposer des collaborations entre systèmes	Flexibilité, adaptabilité, reconfigurabilité, résilience, bas coût	Américain
OMS	Ouverte SOA	Identifier les approches d'architecture afin de réduire les coûts ; développer et intégrer des systèmes d'armes	Modularité, interopérabilité, réutilisabilité, adaptabilité,	Américain
FACE	Ouverte	Définir un environnement de développement, des guides de conceptions et des interfaces pour l'avionique	Interopérabilité, réutilisabilité, abstraction, sécurité, flexibilité	Américain
SOSA	Ouverte	Poursuivre les travaux d'architectures MOSA existants	Interopérabilité, sécurité, modularité, compatibilité, portabilité, évolutivité,	Américain
CESMO	SOA	Améliorer la coopération entre les systèmes ESM	Interopérabilité, partage de bonnes pratiques, meilleure gestion des ressources, coordination	OTAN
CAMELOT	SOA	Concevoir un système de surveillance et de contrôle des frontières européennes	Standardisation, flexibilité, modularité, interopérabilité	Européen
RAMSES II	Centralisée Multi-agent	Proposer une architecture à base d'agents pour l'ordonnancement de tâches senseurs sur une plateforme unique	Autonomie, flexibilité, modularité, robustesse	Thales

TABLE 2.1 – Tableau récapitulatif de quelques architectures de systèmes existants du domaine militaire.

2.4 Systèmes multi-agents (SMA)

L'intelligence artificielle (IA) est devenue un atout indispensable dans les opérations critiques, offrant une vitesse et une efficacité que l'humain seul ne peut égaler. Dans ce contexte, l'intelligence artificielle distribuée (IAD) émerge comme une branche importante de l'IA qui adopte une approche distribuée au sein des architectures. Huhns, dans son ouvrage [Huh12], mentionne les raisons qui amènent à l'utilisation des IAD, notamment la résolution de problèmes complexes qui dépassent souvent les capacités de traitement d'un système centralisé. L'auteur souligne également les avantages des IAD tels que la modularité, la vitesse ou la fiabilité. Les algorithmes d'IA distribuée sont généralement classés en trois catégories : l'IA parallèle, la résolution de problèmes distribués (DPS) et les systèmes multi-agent (SMA) [DKJ18].

Wooldridge, dans son ouvrage [Woo09] qui introduit les SMA, définit l'agent comme « un système informatique situé dans un certain environnement et capable d'agir de manière autonome dans cet environnement afin d'atteindre les objectifs qui lui ont été assignés ». Le système multi-agent comprend plusieurs types d'agents intelligents qui peuvent communiquer entre eux à l'aide de protocoles de communication. Cette approche introduit les concepts de coopération, de coordination et de négociation qui sont des points clés des interactions entre les agents.

Dans cette partie, nous allons d'abord définir ce qu'est un SMA. Puis, les enjeux et challenges de cette technologie seront énumérés et détaillés. Ensuite, plusieurs articles évoquent les avantages et les inconvénients de l'approche multi-agent. Enfin, nous détaillerons l'utilisation des SMA pour l'allocation de tâches, qui est un sujet spécifique que propose la littérature et qui est au cœur de nos recherches.

2.4.1 Définir un SMA

Agent

Il convient de noter que la définition d'un agent varie et peut différer d'un article à un autre [BS10, CDJM01]. Ceci est dû à l'universalité du mot « agent » qui peut être représentatif de plusieurs entités dans différents domaines tels que des robots, des capteurs, des véhicules, des programmes, etc. Les définitions peuvent donc changer selon le domaine d'application qui utilise ces agents. Nous pouvons énumérer une liste des propriétés importantes d'un agent [CDJM01] :

— **Autonomie** : L'agent est capable de prendre des décisions seul, sans l'aide d'un autre agent ou d'un humain. Il connaît ses intentions et ses objectifs.

— **Rationalité** : L'agent est capable de raisonner afin d'optimiser la complétion de ses tâches en fonction de ses connaissances et de ses croyances, il agit avec raison mais également pour ses intérêts. Il pourra changer ses actions de manière intelligente, en tenant compte des contraintes et des informations disponibles sur son environnement.

— **Réactivité** : L'agent est capable de réagir rapidement à la suite d'un changement dans son environnement afin d'accomplir ses objectifs. Un agent purement réactif ne prend pas en compte son historique et l'évolution de son environnement.

— **Engagement** : L'agent s'engage sur certains objectifs, possiblement avec d'autres acteurs, il fera donc tout son possible pour accomplir ses engagements.

— **Intention** : Dans le cadre du modèle BDI (*Beliefs, Desires, Intentions*), l'agent possède un état mental, l'intention d'un agent représente son engagement actuel à entreprendre une action ou un ensemble d'actions en vue d'atteindre un objectif spécifique, en fonction de ses croyances sur l'environnement et de ses désirs pour certains résultats.

— **Sociabilité** : L'agent connaît l'environnement et ses acteurs ce qui lui permet d'interagir avec d'autres agents ou des humains pour atteindre ses objectifs.

— **Proactivité** : L'agent a un comportement orienté vers ses objectifs, il prend l'initiative pour satisfaire ses objectifs.

En résumé, un agent perçoit des modifications dans son environnement, puis il raisonne rapidement en prenant en compte son historique, ses buts et ses croyances pour décider de l'action à mener par la suite. Il peut aussi communiquer avec l'extérieur, avec d'autres agents par exemple, ce qui est le fondement d'un système multi-agent. Un agent peut donc avoir accès à des capteurs qui récupèrent des informations sur ce que nous appellerons l'environnement, détaillé au paragraphe suivant.

Environnement

La plupart des définitions d'agents incluent l'environnement comme élément clé. L'agent est situé dans un environnement, il récupère des données sur cet environnement qu'il est capable d'influencer et de modifier par ses propres actions. L'environnement est défini par Dorri et al. [DKJ18] comme l'endroit où l'agent évolue, pouvant prendre diverses formes

telles qu'un réseau, un logiciel, Internet, ou même le monde réel. L'environnement possède plusieurs caractéristiques importantes :

— **Accessibilité** : Il s'agit de la capacité à récupérer les données de l'environnement avec précision. Une accessibilité limitée peut résulter en des données bruitées ou incomplètes.

— **Déterminisme** : Cette caractéristique concerne la prédictibilité des résultats. Un environnement est déterministe si les résultats peuvent être prédits, sinon il est non déterministe.

— **Dynamisme** : Un environnement est considéré comme statique si les changements sont induits par les actions des agents uniquement. En revanche, il est dynamique s'il peut évoluer de lui-même. Le monde réel, en perpétuel changement, est un exemple d'environnement dynamique.

— **Continuité** : L'environnement est qualifié de discret s'il présente un nombre fini d'actions et de perceptions possibles. Wooldridge présente l'exemple des échecs qui est à un nombre d'états fini. Un environnement est considéré comme continu si le nombre d'états est infini, une course en taxi est un environnement continu. Un environnement continu complexifie souvent les problèmes auxquels les agents sont confrontés.

Système multi-agent

Les systèmes experts ou la programmation orientée objet partagent des similitudes conceptuelles avec les SMA. D'une part, Dori et al. [DKJ18] montrent les différences qu'il peut y avoir entre ces différentes approches, notamment en ce qui concerne les métriques utilisées pour la prise de décision, l'exécution des actions sur l'environnement, les communications internes au système, ainsi que le degré d'autonomie.

D'autre part, Wooldridge [Woo09] relève également les points communs et les différences entre les trois concepts. D'un côté, les SMA se distinguent de la programmation orientée objet par leur degré d'autonomie et leur capacité à offrir des comportements flexibles et autonomes. D'un autre côté, les systèmes experts opèrent généralement de manière isolée par rapport à leur environnement, recevant leurs données d'entrée principalement des utilisateurs et n'ayant pas la capacité de coopérer avec d'autres agents.

Les SMA sont un mélange de plusieurs domaines scientifiques [CDJM01] : l'intelligence artificielle distribuée, le génie logiciel, les systèmes distribués, l'informatique répartie, etc.

L'article [DKJ18] présente les sept fonctionnalités les plus importantes des SMA selon l'auteur :

- **Le *leadership*** : La présence d'un leader dans un groupe d'agents définit les actions pour tous les agents, tandis que l'absence de leader permet à chaque agent de décider de ses actions en fonction de ses propres objectifs. Un leader apportera de la coordination dans le système au détriment d'une dépendance centrale et d'une diminution de l'autonomie des agents.
- **La fonction de décision** : Si cette fonction est linéaire, la sortie reste proportionnelle à l'entrée, ce qui signifie que les décisions des agents dépendent des paramètres de l'environnement.
- **L'hétérogénéité** : Les agents peuvent être homogènes s'ils partagent des caractéristiques similaires, ou hétérogènes s'ils diffèrent les uns des autres. Dans un système, l'hétérogénéité apporte une diversité fonctionnelle mais peut rendre la coordination et la communication entre les agents plus complexe.
- **Paramètres de l'accord** : Les agents doivent définir des accords sur des paramètres appelés métriques, le nombre de métriques reflétant l'ordre du système. Dans un système d'ordre deux, les agents devront trouver un accord sur deux métriques. Augmenter le nombre de métriques permettra d'accroître la cohérence entre les agents, mais rendra difficile la recherche d'un accord.
- **La considération du retard** : On distingue généralement deux types de systèmes : ceux avec retard et ceux sans retard. Le retard peut être généré par des délais sur les communications et les traitements des agents. La plupart des applications du monde réel rencontrent toujours des retards non négligeables.
- **La topologie** : Elle peut être statique ou dynamique, décrivant les positions et relations entre les agents. La topologie dynamique offre une grande flexibilité au système, mais peut impacter la gestion des agents et leurs communications.
- **La fréquence de transmission des données** : Les données peuvent être partagées en réaction à des événements ou de manière périodique (*time-triggered* ou *event-triggered*). Un système où les agents partagent leurs données plus fréquemment sera plus réactif aux événements, mais également plus impacté par des problèmes de latences et de performance globale.
- **La mobilité** : Les agents sont mobiles ou statiques en fonction des besoins du système. Par exemple, la mobilité des agents peut contribuer à la mise à l'échelle

du système en répartissant la charge de travail de manière dynamique et en évitant les goulots d'étranglement potentiels.

2.4.2 Les enjeux et challenges des SMA

L'article [DKJ18] énumère une liste de challenges pour les SMA assez conséquente, nous nous référerons principalement à celui-ci pour cette partie en ajoutant d'autres enjeux qui peuvent être mentionnés ailleurs.

Le contrôle de la coordination Les actions des agents impactent l'environnement, ce qui influence indirectement les prises de décisions d'autres agents. Le contrôle de la coordination entre ces agents induit de nombreuses problématiques telles que le consensus, la contrôlabilité, la synchronisation, la connectivité et la formation. Dans le cadre de nos travaux, le consensus est une des problématiques dans la conception d'un SMA.

— Consensus : Il s'agit ici d'avoir un accord commun entre les agents sur des décisions ou des actions à mener afin d'assurer le cohérence et la coordination dans le système. Le consensus peut par exemple être obtenu par un vote entre les agents où la majorité l'emporte. Une étude sur les problèmes de consensus lors de la coordination multi-agent est effectuée dans [RBA05] et une autre étude récente introduit les motivations, les résultats et les méthodes de plusieurs problèmes de consensus multi-agent [LT19]. Cette dernière présente principalement le développement et l'état actuel de la recherche selon les aspects suivants : le consensus soumis à des contraintes de communication, le consensus qui suit un leader, le consensus de groupe, le consensus basé sur un mécanisme d'événement et le consensus en temps fini.

— Contrôlabilité : Le SMA peut être dirigé de manière spécifique d'un état initial à un état désiré à l'aide de règles. Cela assure une certaine certitude dans la capacité à atteindre un état particulier en suivant des étapes définies. Les deux métriques qui peuvent impacter la contrôlabilité sont le dynamisme dans la topologie et le déterminisme dans l'environnement. La collaboration est affectée quand la topologie change, car de nouvelles communications apparaissent et disparaissent. De plus, les actions des agents ne peuvent pas être anticipées dans un environnement non déterministe. C'est un enjeu de taille qui n'existe pas dans un système centralisé où un leader prend les décisions.

— Synchronisation : Les actions des agents sont ordonnées dans le temps, certaines actions doivent être exécutées avant d'autres pour garantir leur bon fonctionnement. La diversité des agents amène des problèmes de synchronisation, car celle-ci doit être réalisée sur des caractéristiques distinctes.

— Connectivité : Parfois, la connectivité entre les agents se veut permanente. La connectivité est complexifiée lorsque les agents se déplacent ou lorsque l'environnement est bruité entrainant des connexions et des déconnexions régulières.

— Formation : Les agents sont organisés sous forme de structure, parfois pour un intervalle de temps donné avant de subir une nouvelle réorganisation. Dans ce cas, l'enjeu de la formation suit trois étapes : trouver la formation la plus efficace, organiser les agents selon leurs structures, et maintenir la formation pendant le temps nécessaire. Ces étapes varient selon les objectifs du système, les capacités de chaque agent, les communications disponibles et l'environnement dans lequel le SMA évolue.

L'organisation Il s'agit des connexions et des communications entre les agents. Les organisations peuvent être basées sur plusieurs approches. Les agents peuvent être regroupés selon leurs fonctionnalités, leurs objectifs, leurs ressources ou leurs services. L'organisation des agents dans un SMA aura un impact sur les interactions entre les agents. La coalition et la hiérarchie sont des exemples d'organisations utilisées dans les SMA. Une structure hiérarchique peut impliquer des interactions commandées de haut en bas, tandis qu'une coalition regroupera les agents selon leurs objectifs, ce qui réduira les latences lors des communications au sein d'un même groupe.

La sécurité Les challenges en termes de sécurité viennent de la mobilité des agents, de leur sociabilité avec d'autres agents ainsi que de la décentralisation. La mobilité fait qu'un agent infecté par un malware pourrait diffuser plus rapidement de fausses informations à d'autres agents en se déplaçant. Les agents récupèrent les informations des agents voisins ou de l'environnement ce qui les rend vulnérables aux agents malicieux qui tenteraient de propager de fausses informations. Le manque de centralisation fait qu'il est difficile de vérifier l'identité des agents et de créer de la confiance entre les agents. Des mécanismes existent pour vérifier l'authenticité des agents, leurs autorisations, leur intégrité, la disponibilité des ressources et des services qu'ils utilisent, et le respect des confidentialités sur les informations échangées.

L'apprentissage L'apprentissage avec le *machine learning* est un challenge à part entière, suscitant de nombreux travaux de recherches [SV00, CCDN+21]. Cette approche soulève de nouvelles problématiques, telles que le cout accru des traitements pour les agents ainsi que la charge supplémentaire des communications, due notamment aux entêtes additionnels nécessaires pour les méthodes d'apprentissage. Parmi les méthodes les plus couramment utilisées pour l'apprentissage multi-agent, on retrouve l'apprentissage par renforcement (MARL : *Multi-Agent Reinforcement Learning*) et la programmation génétique.

La détection d'erreurs Il est essentiel de repérer les agents défaillants afin de les isoler et de prévenir toute collaboration avec d'autres agents, évitant ainsi la corruption de ces derniers. Parmi les problèmes les plus complexes à résoudre dans ce contexte figure le problème byzantin, où des agents défectueux peuvent agir de manière malveillante en transmettant des informations incorrectes ou en perturbant la communication entre les agents fonctionnels. Il existe des méthodes FDI (*Fault Dectection and Isolation*) qui sont majoritairement centralisées, un agent détecte et isole les agents défectueux. Pour les SMA, tous les agents peuvent échanger des données pour détecter et isoler les nœuds défectueux. Un état de l'art des FDIR (*Restoration*) dans les systèmes distribués est présenté dans [ZKA+16].

La localisation La vision d'un agent est limitée, donc localiser chaque agent peut être un défi. Un agent peut être localisé selon les ressources qu'il possède, les services qu'il exécute, ou selon son identité. La plupart des méthodes pour localiser les agents sont des méthodes centralisées. Cependant, une méthode distribuée est préférable pour des SMA de grande ampleur en raison des problématiques de mise à l'échelle. Le caractère dynamique de certains agents peut affecter la localisation, qui demandera plus de ressources de communication par rapport à un agent statique. Un autre agent peut estimer la localisation future d'un agent en observant sa direction et sa vitesse actuelle.

L'allocation des ressources Il s'agit des tâches que les agents doivent effectuer, chaque tâche demande du temps et un cout d'utilisation du CPU, des traitements et des communications. L'allocation peut se faire de manière centralisée ou décentralisée. Les paramètres importants à prendre en compte sont les ressources et la position d'un agent. Il faut connaître la charge de travail actuelle d'un agent avant de lui envoyer de nouvelles tâches

à faire qui pourraient le ralentir et retarder la réponse. Cet enjeu est la problématique majeure de nos travaux sur l'allocation de tâches à des senseurs.

2.4.3 L'allocation de tâches par SMA

Les articles [PRC+10, SD15, SST21, TE20, TMSS] ont étudié la problématique de l'allocation de tâches à des agents. L'allocation est un procédé qui fait correspondre des tâches à un ensemble de senseurs et qui essaie de maximiser une utilité globale.

Exemples d'utilisation des SMA pour l'allocation de tâches

Beal et al. [BUL+18] décrivent un système à base d'agents pour allouer des tâches aux senseurs dans le but de réduire le nombre de plateformes (ici des drones) pour accomplir un objectif. Des concepts d'agents sont présentés ainsi qu'un algorithme d'allocation de tâches.
Les auteurs proposent deux types d'agents : les agents *plateformes* et les agents *tâches*. Un agent *plateforme* est créé pour chaque ressource d'une plateforme (hypothèse : une seule ressource par plateforme). La particularité de cet agent est qu'il se décline en plusieurs agents *projetés*, en prenant en compte la position future anticipée de la plateforme. Un agent est créé pour chaque tâche et il communique avec les agents *plateformes* à portée de communication pour déterminer quelle plateforme accomplira la tâche.
L'algorithme d'allocation de tâches prend en compte deux métriques afin de classer les tâches proposées : le nombre de plateformes qui couvrent déjà la tâche et la priorité de la tâche. Un budget correspondant au temps de disponibilité du senseur est fixé, les tâches assignées sont prises dans l'ordre d'importance et dans la limite de ce budget. Les travaux restent tout de même limités au partage de la ressource de senseurs hautement directifs (caméras embarquées par exemple). Les auteurs démontrent qu'il est possible d'utiliser moins de plateformes pour effectuer le même nombre de tâches en se servant uniquement du partage de la ressource disponible. Les senseurs ne collaborent donc pas dans le but de proposer de nouvelles fonctionnalités.

Ponda et al. [PRC+10] utilise l'allocation de tâches afin que des agents hétérogènes dans un environnement dynamique puissent effectuer des missions. Ces missions impliquent l'exécution de différentes tâches comme la reconnaissance, la surveillance, la classification de cibles, et les opérations de secours. Certains UAVs (*Unmanned Aerial Vehicules*) sont plus appropriés que d'autres pour effectuer certaines tâches. Pour des

cas avec un nombre important d'agents, les approches centralisées deviennent rapidement impossibles à mettre en œuvre en raison de la complexité et du nombre d'interactions, il est donc nécessaire d'adopter des architectures décentralisées.

Dans l'article [TMSS], le problème d'allocation de tâches s'intéresse plus particulièrement à une tâche unique, avec un seul robot, et des affectations à durée prolongée. Un agent effectue une tâche à la fois, et chaque agent peut se voir affecter plusieurs tâches dans un planificateur. Trouver la solution optimale à cette allocation de tâches dans un environnement temps-réel devient irréalisable avec la charge de calcul croissante à mesure que le nombre de tâches et d'agents augmente.

Des approches pour l'allocation de ressources sont étudiées dans [DBGP21], le modèle d'agent considéré suit trois sous-comportements (*acting*, *communicating* et *planning*). L'article soulève le problème du goulot d'étranglement dû aux communications, point ayant également un impact dans notre approche. De plus, dans un système dynamique, le problème d'allocation de ressources varie au cours du temps rendant impossible la résolution du problème de manière optimale. Dans l'approche proposée par les auteurs, la communication des agents n'est possible que si les agents font partie du même ensemble connecté, par message direct ou par diffusion « broadcast ». Dans nos travaux, il faudra optimiser l'utilisation des communications en précisant les opérations de négociations nécessaires pour l'allocation de tâches.

Dans le cadre de la chasse aux mines sous-marines, Milot et al. [MCLC22] proposent une approche décentralisée d'allocation de tâches multirobots par enchère. La mission est composée de trois objectifs : détection, identification et neutralisation. Les robots se retrouvent en concurrence pour « acheter » des tâches lors d'enchères.

Méthodes d'allocation de tâches dans les SMA

Les articles [SD15, SST21, XCS18] présentent un état de l'art sur les méthodes d'allocation de tâches dans les SMA. La performance optimale d'un système global est un concept qui dépend de la perception de chaque agent et des contraintes du système. De ce fait le terme « utilité », souvent utilisé dans d'autres travaux, représente une valeur ou un coût affilié à cette allocation de tâches.

De nombreuses méthodes d'allocation existent et peuvent être classées selon plusieurs catégories [SST21] : la théorie des jeux ; les techniques basées sur l'optimisation qui englobent les heuristiques ou DCOP (*Distributed Constraint Optimization Problems* [DBGP21]), les optimisations déterministes (algorithme hongrois) ou les métaheuristiques

(*PSO, Bees algorithm, ant colony*) ; les approches orientées apprentissages (*MARL : Multi-Agent Reinforcement Learning*) ; et enfin les approches hybrides qui combinent plusieurs stratégies.

Nous nous intéressons plus particulièrement aux méthodes qui semblent le mieux répondre à nos objectifs, à savoir les algorithmes d'enchères (*auction-based*) qui permettent l'allocation de tâches de manière décentralisée et flexible. Les méthodes d'enchères sont efficaces bien que non optimales, et la mise à l'échelle est possible, car les coûts en calcul et en communication sont modérés [SST21]. Dans les méthodes par enchère, les agents misent sur des tâches, la mise la plus élevée remporte l'allocation de la tâche. La méthode habituelle pour déterminer le vainqueur est d'avoir un « commissaire-priseur » qui reçoit et évalue les mises centralement. Deux algorithmes par enchère sont proposés par Choi et al. [CBH09], le CBAA (*Consensus-Based Auction Algorithm*) et le CBBA (*CB Bundle Algorithm*). Le premier répond au problème d'allocation unique, tandis que le second correspond à l'allocation multiple. Le CBBA, une extension du CBAA, est un algorithme décentralisé qui introduit la notion de groupement de tâches en paquets, permettant ainsi aux agents de proposer des offres groupées pour des ensembles de tâches liées. Il existe d'autres algorithmes, notamment des variantes du CBBA, mais aussi le CNP (*Contract Net Protocol*) qui est un protocole standardisé permettant d'allouer ainsi que de réassigner des tâches à des agents.

L'approche du CBBA permet une décision décentralisée, nécessaire lorsqu'il y a de nombreux échanges dans de grandes équipes d'agents. De plus, cet algorithme est considéré comme polynomial en temps de calcul ce qui lui permet une mise à l'échelle facilitée avec le nombre de tâches ou la taille du réseau qui augmentent. Enfin, des objectifs de conception différents, des modèles d'agents et des contraintes peuvent être incorporés en définissant des fonctions de scores appropriées. Le besoin en synchronisation du CBBA rend son utilisation dans des applications temps réel moins efficace, les mêmes auteurs tentent de compenser cet inconvénient avec le ACBBA (Asynchronous CBBA), une extension de l'algorithme existant [JPCH11].

Otte et al. [OKS20] comparent six algorithmes d'enchères : l'enchère séquentielle, l'enchère parallèle, l'enchère G-Prim, la répétition de l'enchère parallèle, l'enchère combinatoire et la répétition de l'enchère G-Prim. Dans ces algorithmes, un « tour » désigne les échanges entre les agents pour miser et remporter des enchères. Les algorithmes nécessitent donc un ou plusieurs tours pour atteindre un consensus. Un « article » correspond à une tâche mise en enchère et chaque agent fait une offre sur l'article pour le remporter.

Enchère parallèle. Dans cet algorithme, un seul tour d'enchère est effectué. Le commissaire-priseur diffuse la liste des articles et chaque agent soumet une offre en calculant la valeur de chaque article. Une date limite de réception des offres est définie, puis la liste des gagnants est diffusée. Ces derniers doivent accepter leur gain pour valider l'achat.

Enchère séquentielle. Un article est vendu à chaque tour pendant un total de m tours. À chaque tour, le commissaire-priseur choisit aléatoirement un article non vendu et le met aux enchères, en diffusant l'offre à tous les agents. Chaque agent mise sur l'article, et le commissaire-priseur détermine le vainqueur, qui doit également valider l'achat.

G-Prim. Cet algorithme récupère des éléments des deux enchères précédentes. Plusieurs articles sont mis en vente à chaque tour. Les agents évaluent l'article non vendu qu'ils estiment être le meilleur, et les gagnants sont ceux qui ont fait les mises les plus élevées sur chaque article.

Répétition d'enchère parallèle. A chaque tour d'enchère, le commissaire-priseur met en vente n articles qui n'ont pas encore été vendus. Chaque agent soumet une offre pour chaque article. Le commissaire-priseur attribue les articles de manière gloutonne, en accordant le premier article à l'agent ayant fait la mise la plus élevée, et ainsi de suite pour les articles suivants. Les agents acceptent ou refusent les articles, et le nombre de tours de l'algorithme dépend du nombre d'articles vendus à chaque tour.

Répétition des enchère G-Prim. Dans cet algorithme, n articles sont mis en vente et le commissaire-priseur informe sur les articles non vendus à chaque tour. Chaque agent fait une proposition sur les articles non vendus à chaque tour d'enchère, s'ils gagnent ils doivent également accepter.

Enchère combinatoire. L'enchère combinatoire propose que chaque agent puisse proposer une mise sur chaque article en un seul tour. Le commissaire-priseur calcule la manière la plus optimale de répartir les articles gagnés. Cet algorithme peut trouver une solution optimale si les valeurs des articles ne sont pas indépendantes, au détriment d'un temps de calcul exponentiel selon la quantité d'articles à traiter.

Conclusion

Ce chapitre a permis de présenter l'état de l'art des recherches sur le combat collaboratif, les architectures ouvertes, les architectures orientées services et les systèmes multi-agents. Nous avons pu observer que les architectures pour le combat collaboratif cherchent à interconnecter les systèmes et à partager les données pour améliorer la situation tactique. Les études sur les architectures militaires montrent un intérêt particulier

pour l'ouverture des systèmes, qui offre des capacités de modularité, de réutilisabilité ou encore de mise à l'échelle à ces systèmes. Par ailleurs, nous avons exploré les principes des architectures orientées services, dont les avantages en termes de découplage, d'évolutivité et de flexibilité sont particulièrement attrayants. Cette approche inspire de nombreuses solutions architecturales, notamment dans le domaine militaire.

Notre attention s'est ensuite portée sur les systèmes multi-agents (SMA), une technologie qui pourrait permettre de répondre à notre problématique. Nous nous sommes focalisés plus spécifiquement sur les recherches portant sur l'allocation de tâches aux agents. Nous avons identifié plusieurs catégories de méthodes pour résoudre ces problèmes et examiné leur pertinence par rapport à nos objectifs. Il en ressort que les algorithmes d'enchères se démarquent comme des solutions particulièrement appropriées. Leur approche décentralisée et leur efficacité d'implémentation offrent une flexibilité et une mise à l'échelle recherchées dans le cadre de nos travaux. En effet, les enchères permettent une allocation efficace des ressources dans un environnement distribué et dynamique. Leur décentralisation favorise une prise de décision autonome par les agents, tandis que leur adaptabilité permet de répondre rapidement aux changements dans l'environnement. De plus, les mécanismes d'enchères encouragent la concurrence entre les agents, favorisant ainsi une utilisation optimale des ressources disponibles. Cependant, nous reconnaissons que la coordination des agents et les coûts des communications restent des défis importants à relever dans un tel système distribué.

Chapitre 3

Contributions

3.1 Approche conceptuelle d'une architecture de combat collaboratif naval

La collaboration dans un système de combat peut être catégorisée selon quatre niveaux : monoplateforme, multiplateforme, interarmées et internationale. Elle est souvent comparée à la coopération, qui est un terme proche mais bien distinct, comme nous l'avons expliqué dans le chapitre d'introduction. La coopération implique des entités travaillant ensemble vers un objectif commun avec des interactions limitées et sans nécessairement partager leurs ressources. En revanche, la collaboration suppose une coordination plus étroite des ressources avec des échanges de données fréquents. Dans le cadre de la coopération, les tâches sont partagées en fonction des capacités des entités, tandis que la collaboration permet d'accomplir des tâches ou des services collaboratifs nécessitant l'intervention de plusieurs entités.

Tout d'abord, la collaboration monoplateforme met en jeu les senseurs d'une plateforme dans le but de fournir des services collaboratifs. Par exemple, l'amélioration d'un produit senseur en combinant l'utilisation du radar et de l'optronique.

Ensuite, la collaboration multiplateforme reprend la même démarche en étendant la collaboration aux senseurs d'autres plateformes. En effet, une plateforme peut avoir besoin de senseurs complémentaires lui permettant d'avoir une vision alternative sur une menace et d'apporter des données techniques supplémentaires pour améliorer les pistes senseurs. La collaboration multiplateforme procure des choix de fonctions additionnelles pour les opérations grâce à l'utilisation coordonnée des équipements. Des problématiques de réseau et de synchronisation émergent, car des messages devront être échangés pour garantir la cohérence des actions collaboratives.

Les opérations interarmées constituent un troisième niveau de collaboration, avec un besoin d'entraide entre les composantes aériennes, terrestres et navales. Cet axe ajoute des

besoins d'interopérabilité et de nouvelles problématiques dues aux différents milieux opérationnels. Les contraintes varient selon le type de plateforme : l'espace pour l'installation des équipements, les moyens de communication utilisés ou les interfaces.

Enfin, le dernier niveau de collaboration inclut la collaboration avec les forces des nations alliées. Chaque pays souhaite conserver le secret sur les systèmes et sur certaines données. Des problématiques de confidentialité et de standardisation sont donc à prendre en compte pour proposer une utilisation collaborative des ressources sans avoir la connaissance exacte sur leur fonctionnement. Des standards, tels que les normes STANAG (*Standardization Agreement*) de l'OTAN, visent à améliorer l'interopérabilité des systèmes alliés en proposant des interfaces pour les échanges et interactions entre ces systèmes.

Les précédents travaux de nos équipes ont démontré des conceptualisations dans un contexte de collaboration monoplateforme [Gri18]. Les concepts présentés dans le cadre de cette thèse portent sur un niveau de collaboration multiplateforme avec intégration des composantes aériennes et navales. Certaines plateformes navales embarquent des aéronefs, des hélicoptères et des drones. Leur intégration dans les conceptions d'architecture est requise afin de proposer une architecture commune permettant de les interconnecter. L'élaboration d'une solution qui suit les recommandations et bonnes pratiques nous permettra de concevoir un système adaptable à différents types de plateformes. L'utilisation d'un modèle de service simplifié et abstrait permet la réalisation d'un POC (*Proof Of Concepts*). Cependant, la levée de cette abstraction amène de nouvelles problématiques : les services sont plus spécifiques et le développement de leur normalisation doit s'appliquer aux cas d'usages existants.

3.1.1 Approche architecturale sous forme de graphe multipolaire

Une approche employant la théorie des graphes est présentée pour illustrer les attentes des architectures pour le combat collaboratif naval (CCN). Les graphes sont des modèles abstraits bien connus et utilisés pour modéliser et représenter des relations entre des entités dans divers domaines tels que les réseaux sociaux ou les télécommunications. Un graphe est un ensemble de sommets (aussi appelés nœuds) et d'arêtes qui associent deux sommets entre eux. Les graphes permettent de décrire classiquement un réseau de données et de communication, dans notre cas, un réseau de senseurs. Le graphe multipolaire combine les graphes centralisés et décentralisés. Dans cette architecture, les pôles peuvent représenter

des plateformes et regrouper un ensemble de sommets interconnectés. Deux sommets de pôles différents peuvent se connecter par des liens dits « faibles ». Cette qualification vient du fait que la communication peut être interrompue plus facilement selon les protocoles réseau et le médium utilisés, ce qui peut entraîner une diminution de la fiabilité, un débit plus faible, des coûts plus élevés et une qualité de service (QoS) moins satisfaisante. En comparaison, les liens forts sont plus fiables et plus rapides, comme cela peut être le cas avec des liaisons internes directes.

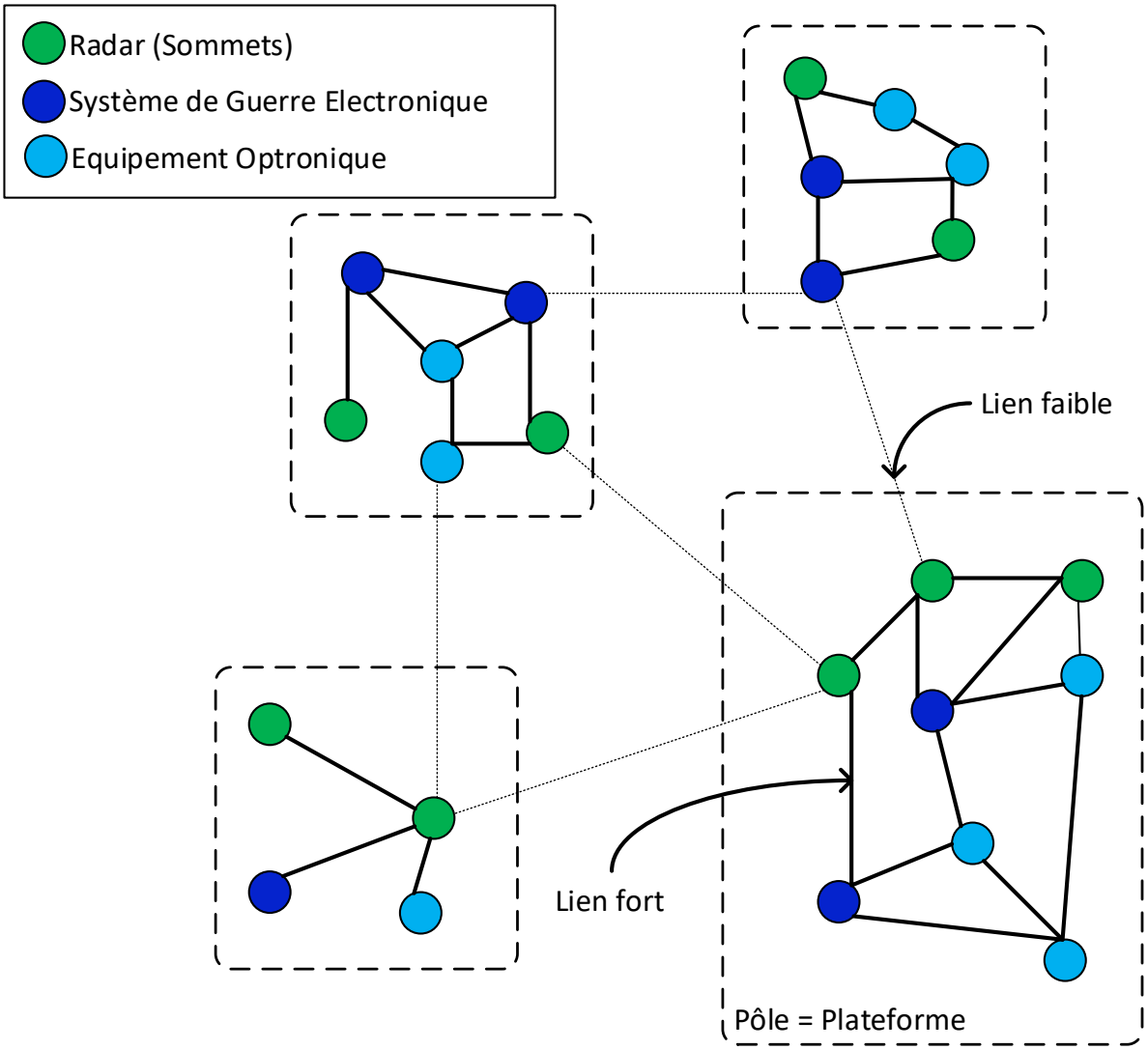

FIGURE 3.1 – Graphe multipolaire représentant une architecture distribuée.

La figure 3.1 représente un réseau multipolaire de 4 pôles comprenant chacun dif-

férents types de senseurs. La définition d'un nœud peut s'étendre à plusieurs éléments comme des systèmes de traitement de l'information et d'aide à la décision, des systèmes d'information et de communication, des systèmes de commandement ou des effecteurs. C'est pour cette raison que nous utilisons le terme de « ressource », terme plus générique pour désigner tout élément pouvant communiquer et contribuer au succès de la mission. Nos travaux porteront plus spécifiquement sur les ressources particulières que sont les senseurs, mais pourraient s'étendre à d'autres ressources. Un nœud, correspondant donc à un senseur, propose un ensemble de fonctions apportées par chaque senseur, telles que la détection, le renseignement ou la tenue de situation. Plusieurs senseurs peuvent potentiellement répondre à un même service, ce qui accroit la résilience du système en permettant en particulier de trouver des solutions alternatives suite à la défaillance d'un équipement. Lorsqu'un nœud se déconnecte ou se reconnecte, sans perte de connectivité pour les autres nœuds en place, cela montre que le réseau peut gérer ces modifications de manière automatique, sans nécessiter d'intervention complexe de l'utilisateur pour rétablir la connectivité ou reconfigurer les paramètres du réseau.

Les arêtes représentent les liens de communication entre les nœuds, ces liaisons peuvent utiliser différentes technologies des communications, comme des liaisons satellites, des câbles réseau ou des réseaux hertziens émettant sur des fréquences définies (HF, UHF, etc.). La modulation des contraintes de connexion doit être possible, en proposant par exemple de la redondance afin d'apporter de la résilience en cas de perte de liaison, ou en ajoutant, supprimant, reconfigurant des liaisons pour s'adapter à certaines situations.

3.1.2 Apport du collaboratif dans la boucle OODA

« La collaboration entre senseurs disposant d'algorithmes embarqués d'Intelligence Artificielle est un axe majeur d'augmentation capacitaire pour réduire drastiquement le cycle OODA dans le cadre du combat collaboratif » [Bar]. Le cycle OODA du Colonel John R. Boyd [Ric20] est une stratégie qui vise à accélérer au maximum les différentes étapes d'observation, d'orientation, de décision et de manoeuvres. L'armée qui arrivera à maitriser ce cycle plus rapidement que son adversaire aura un avantage stratégique certain. Dans notre contexte, la figure 3.2 explique le fonctionnement de cette boucle.

Dans le cadre de nos travaux, la phase *act* n'est pas considérée, elle comprend des effecteurs des plateformes qui sont en dehors du cadre de nos travaux. La phase d'observation requiert les senseurs, ceux-ci vont fournir des données sur l'environnement qui seront traitées lors d'une phase d'orientation par des algorithmes, des analyses, des retours

d'expérience ou de l'IA. Une décision sera rendue sur la suite des actions à effectuer, cela peut être une proposition d'allocation de tâches dans notre cas. Dans un cas plus général, les décisions peuvent également être des choix des meilleures actions à entreprendre. Une fois la décision prise, elle est mise en oeuvre dans une phase d'action : les effecteurs des plateformes peuvent être utilisés, la plateforme peut décider de se replier, les senseurs peuvent être affectés à des menaces. Le système peut également estimer manquer d'informations et repartir sur une phase d'observation. De son côté, l'adversaire tentera, par tous les moyens, de se prémunir en perturbant le cycle ennemi, surtout s'il a connaissance de sa stratégie. En règle générale, les actions prises par le système et les opérationnels en mission doivent suivre les principes fondamentaux des doctrines militaires [Ven08]. Ces doctrines établissent des principes directeurs, des tactiques et des stratégies qui guident les forces armées dans leurs opérations.

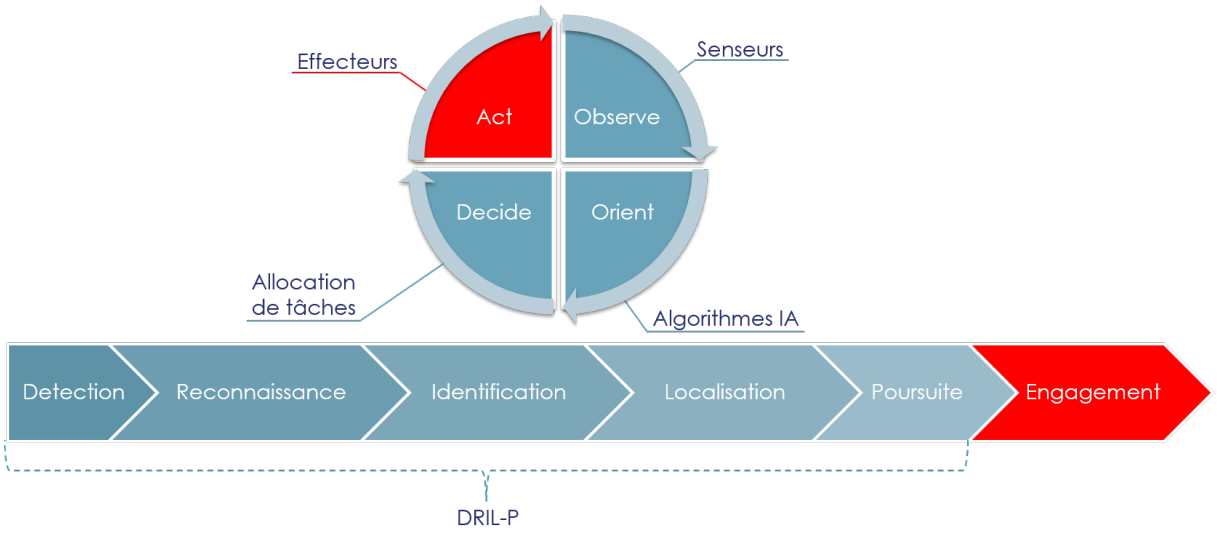

FIGURE 3.2 – Cycle de décisions et d'actions DRIL-P.

En reprenant ces différents éléments, les senseurs sont les ressources qui vont fournir de nouvelles informations aux algorithmes d'IA, des systèmes multi-agents par exemple, pour traiter l'information et déterminer les tâches suivantes à réaliser. Le but du système est de préciser les informations des cibles d'intérêt en itérant des actions senseurs qui permettront de les compléter. L'engagement d'une cible par des effecteurs est en dehors du cadre de nos travaux et nécessite de connaître au préalable la cible en passant par les étapes de la chaine fonctionnelle DRIL-P (détection, reconnaissance, identification, localisation et poursuite). Une meilleure connaissance du théâtre permet de décider des

cibles prioritaires, dont les connaissances doivent être appronfondies rapidement, et des menaces à engager à l'aide des effecteurs lorsque cela est possible et nécessaire.

Frédéric Barbaresco [Bar] décrit deux types de senseurs intelligents permettant le combat collaboratif :

- Le « méta-senseur » est composé de plusieurs senseurs complémentaires coopérants en boucle courte pour partager leurs ressources. Le système ne voit qu'un seul senseur global pour effectuer des missions.
- Le « senseur distribué » collabore en boucle courte et est mis à disposition des utilisateurs du système. L'utilisation optimale des senseurs distribués peut être atteinte par la sélection des senseurs optimaux selon leurs caractéristiques et leur positionnement. Les ressources sont allouées dynamiquement à l'aide d'un orchestrateur permettant la collaboration entre les senseurs (exemple : radar multistatique).

Les axes étudiés dans les études de conception des senseurs collaboratifs sont : les enchères distribuées, l'apprentissage par renforcement distribué, l'optimisation sous contraintes distribuées (DCOP), et les systèmes multi-agents (SMA). Nous focalisons nos travaux sur l'utilisation des SMA pour l'allocation de tâches à des senseurs collaboratifs. Cette décision est motivée par la capacité des SMA à s'adapter dynamiquement et de manière décentralisée, ce qui est crucial dans les environnements distribués comme les réseaux de senseurs.

3.1.3 Spécificité de l'architecture collaborative

La collaboration entre senseurs amène le système à proposer de nouveaux services pour répondre aux attentes des opérationnels lors de missions. Selon le modèle orienté service, chaque service doit décrire ses entrées, ses sorties, ses fonctions, son rôle et spécifier ses connexions et ses interfaces, afin qu'un utilisateur, au sens large, puisse s'en servir. L'ensemble des services est mis à disposition dans un réseau à destination d'un ou de plusieurs consommateurs.

Les ressources détectées dans le réseau fournissent un ou plusieurs services. L'utilisateur n'a connaissance que des services disponibles dans un annuaire et peut les utiliser sans avoir à connaître les détails internes de leur implémentation. Les échanges entre le consommateur et le fournisseur du service passent par une interface, qui permet aux ressources de se connecter dynamiquement au réseau en déclarant les services proposés avec leur description et leur « manuel d'utilisation ».

3.2 Définition d'un scénario

L'élaboration d'un scénario est une étape importante dans la conception d'une architecture d'un système de senseurs. Le scénario appliqué doit permettre de montrer que l'architecture réagit comme l'architecte l'a pensée, et d'évaluer l'architecture selon des critères définis, qui sont autant de marqueurs pour comparer l'efficacité et la performance. Dans un premier temps, le scénario doit couvrir de multiples contextes opérationnels. Ensuite, pour chaque événement du scénario, une action spécifique du système doit avoir lieu pour mener à bien sa mission. Un scénario doit être crédible et représentatif, en tenant compte d'aspects dimensionnants tels que la complexité des interactions, et le besoin de mettre en jeu différentes fonctions et événements pour refléter fidèlement les situations réelles. Le scénario doit prendre en compte un environnement riche comprenant différentes plateformes utilisant de nombreux senseurs, ainsi qu'une variété d'événements imprévus et d'actions qui sont sur le point de se produire. Le scénario permet de décrire les conditions et les interactions attendues dans des situations données, ce qui permet de vérifier chaque fonctionnalité du système.

Dans le cadre de nos travaux, le scénario envisagé exige d'être en phase avec la réalité pour évaluer l'architecture proposée. Notre scénario doit également être non soumis à la confidentialité et justifier les différentes fonctionnalités et besoins de l'architecture explicités dans le chapitre 1. Le scénario positionne, sur un théâtre d'opérations, des forces alliées appelées force bleue, et des forces ennemies appelées force rouge. Avant de définir le scénario, il est crucial de clarifier les objectifs que nous visons à travers celui-ci. Nous voulons montrer l'intérêt de la collaboration entre les plateformes à l'aide de services dits *collaboratifs* afin de gagner en connaissances tactiques dans un contexte chargé, et cela de manière réactive. Par exemple, aujourd'hui, dans le domaine naval, chaque plateforme établit sa situation tactique locale avec à ses propres senseurs et au moyen d'échanges entre les CMS des différentes plateformes par les liaisons de données tactiques (LDT). La fusion de données des senseurs est centralisée sur la plateforme de commandement et cette fusion donne une vision d'ensemble du théâtre des opérations, c'est ce qu'on appelle la situation tactique globale. Le scénario permet donc de montrer l'intérêt d'une architecture distribuée par rapport à l'architecture centralisée existante.

3.2.1 Formalisation

Nous voulons proposer une architecture distribuée selon l'approche orientée services dans laquelle le système de combat n'utilise pas les ressources de chaque plateforme de manière centralisée, mais en réseau avec les ressources d'autres plateformes. Le modèle d'architecture à base de services permet l'intégration de services avec des mécanismes de découverte de services et de composition de fonctions afin de fournir de nouvelles capacités [RLLX08]. Les services peuvent ainsi évoluer ou être modifiés. Dans nos travaux, des ressources senseurs mettent à disposition des services que nous appelons « services senseurs ». De plus, nous évoquons le concept du « service opérationnel » qui correspond à un service sollicité par un opérateur. La localisation ou l'identification sont des services opérationnels qui peuvent être rendus par une combinaison collaborative de plateformes et de senseurs. Une « requête opérationnelle » est une demande de l'opérateur qui définit les spécificités du service opérationnel. Dans la suite des travaux, par abus de langage, nous utiliserons régulièrement les termes de « services » et de « requêtes » pour désigner respectivement les services senseurs et les requêtes opérationnelles. La dénomination « service opérationnel » fait partie intégrante de la requête et nous l'utilisons peu en comparaison du service senseur.

Definition 8 (Service senseur) Un service senseur fournit une capacité (ou une fonction) que proposent un ou plusieurs senseurs. Les senseurs déclarent les services qu'ils peuvent accomplir dans un registre qui sera ensuite partagé entre des utilisateurs de services. Le service senseur est caractérisé par une qualité de service (QoS), des règles, et des ressources qui permettent son utilisation. Il répond à une requête de l'utilisateur sans que celui-ci n'ait besoin de connaître son fonctionnement interne.

Definition 9 (Requête opérationnelle) Une requête opérationnelle est issue d'un ordre de mission interne ou externe à la plateforme. Cette requête décrit les besoins durant une opération, elle comprend notamment : le service opérationnel, la zone géographique d'application, la durée effective de la requête, les contraintes, une priorité et la qualité de service désirée.

Les plateformes échangent entre elles par le biais de liaisons de communication. Une qualité de service est demandée afin de déterminer quelles plateformes et quels senseurs peuvent remplir les objectifs de mission de façon adaptée. Un service opérationnel peut être rendu par plusieurs services senseurs. Par exemple, une requête opérationnelle peut

demander la réalisation d'un service opérationnel de localisation avec une contrainte de discrétion et une priorité élevée. Cette requête pourra être accomplie par différents services senseurs comme des services de TDOA (*Time Difference of Arrival*) ou de FDOA (*Frequency Difference of Arrival*) tout deux présentés dans [SRF20]. Ces services permettent la géolocalisation passive, ce qui écarte l'utilisation du radar par le système en raison d'un choix opérationnel de discrétion. Nous proposons d'abstraire le concept de service, en décrivant un service capable d'effectuer une localisation à la place de spécifier exactement un service de TDOA qui est en dehors du cadre de nos activités. Nous supposons que nos services, une fois sélectionnés, contribuent assurément à l'avancement de la mission. Dans le cas d'un service de localisation, nous admettons que le service a réussi et que le niveau de précision des résultats ne nécessite pas de redemander ce service.

3.2.2 Scénario

Le scénario contient les forces alliées du groupe aéronaval (GAN), qui représente un potentiel de projection de puissance important au regard de l'ennemi. Les unités au sein du GAN sont représentées par la couleur bleue :

- ➤ Le porte-avions Charles de Gaulle ;
- ➤ Deux Rafales marine ;
- ➤ Un E-2D (Hawkeye) ;
- ➤ Une frégate de défense aérienne (FDA) ;
- ➤ Une frégate FREMM (multimissions) ;
- ➤ Une frégate de type FREMM DA (défense aérienne) ;
- ➤ Deux frégates de défense et intervention (FDI).

Ces plateformes sont présentées dans le chapitre 1 et en annexe. Au niveau de la force rouge, nous schématisons les menaces ou cibles ennemies avec des points rouges. Nous supposons que ces menaces sont inconnues, bien qu'en réalité, lors d'une mission, les forces alliées s'attendent à certaines menaces dans des zones spécifiques ce qui permet d'accélérer les processus de reconnaissance et de prise de décision. Les menaces en mer sont souvent des plateformes navales, des avions, des missiles ou des menaces asymétriques (par exemple, des embarcations armées). La difficulté réside dans la masse d'informations que représente l'ensemble des détections civiles telles que les bateaux de pêche, les cargos ou les voiliers, qui s'ajoutent à la liste de menaces déguisées potentielles. Le suivi de ces pistes permet de garde une vigilance sur le changement de leur comportement.

Chaque plateforme bleue possède sa liste de ressources que sont les senseurs embarqués radar, GE et optronique. L'architecture doit permettre de répondre à la problématique de l'allocation de tâches à des senseurs par collaboration multiplateforme. Le scénario mettra en évidence les aspects de reconfigurabilité du réseau et de fonctionnement du système dans les situations suivantes : lorsqu'un allié rejoint le réseau, lorsqu'une plateforme détecte une cible, et lorsqu'une plateforme quitte le réseau. Les défaillances des senseurs pourront également être observées en cas de brouillage ennemi, dans ce cas une demande d'aide à une plateforme alliée est considérée.

Dans le scénario, le GAN (Groupe aéronaval) conserve sa formation de déploiement et des événements particuliers apparaissent tout au long de la mission. Le but principal du GAN est de remplir ses objectifs fixés en préparation de mission, tout en gérant les imprévus comme des menaces non anticipées, par exemple, qui contraignent la force alliée à s'adapter. L'opérateur établit des requêtes opérationnelles qui définissent les actions à mener. Par exemple, cela peut-être une demande de veille dans une zone, ainsi la mission sera focalisée en priorité dans cette zone d'intérêt pour l'opérateur. Les requêtes ont des niveaux de priorités variables qui conditionnent leurs traitements, une requête prioritaire doit être traitée avant une requête moins prioritaire. Pour chaque menace dans la zone, le système doit effectuer une suite d'actions senseurs définis dans la chaine d'acquisition suivante : DRIL-P (détection, reconnaissance, identification, localisation et poursuite). Dans l'idée de simplifier notre problème, nous supposons que les étapes de cette chaine d'acquisition s'enchainent les unes à la suite des autres. Lors de la détection d'une menace, celle-ci doit être reconnue, identifiée, puis localisée avant d'être poursuivie. La requête opérationnelle peut également impacter le choix des actions à effectuer par les senseurs, c'est par exemple le cas si le système demande de simplement localiser une cible. Par conséquent, le traitement s'arrêtera une fois la cible localisée et le système ne la poursuit pas.

3.2.3 Vignettes

Le scénario peut être divisé en vignettes, une vignette représentant un événement plus court et avec une portée réduite. Elle correspond à un sous-ensemble d'un scénario de la flotte et des cibles d'intérêt à un instant donné. Ces vignettes serviront à montrer différentes caractéristiques de fonctionnement de l'architecture. Dans les vignettes qui suivent, le porte-avions est représenté par un rectangle, les plateformes aériennes par des triangles et les bâtiments de surface par des losanges. Les cercles décrivent les portées de

détection des entités, nous supposerons que les plateformes peuvent communiquer entre elles, que ce soit par liaison directe, ou indirecte avec des relais.

Vignette I

Sur cette vignette, nous pouvons observer en rouge la zone géographique de la requête opérationnelle. Les potentielles menaces sont en rouge et ne sont détectées que par les plateformes alliées à portée. De plus, la requête opérationnelle spécifiée en préparation de mission indique que toute détection en dehors de la zone rouge est ignorée, sauf si elle appartient à une autre requête opérationnelle avec des propriétés différentes.

Cette vignette définit un événement ou une situation simple, celui où plusieurs cibles sont à détecter et où le système doit recueillir des informations pour chacune d'entre elles, afin de les spécifier comme dangereuses ou non.

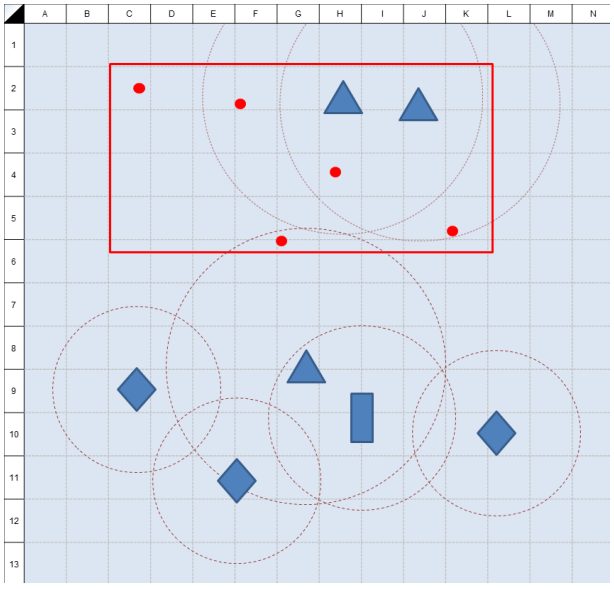

FIGURE 3.3 – Vignette 1 : détection de menaces dans une zone opérationnelle.

Dans les systèmes actuels, seules les plateformes à portée de détection obtiendraient ce qu'on appelle une piste senseur (ou *track* en anglais). Cette piste serait alors partagée dans le réseau avec les autres plateformes. Comme un objet peut être détecté par plusieurs senseurs (radar, GE et optronique), un système de fusion des pistes permet de les corréler et ainsi d'obtenir une piste système [SB17]. Ensuite, chaque plateforme ayant détectée la piste décide des actions senseurs à mener pour enrichir sa situation tactique locale, avant de partager à nouveau les données avec les alliés.

Dans notre système distribué, chaque plateforme est responsable d'un certain nombre de cibles, partage l'état de ses ressources et cherche à exploiter les senseurs les plus adaptés de la flotte pour accomplir les objectifs de la mission. Ainsi, dans cette vignette, les menaces détectées sont prises en charge par les plateformes les plus proches en équilibrant la charge de calcul. Les plateformes échangent des messages pour connaître l'état des ressources et savoir s'ils peuvent les utiliser, afin d'améliorer la connaissance tactique sur la cible. Plusieurs ressources senseurs pourront être utilisées en parallèle pour accomplir un service collaboratif.

Vignette II

Sur cette seconde vignette, une plateforme navale (représentée par un losange vert) approche de la zone de la mission et souhaite rejoindre la flotte alliée. En supposant que la plateforme est à portée de communication, celle-ci propose ses ressources et exécute une demande d'utilisation de ressources à l'ensemble de la flotte. Les tâches en cours ne doivent pas être arrêtées, sauf si l'arrivée de la nouvelle plateforme permet d'améliorer significativement les performances de ces tâches, par rapport à l'impact de l'annulation d'un service en cours.

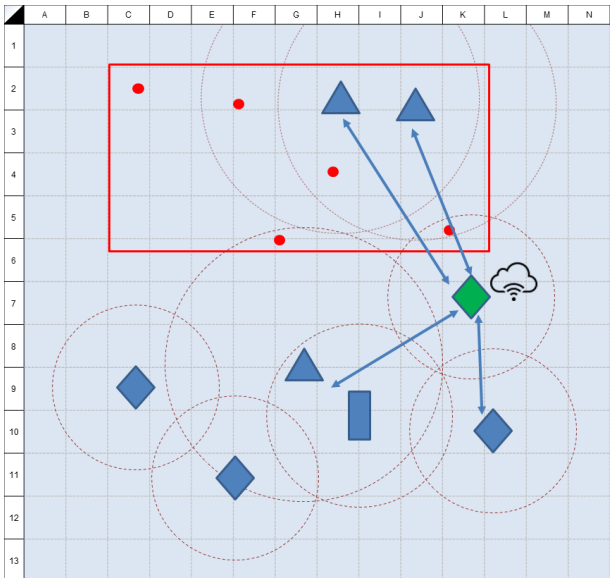

FIGURE 3.4 – Vignette 2 : Intégration d'une plateforme dans le réseau.

Dans le cas d'une coopération avec des alliés, la plateforme ayant rejoint peut également faire connaître ses propres objectifs à accomplir. Selon l'importance de sa mission

et des niveaux de priorités imposés, les plateformes alliées peuvent être amenées à libérer de la ressource. Ce cas d'usage permet de démontrer la capacité « *Plug-and-Play* » avec l'intégration d'une nouvelle plateforme et de ses senseurs dans le catalogue des services du réseau de plateformes existant.

Vignette III

Dans cette vignette, l'ennemi opère un brouillage contraignant l'utilisation des ressources de deux plateformes. Ces dernières perdent en sensibilité radar sur une ou plusieurs bandes de fréquences les empêchant de « voir » clairement la zone orangée. Les plateformes mettent à jour leurs capacités en retirant ou modifiant les caractéristiques des ressources impactées par ce brouillage, puis en informent les autres. Des services en cours sont potentiellement annulés et le système doit composer avec d'autres ressources disponibles pour mener à bien sa mission. Les menaces sont toujours détectées par une autre plateforme, et si ce n'est plus le cas, les plateformes aéroportées peuvent changer leur plan de vol, et se diriger sur une zone afin de récupérer les pistes perdues à cause du brouillage.

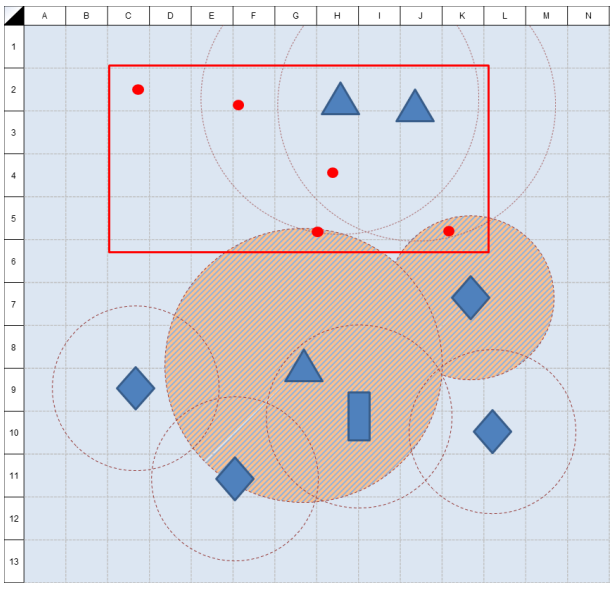

FIGURE 3.5 – Vignette 3 : Brouillage ennemi empêchant l'utilisation de certaines ressources senseurs.

Vignette IV

Dans cette vignette, nous mettons en évidence la dégradation des communications et l'isolation partielle d'une plateforme avec les autres. En effet, une augmentation de la latence entre deux plateformes impacte le choix de nouveaux services ainsi que les retours sur les services en cours sur la plateforme en question. Cela peut parfois impliquer que ces derniers soient interrompus. Les caractéristiques des ressources utilisées pour l'accomplissement de services sont modifiées pour informer les utilisateurs de cette dégradation des services.

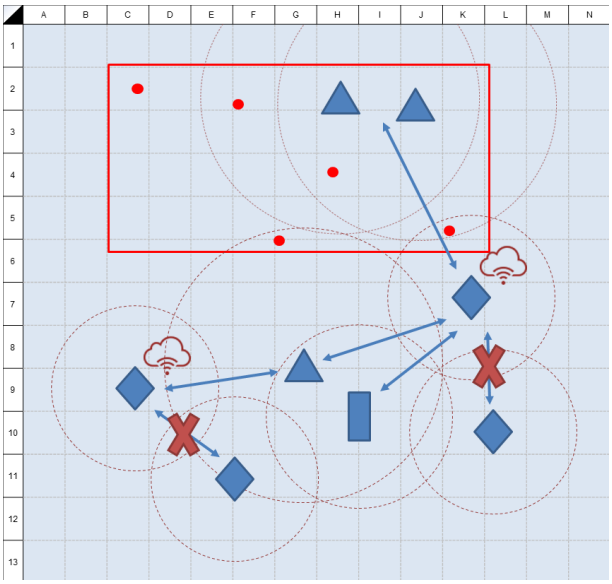

FIGURE 3.6 – Vignette 4 : Dégradation des communications.

La faible connectivité peut compromettre l'exécution d'un service et allonger la durée de sa mise en place. Le système doit donc prévoir les cas où le réseau est dégradé et s'adapter en fonction de la latence du réseau. Chaque plateforme doit posséder un état nominal de fonctionnement, complètement indépendant des autres plateformes. Dans ces conditions, le combat collaboratif apporte de la robustesse en permettant un fonctionnement en mode dégradé, dans le but de poursuivre la mission malgré des services indisponibles ou de faible qualité.

Vignette V

D'après les besoins opérationnels évoqués dans le chapitre un, la préemption de la ressource senseur par un opérateur doit pouvoir être faite à n'importe quel moment. La demande d'un opérateur doit toujours être prioritaire sur le système intelligent, car celui-ci peut posséder des informations que le système ne connaît pas ou ne peut pas comprendre. Lorsque cela arrive, certains services moins prioritaires s'arrêtent et libèrent la ressource nécessaire pour l'opérateur. Le service de l'utilisateur s'octroie une priorité maximale et ne peut donc être interrompu par aucun autre nouveau service. Le système doit donc composer sans la ressource senseur bloquée par l'utilisateur, jusqu'à ce que celle-ci soit libérée.

Sur la vignette, un opérateur présent sur le porte-avions (rectangle orange) réalise une demande particulière de localisation sur une cible que le système avait considérée comme non prioritaire. L'opérateur peut décider lui-même des ressources senseurs qu'il veut déployer, ou bien augmenter artificiellement la priorité de la cible pour forcer le système à employer les meilleures ressources pour remplir cette demande. Ici, nous pouvons voir en orange les messages envoyés aux plateformes concernées par l'annulation d'un service, afin d'utiliser la ressource pour la demande spécifique de l'opérateur.

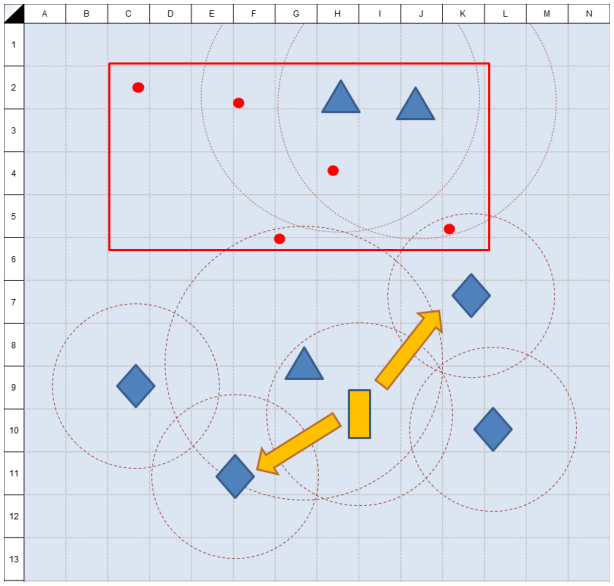

FIGURE 3.7 – Vignette 5 : Demande opérateur d'utilisation de senseurs.

3.2.4 Conclusion

Cette partie introduit les fonctionnalités attendues par le système ainsi que les hypothèses que nous avons posées pour définir le cadre de notre étude. Nous avons proposé une formalisation du contexte et de la problématique en définissant les concepts des services senseurs et des requêtes opérationnelles. Les études d'architecture de système proposent des scénarios d'usage pour définir les événements et les actions, permettant d'évaluer les performances du système. Ainsi, nous avons proposé cinq vignettes pour explorer les différents cas d'usage du système. Ce dernier doit pouvoir recueillir des informations sur chaque cible à l'aide de senseurs en proposant une allocation de tâches adéquate à la situation. Des services collaboratifs utiliseront de multiples senseurs en parallèle pour améliorer la situation tactique. Ensuite, le système doit pouvoir s'adapter à l'interconnexion avec de nouvelles plateformes et à la mise à jour des ressources senseurs communes en maintenant une continuité des services proposés. Inversement, il doit pouvoir faire face à des cas de dégradations des liaisons de communication ou de destructions matérielles, qui engendrent une diminution de choix et de qualité sur les services proposés.

Dans certaines situations, le système devra montrer qu'il peut préempter de manière cohérente les ressources senseurs pour effectuer des services prioritaires, notamment des demandes opérateurs. Cette préemption est complexe, elle suppose d'arrêter l'utilisation en cours de certaines ressources pour satisfaire d'autres services. Du fait de l'architecture distribuée, la perception d'un état global dans le système peut être limitée ou différée, ce qui complique le maintien de la cohérence. Bien que chaque plateforme possède son propre état local, les retards de communication et les contraintes distribuées peuvent entraîner des variances de perception de l'état global. Une ressource pourra être libérée localement, par exemple à la fin d'un service, sans que l'ensemble du système n'en ait la connaissance immédiate.

En complément des scénarios présentés, nous effectuerons des tests capacitaires du système pour démontrer sa scalabilité et ses limites. Nous augmenterons le nombre d'acteurs dans un scénario simple (typiquement la vignette I), d'une part le nombre de menaces et d'autre part le nombre de plateformes alliées. Le besoin en communication pourra être quantifié selon la taille du système.

3.3 Conception de l'architecture du système

3.3.1 Intérêts des SMA

La problématique de l'allocation de tâches à des ressources physiquement distribuées dans un réseau est un problème pouvant être résolu par des techniques multi-agents [SST21] et qui apportent plusieurs points d'intérêt dans le cadre de nos travaux :

- ➤ La décentralisation de la prise de décision dans l'utilisation des ressources pour accomplir des services peut être réalisée par un SMA, car les agents sont des entités autonomes capables de décider localement.
- ➤ Les agents peuvent communiquer avec d'autres agents et s'adapter lorsque la liste des ressources évolue.
- ➤ L'utilisation d'une approche à base d'agents permet également une forte adaptabilité dans un environnement dynamique, où de nouveaux agents peuvent être créés ou supprimés au besoin.
- ➤ Les agents sont autonomes et peuvent avoir des logiques de prise de décisions différentes en fonction du contexte et de l'environnement dans lequel ils se situent.
- ➤ Ensemble, les agents peuvent résoudre des problèmes plus complexes et plus larges que des solutions monolithiques.
- ➤ Lorsque les agents sont communicants, ils permettent d'interconnecter les systèmes les utilisant.

3.3.2 Réflexion sur l'agentification

L'agentification est l'étape de la conception d'un système multi-agent qui a pour objectif d'associer les agents à des rôles. « Un agent est une entité autonome capable de percevoir dans un environnement et d'agir sur celui-ci dans le but d'accomplir ses objectifs » [Woo09]. Il s'intègre dans un SMA pour accomplir un objectif commun à l'aide de processus d'interaction, de coopération, de coordination ou de négociation. L'interaction facilite les échanges d'informations entre les agents, la coopération leurs permet de travailler ensemble, la coordination organise leurs activités et l'utilisation des ressources, tandis que la négociation résout les conflits et favorise le consensus. Dans ce contexte, un rôle est une fonction ou un ensemble de responsabilités attribuées à un agent qui permettent de réaliser les objectifs du système. La méthodologie GAIA permet l'analyse et la conception d'un SMA [WJK00]. Moraïtis et al. proposent de combiner cette méthodolo-

gie avec JADE (*Java Agent Development*), une plateforme de développement multi-agent [MPS02]. En effet, la méthodologie GAIA précise que l'implémentation des agents peut être effectuée sur tout type de framework multi-agent.

L'étape d'analyse de la méthodologie consiste à comprendre le système et sa structure, sans penser à l'implémentation logicielle ou physique de celui-ci. La compréhension du système passe par la visualisation de son organisation. Dans un système, des agents ont des rôles définis par quatre attributs : les responsabilités, les permissions, les activités et les protocoles. Les responsabilités sont les fonctions associées à un rôle. Les permissions sont les ressources disponibles pour remplir les responsabilités. Les activités sont des algorithmes internes et privés des rôles. Enfin, les protocoles sont les moyens d'interactions entre plusieurs rôles.

L'étape suivante, celle de la conception, vise à réduire le niveau d'abstraction proposé lors de la phase d'analyse afin d'y appliquer des techniques de conceptions traditionnelles (dont la programmation orientée objet). Ce processus est décomposé en trois modèles : le modèle agent, le modèle service et le modèle d'interaction. Le modèle agent documente les types d'agents du système et leurs instances. Un type d'agent peut convenir à un ou plusieurs rôles définis lors de l'analyse du système, le nombre d'agents de ce type peut demeurer constant ou varier au sein du système. Le modèle service spécifie les fonctions des agents. Dans un langage orienté objet, un service équivaudrait à une méthode. Une activité identifiée lors de la phase d'analyse correspondra à un service, mais les services proposés par un agent dérivent également des attributs associés aux rôles. Un service est défini par ses entrées, ses sorties et ses contraintes d'utilisation. Enfin, le modèle d'interaction définit les communications que peuvent avoir les agents entre eux sans indiquer le contenu des messages échangés. Un graphe schématise généralement le modèle, où les nœuds correspondent à des types d'agents, et les liens représentent les communications.

Dans de précédents travaux, une première proposition d'agentification suggérait d'associer un agent à un senseur, qui peut communiquer, échanger des données et coopérer. Cependant, cela revient à attribuer un rôle de décisionnaire à chaque senseur. Ainsi, de nombreux échanges asynchrones sont requis pour la coopération entre les senseurs, qui doivent posséder des connaissances conséquentes et mises à jour régulièrement sur l'ensemble des menaces afin de prendre des décisions de traitement. Les ressources et les capacités de calcul sont partagées pour le fonctionnement des senseurs. La vision de l'environnement de chaque senseur n'est que partielle, car la diffusion des connaissances entre senseurs est soumise aux limites du réseau : les échanges entre les senseurs deviennent

denses à mesure que le nombre de senseurs augmente.

En partant du constat que les décisions des agents sont souvent prises pour les menaces présentes sur le théâtre des opérations, la solution finale propose que l'agent joue le rôle d'une cible/piste. Cette agentification allège les contraintes sur la duplication des données sur chaque senseur et sur la densité des échanges. En effet, dans le cas où un agent serait affilié à un senseur, les données de menaces doivent être connues sur chaque senseur pour que celui-ci prenne des décisions localement sur les actions à mener. Les senseurs complètent ces données par des échanges avec les autres senseurs. L'agentification retenue assimile les senseurs à des ressources qui servent les agents dans l'accomplissement de leurs tâches. Cet agent est appelé **agent tactique**, car il possède une connaissance approfondie de la cible qu'il représente, comme sa vitesse ou sa trajectoire. La connaissance de l'ensemble des agents (SMA) représente donc une situation tactique globale. Nous adaptons le concept de l'agent tactique à nos travaux en proposant de l'intégrer au sein d'un système multi-agent d'une architecture distribuée. Nous décrivons son rôle et ses interactions en section 3.4.4.

3.3.3 Etape d'analyse et identification des rôles

Dans notre modèle d'allocation de tâches senseurs, le système consomme des ressources senseurs hétérogènes pour détecter, reconnaître, identifier, localiser et poursuivre (DRIL-P) les différentes menaces sur le théâtre d'opérations. Le système cherche à minimiser le temps mis pour l'allocation des tâches tout en maximisant le nombre de menaces prises en charge de manière satisfaisante.

Nous identifions plusieurs rôles pour notre système, ces rôles sont décrits dans les tableaux ci-dessous, selon la méthodologie GAIA. La propriété *Liveness*, présentée dans ces tableaux, définit le cycle de vie du rôle avec des compositions d'activités qui se répètent. La propriété *Safety* montre les conditions que le rôle doit intégrer pour ne pas être mis en défaut. Les activités sont soulignées et les protocoles ne le sont pas.

La clarification des rôles, incluant leurs descriptions, leurs fonctions, leurs ressources, leurs responsabilités et leurs interactions, permet une compréhension approfondie de la structure interne du système, favorisant ainsi une modélisation plus précise lors de la phase de conception de la méthodologie GAIA. Dans cette seconde étape, les rôles définis doivent à présent être assignés à des types d'agents. Les fonctions de ces agents découlent des activités, des protocoles et des responsabilités. Enfin, les protocoles servent également à identifier les interactions nécessaires entre les agents.

Rôle	Responsable des services
Description	Il contrôle la base de connaissances des services et la partage. Il gère également l'utilisation des services.
Activités et protocoles	MajListService, DiffusionServices, AllocationServices, RequeteListeRessources, ReceptionDemandeServices
Permissions	**Read :** liste des ressources, demandes de services **Write :** liste des services
Responsabilités	**Liveness :** MAJSERVICES = (RequeteListeRessources.MajListService. DiffusionServices)+ ALLOCSERVICES = (ReceptionDemandeServices. AllocationServices)+ **Safety :** On donne ici un invariant du système, par exemple : « L'utilisation des services ne doit pas dépasser la capacité des ressources. »

TABLE 3.1 – Schéma du rôle : Responsable des services.

Rôle	Responsable plateforme
Description	Il contrôle la base de connaissances des plateformes (ressource, position). Il gère également l'utilisation des ressources.
Activités et protocoles	MajRessourcePosition, DiffusionRessources, DiffusionPosition, AllocationRessources, RequetePosition, RequeteRessource, ReceptionDemandeRessources, DiffusionEtatServices
Permissions	**Read :** liste des ressources, demandes de services **Write :** état des ressources
Responsabilités	**Liveness :** MAJCONNAISSANCE = (DiffusionRessources.DiffusionPosition. RequeteRessource.RequetePosition.MajRessourcePosition)+ ALLOCRESSOURCES = (ReceptionDemandeRessources. AllocationRessources)+ **Safety :** Les ressources doivent être libérées une fois le service terminé.

TABLE 3.2 – Schéma du rôle : Responsable plateforme.

Rôle	Responsable des connaissances tactiques
Description	Il contrôle la base de connaissances des cibles. Il doit également améliorer les connaissances.
Activités et protocoles	MajTactique, Planification, RequeteListServices, DiffusionPlanification, RequeteEtatServices
Permissions	**Read :** liste des services, informations tactiques **Write :** informations tactiques
Responsabilités	**Liveness :** MAJTACTIQUE = (RequeteListServices.MajTactique)+ PLANIFICATION = (RequeteEtatServices.Planification.DiffusionPlanification)+ **Safety :**

TABLE 3.3 – Schéma du rôle : Responsable des connaissances tactiques.

3.4 Architecture multi-agent pour l'allocation de tâches senseurs

3.4.1 Conception interne d'un agent

Dans notre approche multi-agent, l'architecture interne d'un agent comporte une mémoire, une messagerie et un ensemble de fonctions. La distribution des agents sur les différentes plateformes constitue une proposition d'architecture permettant l'usage de services multi-plateformes.

L'agent constitue et reconstitue ses connaissances sur lui-même et sur son environnement en communiquant avec ses pairs et avec une base de données externe contenant les capacités et les informations collectées par les senseurs, ainsi que les données opérationnelles (par exemple, des renseignements connus sur une cible potentielle). La mémoire, initialement vide pour chaque agent, se complète avec différentes connaissances : les informations provenant de la plateforme, les requêtes opérationnelles et les données sur les cibles engrangées par les senseurs actifs. Un mécanisme de communication permet à l'agent d'échanger avec d'autres agents dans le but de collaborer et de partager des informations.

Pour remplir les rôles considérés dans la sous-partie précédente, nous proposons 3 concepts d'agents :

- L'agent plateforme a pour rôle de maintenir la base de connaissances de la plateforme qu'il symbolise et des plateformes alliées. Il initialise un agent service et il actualise

sa situation tactique locale par la création et la mise à jour des agents tactiques.

- ➤ L'agent service représente la liste des services que fournit le système, chaque plateforme en possède un. Il a pour objectifs de proposer des services disponibles et de planifier les services en échangeant des messages avec les agents tactiques et les autres agents services.
- ➤ L'agent tactique exprime un objet du théâtre des opérations. Il est donc créé en réponse à la détection d'une piste « senseur ». Dans nos travaux, cet agent suggère des plans senseurs à l'agent service.

Chaque agent possède des fonctions en communs telles que l'initialisation de l'agent, son démarrage ou ses mises à jour de mémoire. Selon le type de l'agent, il pourra utiliser des fonctions plus spécifiques pour remplir son rôle avant de prendre des décisions selon l'évolution des informations qu'il possède. Ensemble, ils permettent d'allouer des tâches à des senseurs dans un contexte multi-plateforme.

3.4.2 L'agent plateforme

L'agent plateforme possède des connaissances sur lui et son environnement, telles que sa position géographique, ses ressources et leur disponibilité, les liens de connexions avec les autres plateformes et des renseignements de missions fournis par les requêtes opérationnelles. Dans le cas de nos plateformes, nous supposons que les agents se connectent directement dans un réseau.

L'agent plateforme communique sa position avec les autres plateformes à intervalle régulier. Cette fréquence d'échanges d'informations influence la qualité des mises à jour et le taux d'utilisation de la bande passante. En effet, à mesure que l'intervalle entre deux échanges diminue, les informations deviennent plus précises et récentes, mais cela sollicite également plus le réseau. Il échange également sur ses ressources senseurs lorsque celles-ci sont modifiées, que ce soit par un ajout ou par une suppression d'une ressource. De plus, les agents plateformes se coordonnent sur le choix des agents tactiques qu'ils prennent en charge, en fonction de la disponibilité des ressources et de la distance relative à la menace considérée. Ainsi, un seul agent tactique concernant une même piste propose des services à effectuer depuis une plateforme. Les plateformes partagent la connaissance de tous les agents tactiques dont ils ont la charge, contribuant à l'élaboration d'un état global. Cela crée de la redondance, car si une plateforme quitte le réseau, les connaissances tactiques de ses agents sont perdues. L'état local d'un agent tactique sauvegardé sur une autre

plateforme permet la récupération de données cohérentes, dont la plateforme nouvellement en charge de l'agent tactique pourra se servir pour poursuivre la mission.

La ressource

Les ressources des plateformes sont nécessaires à l'exécution des tâches. Dans nos travaux, nous considérons uniquement les ressources de type senseurs. Étant donné le caractère hétérogène des senseurs, nous proposons une abstraction du fonctionnement propre d'un senseur en proposant un modèle qui conviendrait à plusieurs types de senseurs. De futurs travaux pourront également compléter ces modèles pour y intégrer des ressources de calcul ou des fréquences d'utilisation, qui sont également nécessaires au bon fonctionnement de l'exécution de certaines tâches.

Une ressource possède donc un nom définissant le type de ressource (R1, R2, R3...), un ID unique, les services que la ressource propose à son environnement, l'énergie que possède la ressource actuellement ainsi que son énergie maximale, sa portée, les services en cours qui utilisent la ressource et leur taux de complétion, le nom de la plateforme à laquelle la ressource est rattachée, et les caractéristiques d'utilisation d'une ressource. Le tableau 4.4 présente un exemple de la structure d'une ressource.

Type	Plateforme	Energie use/max	Services	Services en cours	Caractéristiques
R1	A	2/5	S1, S3	$S1_1$: 35% $S3_1$: 80%	C1=1 C2=5 C3=2
R1	B	1/5	S1, S3	$S1_1$: 35%	C1=1 C2=5 C3=2
R2	A	1/1	S2	$S2_1$: 10%	C1=5 C2=1 C3=1
R3	A	6/10	S2	$S2_2$: 95% $S2_3$: 5% $S2_4$: 50%	C1=4 C2=3 C3=3

TABLE 3.4 – Connaissance de chaque agent plateforme sur l'utilisation de ses ressources.

Le type de la ressource permet de décrire des ressources identiques par leur fonctionnement d'une plateforme à une autre, seul l'ID de cette ressource sera différent. La liste des types de services décrit les services que peut effectuer la ressource, nous proposons

également une abstraction du service. Un service de TDOA, par exemple, sera simplement appelé service *S1*, car nous nous intéressons à l'allocation de la tâche à des senseurs, et non à l'exécution fonctionnelle d'un service de TDOA. Ainsi, une ressource senseur R1 pourrait par exemple réaliser les types de services S1 et S2. L'énergie d'une ressource représente la capacité d'un senseur à pouvoir effectuer plusieurs actions en parallèle, selon la charge imposée par le service qui les utilise. Nous l'avons bien vu avec l'exemple du radar RBE2 AESA dans le premier chapitre. Dans le futur, de plus en plus de senseurs pourront répondre à une multitude de tâches en parallèle. Pour accomplir un service, des précisions sur la ressource utilisée doivent être fournies : la ressource est-elle en cours d'utilisation ? Quels services l'utilisent ? Pour combien de temps ? Enfin, nous avons intégré une notion de caractéristique d'utilisation de la ressource pour spécifier qu'un service est plus intéressant qu'un autre en fonction de la requête opérationnelle. Si la requête demande un niveau de discrétion élevé, nous pouvons l'abstraire par une contrainte nôtée C1 et lui donner un niveau entre 1 et 5. Si la caractéristique C1 est différente de la contrainte C1 (même appellation au niveau du service et de la ressource), alors le service sera moins pertinent à utiliser.

Répondre à la problématique d'interblocage

Une problématique des systèmes distribués apparaît ici. Seul l'agent plateforme en charge de sa propre ressource peut accepter qu'un autre agent l'utilise. De fait, un cas d'interblocage peut se poser si deux plateformes veulent accéder simultanément à deux ressources gérées sur des plateformes différentes dans le cadre d'un service multi-plateforme. Chaque agent pourrait réserver une première ressource et attendre la libération de la seconde pour accomplir son service. Des méthodes pour éviter ces problèmes existent [Ray12]. Par exemple, en imposant un ordre d'acquisition des ressources, un service devra obligatoirement récupérer la ressource R1 avant la ressource R2. Cette solution ne convient pas dans notre cas, en raison des problèmes de latence que peut apporter le réseau. Un processus communicant par message peut accéder rapidement à une ressource R1 et mettre plus de temps à obtenir la ressource R2. Tandis qu'un second processus présenterait peut-être un délai plus court dans l'obtention des deux ressources, bien que la demande de la ressource R1 prenne plus de temps de son côté. Une seconde solution pourrait être de partager un tour de parole, pour que chaque plateforme réserve ses ressources à tour de rôle. Cette solution est cependant incompatible avec le passage à l'échelle de notre architecture, pour chaque plateforme supplémentaire on augmente l'attente de

réception du « token » qui matérialise le privilège pour entamer les échanges et obtenir des ressources. La latence de ces échanges varie d'une plateforme à une autre, si une plateforme a beaucoup de latence et peu de besoins d'utiliser la ressource, elle risque de faire perdre du temps à d'autres plateformes qui auraient déjà pu réserver des ressource disponibles. Une sous utilisation de la ressource peut également apparaître avec l'attente des plateformes.

Pour résoudre la problématique, nous proposons d'allouer les ressources en deux étapes. Une première demande d'allocation permet de vérifier la bonne connectivité avec les plateformes en charge des ressources et de discriminer les services qui ne peuvent pas être effectués, car des requêtes prioritaires occupent déjà la ressource. La seconde étape est la demande d'allocation réelle des services qui ont fait l'objet d'une première acceptation. C'est une sollicitation d'accès à la ressource en multicast où le processus doit avoir une réponse de tout le monde pour accéder à une section critique. Le détail du mécanisme d'allocation est décrit dans la sous-partie suivante.

Nous utilisons un modèle d'enchère qui sera détaillé en partie 3.4.4. Nous verrons que chaque agent propose une mise sur les différents plans proposés qui correspond à une valeur unique pour chaque agent, et que les plans peuvent être préemptés au profit de demandes plus prioritaires. Ainsi, la prédemande d'allocation permet d'obtenir un premier état des services indisponibles et des problèmes de connectivités qui peuvent apparaître.

Allocation de ressources par l'agent plateforme

L'agent plateforme a également un rôle de décisionnaire, il effectue l'allocation réelle des ressources en fonction des demandes qu'il reçoit. Comme tous les agents de nos travaux, il fonctionne selon une machine à état présentée en figure 3.8. L'initialisation met en place les canaux d'échanges entre l'agent et ses pairs. La mise à jour des données en mémoire de la plateforme concerne les positions, les ressources et les requêtes opérationnelles des plateformes. Ensuite, il exécute les processus et fonctions des agents tactiques. En parallèle, l'agent plateforme écoute sur des canaux de communication, s'il n'a rien reçu, il continue de mettre à jour ses données et d'exécuter les agents tactiques.

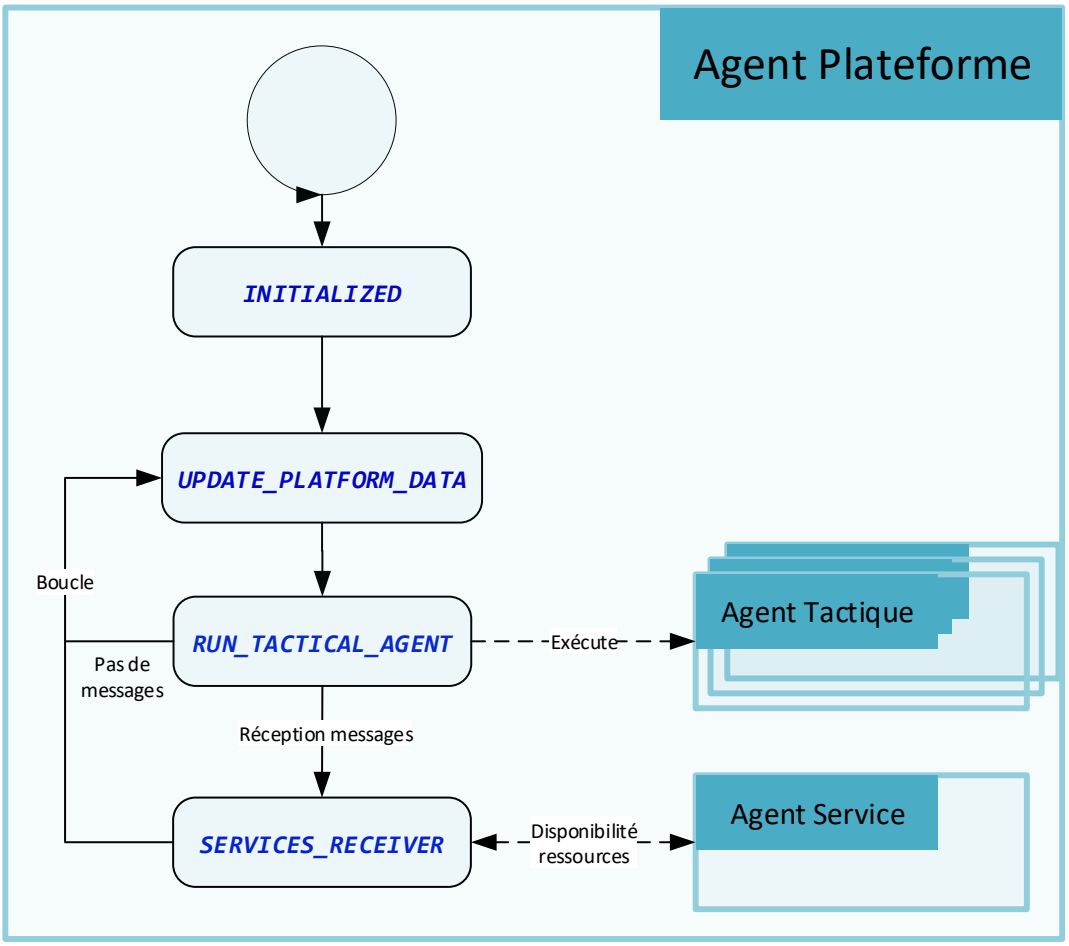

FIGURE 3.8 – Machine à état de l'agent plateforme.

Lorsqu'il reçoit une demande d'allocation de tâches, il vérifie la disponibilité des ressources avec l'algorithme 1. Celui-ci prend en entrée la liste de plans obtenue et renvoie en sortie l'état d'acception de chaque plan qui sera envoyé à l'agent service. C'est un premier échange qui a lieu pour faire une prédemande d'utilisation de la ressource. L'algorithme se répète une deuxième fois pour une réservation de service finale, des plans peuvent également être annulés. Pour annuler un plan au profit d'un autre, l'algorithme vérifie la différence de performance des plans puis la compare à un seuil qui dépend du coût de l'annulation du service.

Algorithme 1: *checkIfResourceIsAvailable*, Vérification de la disponibilité des ressources.

Input: Une liste de plans $\mathcal{P}$ proposée par les agents services
Output: Une liste de plans $\mathcal{P}_{return}$ avec l'état des plans modifiés par rapport à $\mathcal{P}$

```
1  P_return ← ∅
2  foreach p ∈ P do
3      E_d ← Energie disponible
4      E_m ← Energie max initiale
5      E_p ← Energie requise pour le plan p
6      foreach r ∈ p do
7          if E_m >= E_p then
8              if E_d >= E_p then
9                  p ← setPlanStatus(OK)
10                 Add(p , P_return)
11             else
12                 P_r ← {Les plans actifs sur la ressource r}
13                 foreach p_actif in P_r do
14                     C_p ← getPercentageCompletion(p_actif)
15                     diff_perf = getPerformance(p) -
                         getPerformance(p_actif) + C_p
16                     if diff_perf>threshold then
17                         p ← setPlanStatus(OK)
18                         Add(p , P_return)
19         else
20             p ← setPlanStatus(KO)
21             Add(p , P_return)
22 return P_return
```

3.4.3 L'agent service

L'agent service est un agent présent sur chaque plateforme. La machine à état de la figure 3.9 décrit son fonctionnement. Il est responsable des services du système, de leur diffusion et de l'allocation locale de ces services.

Un service a besoin de ressources pour fonctionner, l'agent service va donc récupérer la liste des ressources de l'ensemble des plateformes et créer une table interne de services uniques comme présentée dans le tableau 3.5. Il mettra à jour cette table en ajoutant les nouveaux services, en retirant les services qui n'existent plus et en modifiant les caractéristiques des services s'ils évoluent.

L'algorithme 2 permet de créer la liste de toutes les combinaisons de services réalisables en se basant sur les hypothèses suivantes : un service utilise obligatoirement des ressources

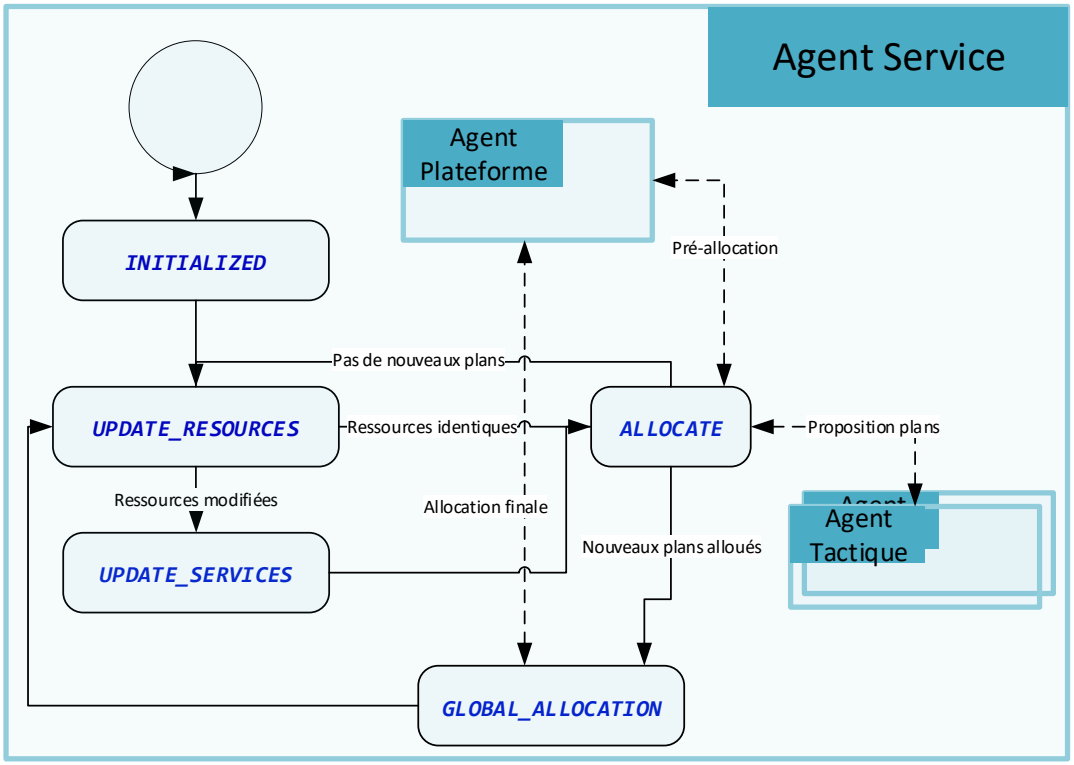

FIGURE 3.9 – Machine à état de l'agent service.

Service	Performance	Ressources	Type	Durée	Energie
$S1_1$	550	R1	Localisation	2s	1
$S1_2$	530	R2	Localisation	2s	1
$S2_1$	500	R2, R3	Identification	5s	2
$S3_1$	580	R1,R4	Détection	1s	1

TABLE 3.5 – Connaissance de l'agent sur les services disponibles.

de plateformes différentes pour se focaliser sur l'aspect collaboratif multiplateforme des services, et un service utilise au plus deux ressources afin d'éviter l'explosion combinatoire présentée sur la figure 3.10.

Algorithme 2: *updateServices*, Mise à jour de la liste des services de l'agent.

Input: Une liste de ressources $\mathcal{R}$
Output: Une liste de services $\mathcal{S}$

```
1  S ← ∅
2  T_s ← Liste des types de services
3  foreach t ∈ T_s do
4      n_r ← Nombre de ressources de R pouvant réaliser t
5      n_s ← Nombre de ressources nécessaires à la réalisation du service
6      switch n_s do
7          case 1 do
8              for i = 0 to n_r do
9                  resource ← R[i]
10                 performance ← getPerformance()
11                 service ← newService(performance, resource)
12                 Add(service , S)
13         case 2 do
14             for i = 0 to n_r do
15                 for j = 0 to n_r do
16                     resources ← R[i][j]
17                     performance ← getPerformance()
18                     service ← newService(performance, resources)
19                     Add(service , S)
20 return S
```

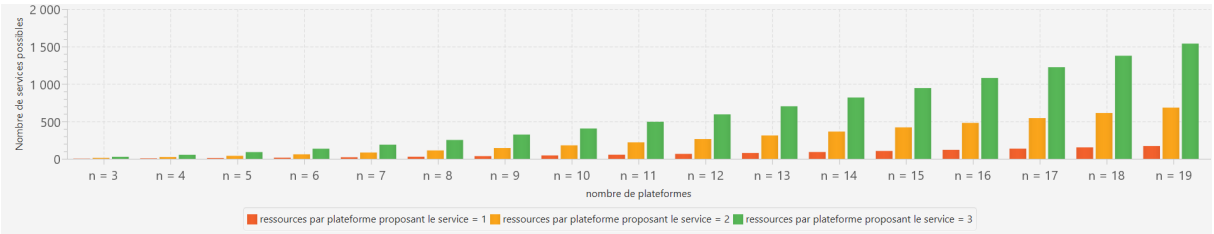

FIGURE 3.10 – Combinaison de services pour deux ressources de plateformes différentes.

Mathématiquement, la formule 3.1 définit le nombre de combinaisons de ressources possibles pour un service S1 :

$$\binom{n}{k} * r^k \tag{3.1}$$

Avec n le nombre de plateformes, k le nombre de ressources pour accomplir un service et r le nombre de ressources disponibles sur chaque plateforme pour un même service.

Par exemple, si un service S1 nécessite deux ressources et que deux ressources peuvent l'accomplir sur un ensemble de dix plateformes alors 180 combinaisons du service S1 sont possibles ($S1_1, S1_2..., S1_{180}$). L'ensemble des plateformes propose un nombre important de services qui augmente avec le nombre de plateformes permettant de mettre en avant l'aspect de mise à l'échelle du système et l'impact sur les prises de décisions des agents. Dans la réalité, le nombre de services proposés pourra toutefois être plus restreint. Cependant, la quantité de services disponible pourra croître à mesure que les systèmes de senseurs évoluent, cette possible évolution doit être prise en considération dans notre approche architecturale.

Chaque nouveau service de la liste de services de l'agent se verra assigner une valeur que nous appelons « performance ». Cette valeur dépend de plusieurs métriques : la localisation des plateformes qui embarquent les ressources requises, les ressources considérées et leurs caractéristiques propres, le nombre de ressources pour effectuer le service. Chaque métrique se voit accorder un poids selon son importance et son impact sur la qualité d'un service. Ainsi, la distance entre les plateformes est mesurée, dans certains cas le service ne pourra pas se faire, en raison de l'éloignement des plateformes. Cette notion de performance a l'avantage de pouvoir être ajustée et complétée par de nouvelles métriques, ces dernières pourront aussi avoir des modèles plus complexes tenant compte de multiples paramètres. Dans nos travaux, la performance assignée à chaque service nous permet de sélectionner les meilleurs services selon la situation et ainsi réduire la charge de la fonction d'allocation de tâches. Nous verrons que l'agent tactique procède également à un calcul de score qui dépend de cette performance et de connaissances supplémentaires propres à la cible, notamment sa position.

L'agent service partage localement sa table de services avec les agents tactiques colocalisés. En retour, les agents tactiques calculent et proposent des plans à l'agent service qui planifiera les meilleurs en fonction des ressources potentiellement disponibles. L'agent service peut être vu comme un « commissaire-priseur » d'un algorithme de coordination par enchère. En effet, il obtient des demandes d'allocation de manière centralisée tout en étant distribué sur différentes plateformes. Le score qu'il reçoit correspond donc à la mise que l'agent tactique lui propose. La phase de l'allocation locale (ALLOCATE sur la figure 3.9) se termine lorsque chaque agent tactique possède un plan d'accepté, ou lorsque la ressource est épuisée. Cette phase est présentée dans l'algorithme 3.

Algorithme 3: *resourceAllocation*, Allocation locale des ressources.

Input: Une liste de plans $\mathcal{P}$ proposée par les agents tactiques
Output: Une liste de plans locaux acceptés $\mathcal{P}_{accept}$

```
1  P_accept ← ∅
2  Sort(P)
3  foreach p ∈ P do
4      if getAgent(p).isNotSatisfied then
5          R_p ← p.getResourcesNeeded()
6          R ← agent.getResources()
7          for i = 0 to R_p.size() do
8              if R[i].energy >= R_p[i].energyNeeded then
9                  serviceOk ← true
10             else
11                 serviceOk ← false
12                 p.setPlanStatus(REFUSED)
13                 Stop(for)
14         if serviceOk then
15             for i = 0 to R_p.size() do
16                 R[i].energy ← R[i].energy − R_p.energyNeeded
17                 Agent(p).setSatisfied(true)
18                 p.setPlanStatus(ACCEPTED)
19         Add(p, P_accept)
20 return P_accept
```

L'agent service échange ensuite sur cette première allocation avec les agents plateformes considérés pour les services retenus, il réitère son allocation pour les plans refusés. La phase suivante, celle de l'allocation globale (GLOBAL_ALLOCATION), consiste à utiliser directement auprès des agents plateformes leurs ressources pour accomplir les services. La différence ici est que les agents services sont en concurrence pour réserver la ressource avec les agents services des autres plateformes. Un plan peut alors se retrouver dans plusieurs cas de figure : accepté, refusé, annulé ou terminé. Un plan reste à l'état accepté tant qu'aucun autre plan plus important ne l'annule, ou jusqu'à ce que le service se termine. Un plan peut être accepté localement puis refusé lors de l'allocation globale, car moins prioritaire que d'autres. Dans tous les cas, l'agent service informe les agents tactiques sur le statut de leurs plans, ceux-ci retournent dans un état de planification s'ils n'ont plus de plans acceptés (équivalent à des plans annulés, terminés ou refusés). Un délai est également apposé sur chaque plan, qui sera refusé passé ce délai.

3.4.4 L'agent tactique

À chaque piste détectée est associé un agent tactique, celle-ci contient des données caractérisants un objet ou une cible sur le théâtre des opérations. La mémoire interne de l'agent comprend des informations sur la position, l'attitude et la vitesse de la cible, ses intentions pour compléter les connaissances de la piste et des informations d'identification (si disponibles). La machine à état de la figure 3.11 décrit son fonctionnement. Les agents tactiques fonctionnent par requête opérationnelle, les besoins pour la mission vont influencer les choix de leurs plans. Par exemple, un opérateur a défini une zone de veille, si la menace détectée se situe hors zone alors l'agent tactique sera créé, mais inactif, et ce jusqu'à ce qu'une nouvelle requête lui ordonne le contraire.

L'agent tactique propose d'accomplir des objectifs opérationnels selon la suite logique nommée DRIL que nous avons présenté précédemment. Ainsi, le choix des plans senseurs différera en fonction de l'objectif opérationnel en cours. Par exemple, si l'agent a réussi à identifier la cible, son objectif suivant sera de la localiser. L'agent ne propose plus de plans quand son objectif final est atteint. L'agent tactique approfondira ses connaissances sur la menace grâce à l'utilisation des senseurs, le niveau de priorité dépendra de la criticité de la menace.

La détection d'une cible est possible sur deux plateformes en simultané, ce qui créera deux agents tactiques qui soumettront des plans pour accomplir des objectifs proches en fonction des connaissances acquises. Dans la réalité, une corrélation entre les pistes doit être réalisée afin de déterminer les correspondances entre les cibles. Dans le cas de nos travaux, nous considérons une corrélation parfaite et des agents tactiques uniques seront exécutés sur chaque plateforme.

L'agent tactique va étudier la faisabilité de plans senseurs en fonction des services proposés par l'agent service. Pour cela, il va calculer une valeur que nous appellerons « score » pour la différencier de la performance des services vus précédemment. Ce score sera affecté à chaque plan et dépendra de la performance du service, de la position des plateformes possédant la ressource par rapport à la cible, de la priorité de la requête opérationnelle, des contraintes imposées par cette requête (une requête avec une contrainte de discrétion aura un score faible si les ressources proposées par le service ne sont pas discrètes également) et de la satisfaction de l'agent (un agent non satisfait à plusieurs reprises verra le score de ses plans augmenter).

Ce dernier point est important, car il permet d'éviter les situations de famines présentes dans les systèmes distribués non équitables. Un système est dit non équitable s'il

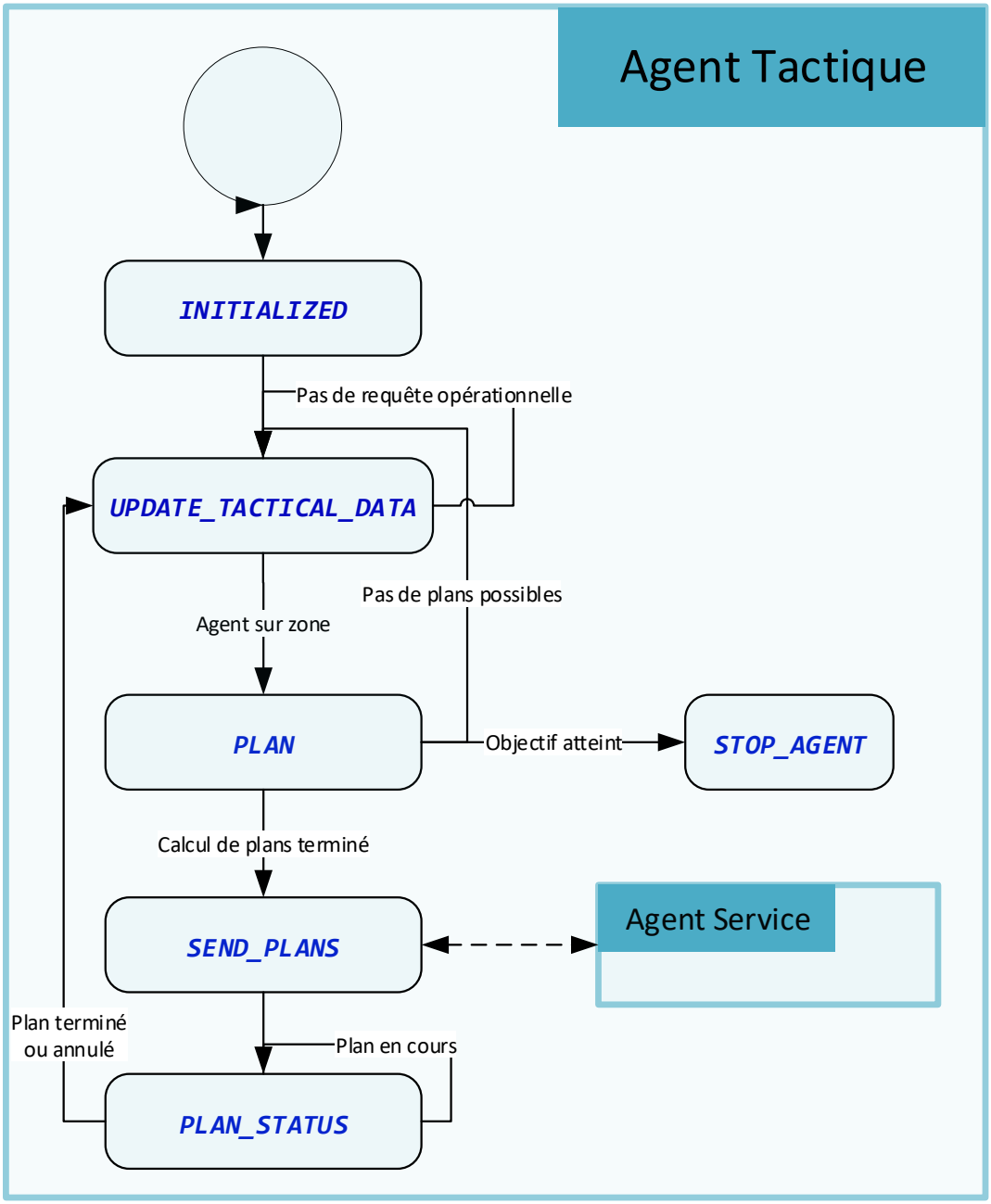

FIGURE 3.11 – Machine à état de l'agent tactique.

ne garantit pas que chaque processus puisse accéder à une section critique avec une probabilité non nulle d'y parvenir en un temps fini [CDKB05]. Notre système favorise les agents tactiques prioritaires. Cependant, certains agents pourraient ne jamais être pris en charge. En augmentant artificiellement le score des plans proposés par un agent insatisfait,

celui-ci verra obligatoirement un de ses plans acceptés. Les cibles peu importantes ou avec des propositions de plans de faible qualité utiliseront donc très peu la ressource laissant plus d'opportunité aux tâches les plus essentielles. La priorité de l'agent est réévaluée afin de tenir compte de la situation et de son environnement, donc son traitement reste important.

Les plans senseurs que l'agent tactique soumet comprennent : le service en question, le score calculé, le moment où le plan a été calculé, et les ressources nécessaires à sa réalisation. Cette étape correspond à l'algorithme 4.

Algorithme 4: *calculatePlan*, Calcul de plans par l'agent tactique.

Input: Une liste de services $\mathcal{S}$ proposée par l'agent service
Output: Une liste de plans $\mathcal{P}$

```
1 S ← sélection services : performance et objectif
2 P ← ∅
3 foreach s ∈ S do
4     score ← getScore()
5     if score > 0 then
6         p ← s
7         p.setScore(score)
8         Add(p,P)
9 return P
```

L'objectif de l'agent est de compléter ses données sur la cible afin d'améliorer sa connaissance tactique sur celui-ci. Dans ce but, il va fournir son ou ses meilleurs plans senseurs à l'agent service selon un algorithme d'enchère. L'agent tactique adopte le comportement d'un algorithme glouton, il ne se coordonne pas avec les autres agents avant d'envoyer ses plans et il ne sait pas si ses plans seront acceptés, mais il maximise son propre score. Nous supposons que l'agent planifie une seule utilisation de service à la fois, il peut présenter un nouveau plan senseur dès que son plan est terminé ou annulé. Une fois les plans envoyés, l'agent tactique se met en état d'attente du statut des plans et exécute l'algorithme 5 tant qu'il est satisfait, puis il retourne planifier. Le processus d'enchère entre les agents tactiques et les agents services est détaillé dans la sous-partie suivante.

Algorithme 5: *checkPlanStatus*, Vérification du statut des plans en cours.

Input: La liste des plans $\mathcal{P}$ envoyés par un agent tactique

```
1  agentState ← PLAN_STATUS
2  P_refus ← ∅
3  foreach p ∈ P do
4      switch status do
5          case ACCEPTED do
6              agentSatisfied(true)
7          case REFUSED do
8              p.setLastTimeRefused(time)
9              Add(p,P_refus)
10         case CANCELED do
11             p.setLastTimeRefused(time)
12             Add(p,P_refus)
13             agentSatisfied(false)
14         case TERMINATED do
15             updateAgentKnowledge()
16             agentSatisfied(false)
17 if !agentSatisfied then
18     agentState ← UPDATE_TACTICAL_DATA
```

Allocation par enchère

L'objectif d'une allocation par enchère dans un SMA est de parvenir à une répartition efficace de tâches à un ensemble d'agents en ayant pour contrainte des limitations sur l'utilisation des ressources disponibles. Les enchères permettent de déterminer la valeur que chaque agent attache à une tâche particulière. Chaque agent cherche à maximiser sa fonction d'utilité individuelle tout en cherchant à optimiser une fonction d'utilité globale, qui est la somme des utilités de chaque agent (voir 1.2). Cependant, dans un environnement concurrent, la sélection d'une tâche par le commissaire-priseur peut bloquer des ressources pour d'autres tâches, ce qui peut entraîner des situations où l'optimisation de l'utilité globale n'est pas garantie. Nous avions introduit les algorithmes d'enchère en section 2.4.3. En règle générale, une enchère se déroule en deux phases :

- **L'annonce :** le commissaire-priseur diffuse des articles à vendre. Puis, les enchérisseurs misent sur les articles.
- **La détermination du vainqueur :** une fois toutes les mises reçues, le commissaire-priseur décide du gagnant et le transmet à tout le monde.

Dans le cas de notre problématique, les articles correspondent à des tâches qui doivent

être allouées en optimisant l'utilisation des ressources disponibles. L'agent service joue le rôle de commissaire-priseur qui diffuse la liste des services (articles) aux agents tactiques (les enchérisseurs) qui proposent des offres sur chaque service pouvant être accompli. L'agent tactique envoie uniquement un ou plusieurs meilleurs plans. L'agent service reçoit toutes les offres et les classe par ordre de valeur, il choisit autant de vainqueurs que de ressources disponibles. Une fois qu'un agent tactique a gagné une enchère, il se met en attente jusqu'à l'accomplissement ou l'annulation du service, et la ressource est localement indisponible pour les autres plans. Cette première enchère est synchrone, car l'agent service est centralisé sur une plateforme et il attend une mise de chaque agent tactique de la plateforme.

Algorithme 6: *Enchère.*

```
   En bleu l'agent service s (auctioneer/bidder)
   En rouge chaque agent tactique t (bidders)
   En vert l'agent plateforme p (auctioneer)
 1 repeat
 2     S ← s.updateServices(R)
 3     s.broadcast(S)
 4     if t.receive(S) then
 5         P ← t.calculatePlan(S)
 6         t.reply(P)
 7     if s.receive(P) then
 8         P_accept ← s.resourceAllocation(P)
 9         s.sendAllocationToPlatforms(P_accept)
10     if p.receive(P_accept) and p.timeout(true) then
11         P_return ← p.checkIfResourceIsAvailable(P_accept))
12         p.reply(P_return)
13     if s.receive(P_return) then
14         s.broadcast(P_return))
15     if t.receive(P_return) then
16         while status == ACCEPTED do
17             t.checkPlanStatus()
18 until Objectifs atteints;
```

Une seconde enchère a lieu entre les différents agents services, cette fois ce sont les meilleurs plans de chaque agent tactique qui sont mis en concurrence pour l'accès à la ressource. Les agents services travaillent en parallèle, un agent peut avoir fini son allocation locale bien avant un autre. Dans ce cas, il n'est pas concevable d'attendre que tous les agents services terminent leurs enchères puis se synchronisent, cela prendrait trop de temps dans un contexte critique. C'est pour cette raison que les plateformes effectuent

une enchère toutes les quelques millisecondes, afin de recevoir plusieurs offres. L'enchère suivante peut toujours préempter les ressources d'une offre déjà gagnée, dans ce cas l'agent service est averti. Ce système d'enchère, détaillé dans l'algorithme 6, résout les problèmes de concurrence entre les agents. Les agents se coordonnent sur l'utilisation de la ressource en communiquant leurs demandes.

3.5 Communication inter-agent

L'aspect distribué de la solution architecturale suppose que les agents échangent des données de manière synchrone ou asynchrone, par le biais d'une base de données partagée ou d'un canal de communication. Nous avons vu dans la partie précédente que les agents devaient interagir pour proposer des allocations de tâches. Nous verrons dans un premier temps les échanges effectués entre les agents pour que le système puisse fonctionner, ainsi que les problématiques des communications dans un système distribué. Enfin, nous proposons d'utiliser le middleware RabbitMQ pour prendre en charge les communications entre les agents, ainsi que Prometheus et Grafana pour monitorer le système et observer l'utilisation de la bande passante et l'évolution des latences.

3.5.1 Echanges entre les agents du système

La figure 3.12 présente les différentes interactions possibles entre les agents du système. Nous pouvons observer que tout échange entre la plateforme A et la plateforme B passe obligatoirement par un réseau. Les échanges entre les agents services, tactiques et plateformes situés sur une même plateforme peuvent passer par une base de données locale. Des verrous contrôlent l'accès à cette base de données afin de garantir que les opérations de modification sont exécutées de manière cohérente et ordonnée. Cela empêche les conflits d'accès qui pourraient entraîner des incohérences ou des corruptions des données.

Les échanges indirects : canal de communication

Les agents plateformes s'échangent régulièrement des données sur leurs ressources, leur position et leurs pistes en vue de partager leur situation tactique. Cela permet d'avoir de la redondance sur les informations tactiques, mais également de la résilience en vérifiant la présence des plateformes dans le réseau. Une élection de la plateforme en charge de la menace est effectuée afin de déterminer sur quelle plateforme sera exécuté chaque agent

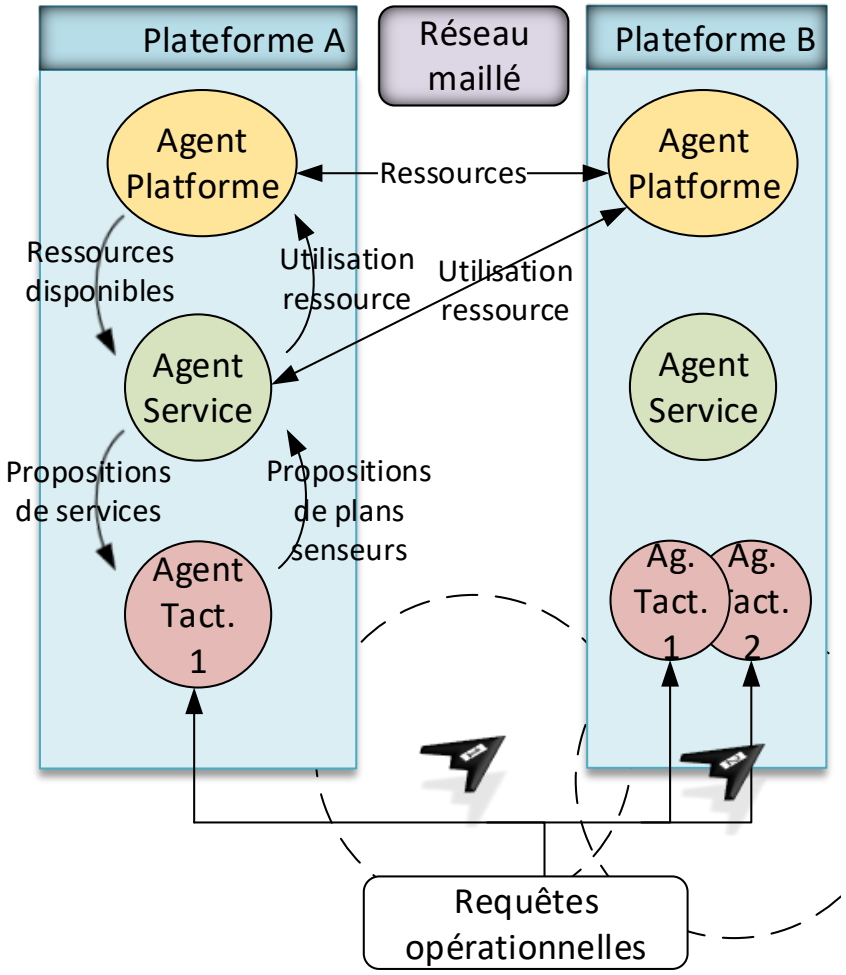

FIGURE 3.12 – Schéma d'échanges entre les agents du système.

tactique. Cette élection prend en compte la distance entre la plateforme candidate et la piste, ainsi que la charge de calcul disponible. Enfin, les plateformes peuvent avoir des requêtes opérationnelles différentes qui devront être partagées afin que les agents tactiques en soient informés. Tous les ordres de mission peuvent également provenir d'une même plateforme, dans ce cas les requêtes opérationnelles sont diffusées à toutes les plateformes. Si une plateforme n'envoie plus de messages après un certain temps, les autres plateformes la considèrent hors réseau, les ressources sont alors supprimées, certains services sont annulés et recalculés. Nous définissons plusieurs interfaces de communication entre les agents plateformes *sendMessageResource*, *sendMessagePosition* et *sendMessageRequest*, qui transmettent des informations à intervalle régulier, ou plus fréquemment quand la

donnée est modifiée.

Les agents services échangent à plusieurs reprises avec les agents plateformes dans une communication bidirectionnelle, afin d'utiliser les ressources des plateformes dans le cadre de services collaboratifs. La fonction *sendMessagePreReservation* permet d'envoyer des messages de pré-réservation de services. Les agents plateformes répondront à l'aide de l'interface *replyToPreReservation*. Ensuite, les agents services proposent une réservation effective des ressources avec la fonction *sendMessageRealReservation*. L'agent plateforme ne répond à ce message que lorsqu'un plan est annulé, réfusé ou terminé grâce à la fonction de communication *sendMessageServiceUpdate*. Un agent service va regrouper, pour chaque plateforme concernée, les plans qui lui sont proposés par les agents tactiques. Chaque ensemble de plans sera donc envoyé aux plateformes concernées selon le schéma présenté en figure 3.13. Ce schéma démontre la complexité des échanges à mesure que le nombre de plateformes augmente. Ainsi les agents services et les agents plateformes, à la fois producteurs et consommateurs de messages, ouvrent N-1 canaux de communication où N correspond au nombre total de plateformes. Une production de messages excessive peut surcharger le destinataire des messages qui répondra avec un délai plus important. Cette complexité est détaillée dans la sous-section 3.6.1 pour chaque fonction de communication.

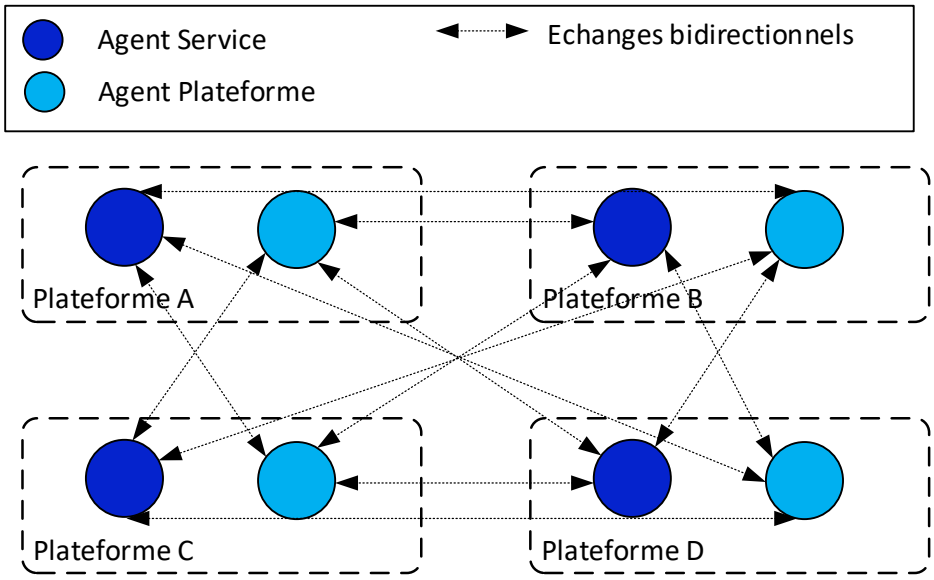

FIGURE 3.13 – Lien de communication entre les agents services et les agents plateformes pour la réservation des services.

Dans le cadre de services nécessitant l'accès à deux ressources de plateformes distinctes,

l'agent service devra utiliser deux canaux de communication et attendre un retour des deux plateformes pour confirmer la possibilité d'effectuer le service. La figure 3.14 présente les différents échanges entre les agents services et les agents plateformes dans le but de réserver la ressource. L'agent service envoie l'ensemble des services qu'il demande à chaque plateforme concernée puis attend un retour sur la disponibilité de sa ressource.

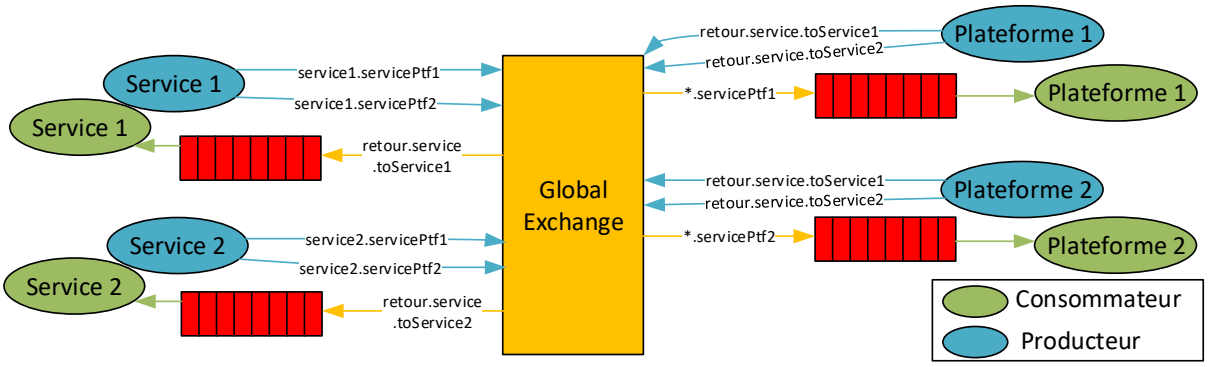

FIGURE 3.14 – Détail d'un échange entre les agents services et les agents plateformes pour la réservation de services.

L'agent plateforme peut aussi communiquer lorsque des plans en cours sont annulés, si par exemple un meilleur plan nécessite l'utilisation des ressources. L'agent service écoute les messages de ce type pour annuler le service en cours, il prévient également la seconde plateforme si nécessaire pour libérer la ressource.

Les échanges directs : mémoire partagée

Une fois que la liste des ressources de la flotte est mise à jour, l'agent plateforme peut partager cette connaissance avec les agents services et les agents tactiques locaux par le biais d'une mémoire partagée. L'agent service a, par exemple, besoin de connaître l'état des ressources disponibles pour établir sa table de services. L'agent tactique pourra récupérer la liste de services proposée directement dans la mémoire partagée pour calculer ses plans senseurs, qu'il mettra également à disposition dans la mémoire pour que l'agent service y ait facilement accès. Il pourra par ailleurs vérifier l'avancement de l'état de ses plans et décider s'il doit de nouveau proposer des plans ou non. La figure 3.15 schématise l'ensemble des échanges locaux entre l'agent service et les agents tactiques d'une même plateforme.

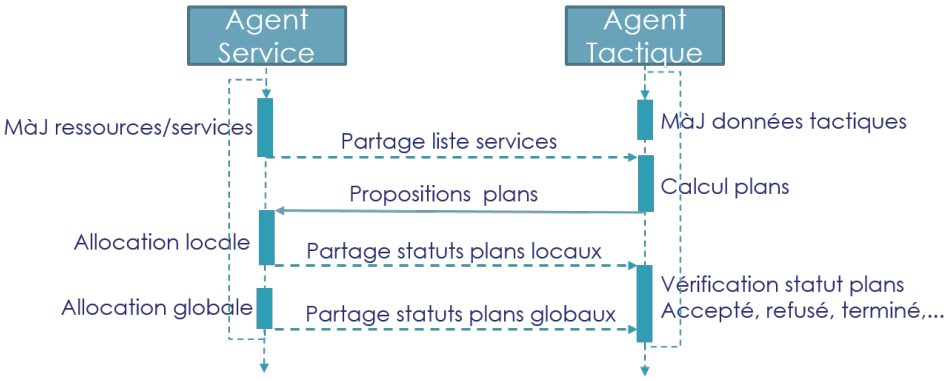

FIGURE 3.15 – Exemple simplifié d'un échange entre un agent tactique et un agent service.

3.5.2 Problématique de la latence

La latence peut apparaître à plusieurs instants. Lorsqu'un agent service souhaite réserver deux ressources sur deux plateformes différentes, le délai avant réception du message par chacune peut varier. L'agent service aura par exemple reçu le premier retour rapidement et le second bien plus tard, si la ressource a déjà été affectée, elle bloque un autre service qui aurait pu se faire. La figure 3.16 permet d'appréhender la problématique des blocages due aux différentes latences présentes dans le système. Nous pouvons y observer que les agents services C et D envoient des messages pour réserver une ressource sur chacune des plateformes A et B. Les ressources des plateformes A et B sont bloquées pour des services différents, ce qui fait que lorsque la seconde ressource du service doit être utilisée, celle-ci n'est plus disponible et le service ne peut donc pas s'effectuer correctement.

La figure 3.17 présente un schéma d'allocation avec demande d'accès concurrent à la ressource. Chaque agent service sur sa plateforme fait une demande pour un même service à un instant T_0 puis à un instant T_1. Cette demande concerne deux avions, avec deux ressources nécessaires à l'exécution du service. L'avion 3 reçoit la demande de la plateforme 1, la ressource n'est pas réservée tant que le service n'est pas exécuté. La plateforme 2 fait de même plus tard. Si le délai d'acheminement dépasse une seconde, le message peut être rejeté en raison de la latence importante du réseau. Ce schéma correspond à la première demande d'allocation qui vient avant l'allocation réelle, elle permet notamment d'évaluer la liaison de communication et de vérifier la disponibilité des ressources sur les deux plateformes avec un premier échange.

Plusieurs cas de figures peuvent se présenter.

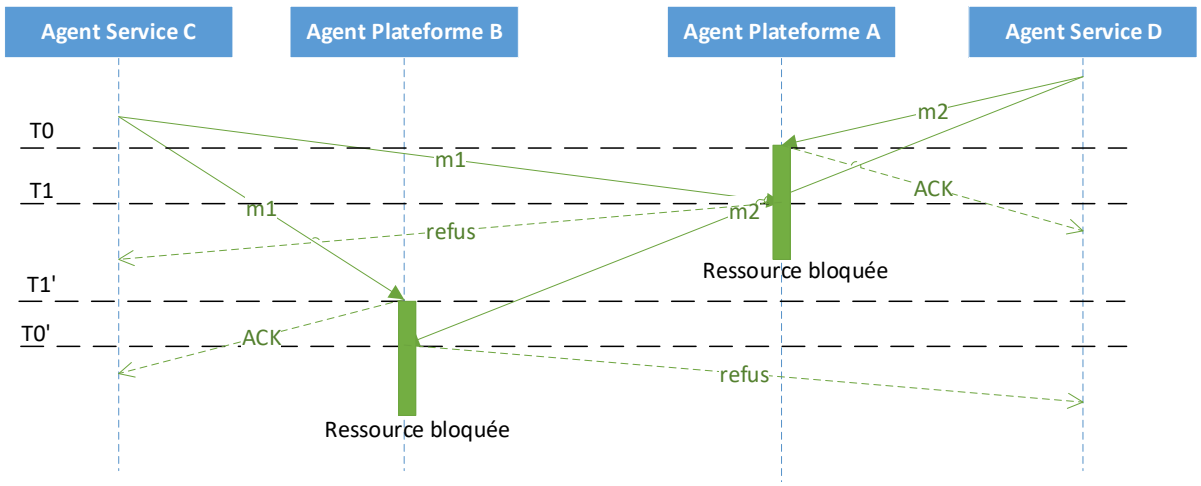

FIGURE 3.16 – Diagramme séquence de la réservation d'une ressource pouvant générer un interblocage.

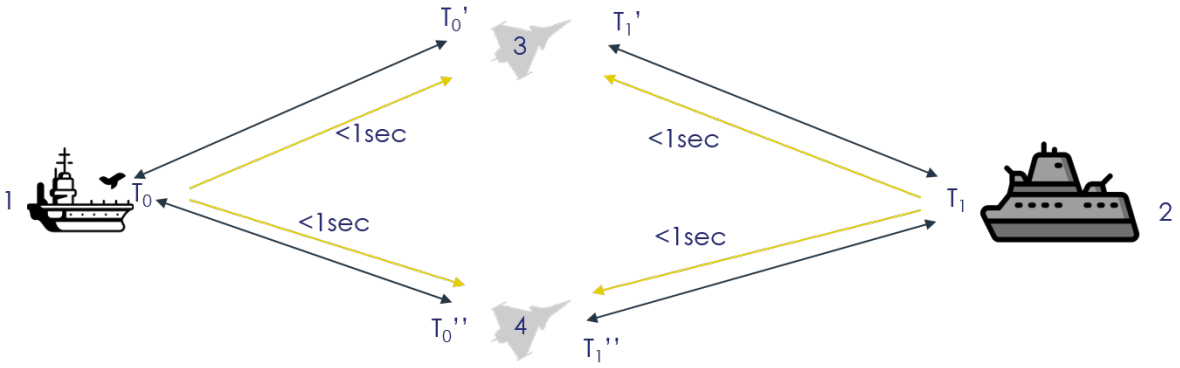

FIGURE 3.17 – Allocations concurentielles de services et impact de la latence.

La plateforme numéro 2 a reçu deux retours positifs pour un service demandé avant la plateforme 1. Le service est donc déjà en cours lorsque la plateforme 1 reçoit son premier retour positif. En raison de la latence, la plateforme 1 n'est pas au courant que la ressource est occupée et va donc demander d'utiliser réellement les ressources.

- ➤ Le service de la plateforme 1 a un score calculé plus important au moment de déterminer si la ressource peut être préemptée. Le service préempte la ressource et informe l'agent service de la plateforme 2 qui vérifie ensuite l'annulation conjointe sur l'avion binome.
- ➤ Le service de la plateforme 1 est de priorité équivalente, dans ce cas il doit regarder le

score du service en cours. Étant donné qu'il est déjà lancé, son score aura augmenté, un service proche de la fin ne devrait pas être annulé car cela couterait cher de l'annuler. Par conséquent, si le score est plus élevé, il faut également regarder le coût d'annulation du service en cours par rapport à la différence de score. On peut estimer que certains services sont moins couteux à arrêter que d'autres.

- ➤ Le service de la plateforme 1 est moins prioritaire, dans ce cas il est tout simplement annulé.

La première demande de la plateforme 2 arrive sur la plateforme 3 alors que la ressource est déjà occupée.

- ➤ Le service de la plateforme 2 a un score plus important, il peut potentiellement préempter la ressource. Il obtient donc un retour positif pour faire une réelle demande d'allocation de la ressource, à condition que la seconde ressource soit également disponible pour le service en question.
- ➤ Le service de la plateforme 2 a un score plus faible, dans ce cas le service est simplement refusé et n'entraîne donc aucune annulation. Il vérifie sur l'autre avion que la ressource n'a pas été réservée pour ce même service sinon l'annule.

Dans notre système, certains processus ou certaines demandes peuvent être plus importants et prioritaires que d'autres. Si ces processus demandent la même ressource et qu'elle ne peut pas être partagée, dans ce cas, le processus moins urgent peut retarder le plus prioritaire en gardant la ressource, c'est ce qu'on appelle l'inversion de priorité ou *priority inversion* [BMS93]. Il y a plusieurs approches pour y répondre [Wu17], la plus simple étant de forcer les processus moins importants à relâcher la ressource lorsque cela est nécessaire, cette condition s'appelle la « préemption ».

Les scores que nous proposons permettent d'exécuter des plans plus importants. Nous avons vu que nous augmentions artificiellement le score des agents tactiques non satisfaits afin d'éviter des problèmes de famines. La satisfaction de chaque agent tactique est importante, un manque de mise à jour de l'information peut présenter un défaut. Par exemple, une cible lointaine peut être considérée comme peu importante, notamment car les ressources à disposition n'apporteront pas plus de précisions en raison de la distance les séparant de la cible. À mesure que la cible se rapproche, celle-ci peut prendre de l'importance et devenir une réelle menace.

Des plans urgents pourront toujours préempter n'importe quelle ressource. Ils doivent cependant se faire rare, car l'annulation régulière de plans en cours d'exécution est à éviter en raison du coût de ces annulations. Les plans qui sont proches de la fin de leur exécution

ne devrait pas être arrêtés, leur score est également augmenté artificiellement pour éviter qu'un plan ayant un score un peu plus élevé ne l'arrête. L'augmentation de son score est proportionnelle à son pourcentage d'avancement.

3.5.3 RabbitMQ : un middleware pour la communication entre les agents

Les auteurs de [SSFS18] comparent plusieurs MOMs pour la communication dans des systèmes distribués : AMQP (RabbitMQ), Kafka, MQTT et ZeroMQ. Les investigateurs confrontent des caractéristiques comme la qualité de service, la sécurité, la standardisation ou les protocoles de transports utilisés. Ils considèrent que RabbitMQ est un middleware équilibré avec plus de flexibilité et de nombreuses fonctionnalités complémentaires. ZeroMQ surclasse les autres en raison de sa performance, critère très important pour des applications industrielles. Cette performance inclut la latence et le débit d'envoi des messages, qui sont optimisés par l'absence de *broker* et de fonctionnalités pouvant ralentir son utilisation.

De son côté, RabbitMQ présente des avantages supplémentaires en matière de tolérance aux fautes, de résilience et de facilité d'utilisation. L'intégration d'outils de monitoring tels que Prometheus et Grafana simplifie la surveillance du système. Dans l'ensemble, ces propriétés font de RabbitMQ un choix judicieux pour la communication de nos agents au sein de notre architecture distribuée.

RabbitMQ : définitions et utilisation

RabbitMQ est un middleware orienté messages [Dos14], aussi appelé MOM de l'anglais *Message Oriented Middleware*, utilisé dans les architectures distribuées pour la communication et la coopération entre les services. Le terme *broker* est couramment utilisé pour décrire RabbitMQ, un concept qui trouve également application dans l'architecture logicielle CORBA (*Common Object Request Broker Architecture*) ainsi que dans les bus logiciels. Le standard AMQP (*Advanced Message Queuing Protocol*) permet de délivrer des messages de manière asynchrone depuis un producteur [1] vers un consommateur [2] par le biais d'une file d'attente (*queue* en anglais).

1. Producteur : entité publiant les messages
2. Consommateur : entité recevant les messages

Dans le cadre de la thèse, les agents communiquent entre eux des informations permettant d'allouer des tâches aux ressources. RabbitMQ sert de passerelle pour le routage des messages et se configure en conséquence. Un même agent peut être à la fois producteur et consommateur de messages lors d'un échange bidirectionnel. L'agent définit une *connection* qui correspond à la couche transport du modèle OSI utilisant TCP entre les producteurs (ou consommateurs) et le broker. Pour chaque *connection*, plusieurs *channels* peuvent être établis, ce sont des connexions virtuelles entre un producteur (ou consommateur) et le broker. L'*exchange* est l'entité où les messages sont initialement envoyés avant d'être délivrés selon des règles de routage de différents types (*direct*, *fanout*, *topics*). Dans nos travaux, nous utilisons les règles de routage par *topics*, ce qui permet de mettre en œuvre le paradigme *publish/subscribe* et qui offre des fonctionnalités avancées permettant un routage plus précis avec le système de clés. Le message prêt à être délivré est mis en file d'attente et peut être dupliqué dans plusieurs files d'attente si les règles de routage le précisent. Enfin, le *binding* établit la connexion entre un *exchange* et une file d'attente. La clé de routage, quant à elle, est associée à la clé de *binding* pour diriger la transmission d'un message vers la file d'attente appropriée.

La figure 3.18 schématise des envois de messages depuis des producteurs avec des clés de routage « Test.critical » et « Test.common », ces messages sont envoyés à l'*exchange* qui est défini pour fonctionner sur du routage de topics. Le consommateur C_1 est abonné à la file d'attente nommée « Critical queue », il recevra uniquement les messages critiques de 16 producteurs grâce à l'abonnement à la *binding keys* « *.critical ».

Propriétés de RabbitMQ

Le caractère asynchrone est important. Dans notre architecture, les agents sont autonomes et ne peuvent pas se permettre d'attendre un message avant de continuer à travailler. L'asynchronisme du système rend l'atteinte d'un consensus difficile pour les agents sur l'utilisation concurrente des ressources.

RabbitMQ amène des propriétés et services importants pour l'architecture :

- La modularité : RabbitMQ offre une modularité permettant l'interconnexion de services ou d'autres entités dans le réseau à l'aide de protocoles de sérialisation tels que JSON et des mécanismes de routage, indépendamment des services déjà en place.
- La mise à l'échelle : RabbitMQ permet l'ajout et le retrait dynamique de nouveaux nœuds (producteurs et consommateurs) dans le réseau, le *broker* étant conçu pour

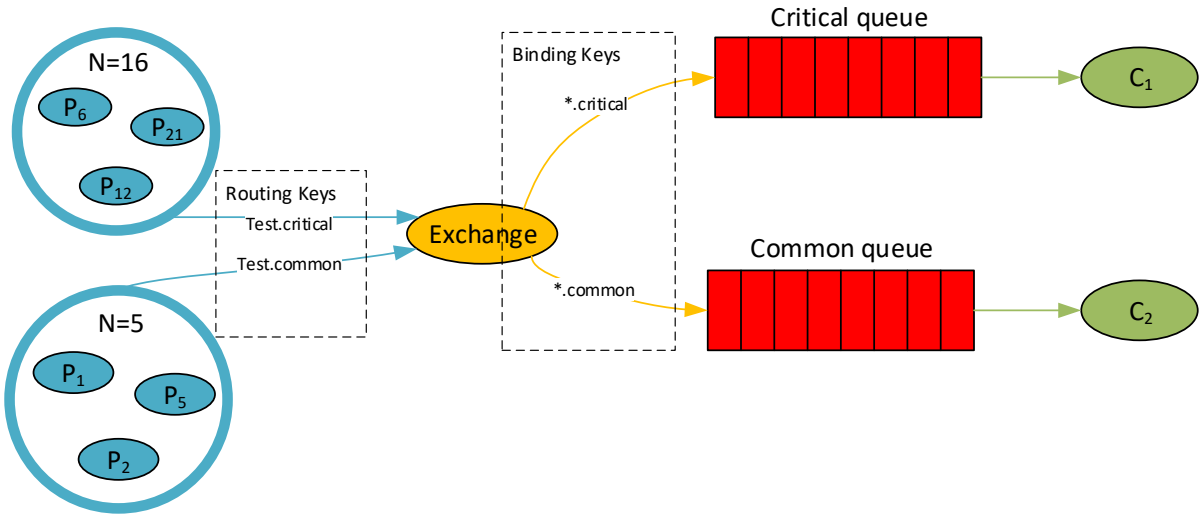

FIGURE 3.18 – Schéma de la structure de RabbitMQ pour l'envoi de message par Topics.

gérer des centaines de nœuds.

- La qualité de services (QoS) : RabbitMQ garantit la livraison des messages, permet de spécifier la durée de vie d'un message (TTL : *Time-To-Live*), et offre la possibilité d'appliquer des priorités sur les messages et les files d'attente. Par exemple, la figure 3.18 montre une file d'attente critique, le consommateur C_1 est un processeur capable de traiter plus rapidement l'information qu'il reçoit.
- Le monitoring du réseau : RabbitMQ offre une IHM pour la gestion du réseau en temps réel et exporte des métriques, telles que la quantité de messages entrant et sortant du *broker*, ou le nombre de producteurs et de consommateurs. D'autres outils de monitoring peuvent également être utilisés avec RabbitMQ [CK21].
- Fiabilité et performance : RabbitMQ garantit la tolérance aux pannes et aux échecs lorsque les nœuds ou les messages sont perdus, offre des options de configuration pour rendre les messages persistants sur le disque, permet l'acquittement des messages reçus et assure la prise en charge des messages par le serveur avec le *publisher confirm*.
- Interopérabilité : RabbitMQ supporte différents langages de programmation (Java, Python, C++, etc.) et prend en charge des protocoles de messageries (AMQP, MQTT, etc.). Il permet la distribution de messages de manière transparente en utilisant des modèles de communications variés (*fanout*, *direct*, *topic*, *header*).

Definition 10 (Métriques) Les métriques d'un système désignent les données perti-

nentes que nous surveillons pour garantir son bon fonctionnement. Elles fournissent des indicateurs quantitatifs permettant d'évaluer les performances, la disponibilité, la fiabilité et d'autres aspects du système. Dans le cadre de RabbitMQ, ces métriques peuvent inclure des indicateurs spécifiques tels que le nombre de *sockets* ouverts, le nombre de messages présents dans une file d'attente, ainsi que les taux d'utilisation des ressources système tels que le CPU, la mémoire et le stockage. La latence et le débit moyen permettent d'évaluer la performance globale d'un système d'un point de vue du réseau.

RabbitMQ dans d'autres projets

D'autres travaux utilisent également RabbitMQ. C'est le cas du projet européen CAMELOT (C2 Advanced Multi-domain Environment and Live Observation Technologies) [PIA20], qui vise à améliorer l'interopérabilité des systèmes en permettant aux usagers de contrôler des drones et des senseurs pour fournir des services complexes. C'est dans ce cadre que RabbitMQ est utilisé, afin de gérer la transmission de messages entre des modules hétérogènes distribués, et d'abstraire les couches basses du modèle OSI (1-4) [ASD18]. Les critères qui ont poussé ce choix de middleware sont les suivants : la modularité, la mise à l'échelle, l'orienté services, les capacités *publish/subscribe*, la standardisation (AMQP), le registre des services, la mise en réseau (via ethernet), etc. Ce projet utilise JSON pour structurer et sérialiser les données entre les composants du système. Le middleware a été sélectionné selon les spécifications du projet, en concurrence avec ZeroMQ, DDS, ECOA, JAUS et Kafka.

OpenStack est une plateforme open source qui offre une infrastructure en tant que service (IaaS : *cloud computing*) [RB14]. La plateforme est fondée sur une combinaison modulaire de logiciels, tels que Cinder, Glance ou encore Nova. Ces différents composants permettent aux entreprises qui utilisent OpenStack de choisir les services qui leurs seront utiles, que ce soit des services de calcul, de stockages, d'image ou de réseau. RabbitMQ est utilisé pour la diffusion des messages entre les services d'OpenStack, ce qui permet une communication fiable et asynchrone entre les composants.

RabbitMQ a également été utilisé dans la communication entre des agents d'un SMA [AYBA20]. L'interopérabilité entre des SMA hétérogènes nécessite des protocoles de communication fiables pour l'échange asynchrone de messages. Ensuite, des recherches [MIAP19] présentent un outil pour faire du monitoring de message entre des éléments d'un réseau intelligent appelé « Smart Home System ». L'outil permet également de faire de la mise à l'échelle automatique lorsque des problèmes sont détectés. Ils ont choisi d'utiliser

RabbitMQ pour les raisons suivantes : la multiplicité des protocoles de messagerie, la mise en file d'attente des messages et leur acquittement. Dans l'architecture proposée, les microservices envoient des données du côté du producteur, les nœuds consommateurs consomment les messages stockés dans une file d'attente. En outre, le document décrit un environnement de test utilisant Zabbix comme outil de surveillance. Les auteurs ont introduit différentes valeurs pour des paramètres tels que le *prefetch* (la quantité de messages que le consommateur récupérera de la file d'attente afin de les traiter effectivement) ou le nombre de consommateurs. Lorsque la charge sur un consommateur est trop importante (sa file d'attente est proche de la saturation), l'outil permet de créer de nouveaux consommateurs pour subvenir à une charge de messages importante, avant de supprimer des consommateurs si nécessaire quand le système est stabilisé.

3.5.4 Monitoring des communications

La documentation de RabbitMQ sur le *monitoring* justifie son importance dans les systèmes distribués par l'analogie suivante :

> « Operating a distributed system without monitoring data is a bit like trying to get out of a forest without a GPS navigator device or compass. It doesn't matter how brilliant or experienced the person is, having relevant information is very important for a good outcome. » [Rab]

Le monitoring du système permet d'évaluer les performances de celui-ci dans des conditions de forte charge, afin d'identifier les goulots d'étranglement et les limites du système. L'identification des problèmes permet ensuite d'adapter le système, par exemple, en réduisant la fréquence d'envoi des messages si possible, en dimensionnant les ressources du systèmes pour convenir à la charge nécessaire, ou encore en révisant les politiques de routage pour les adapter aux besoins. La variation du nombre d'agents plateformes et tactiques impacteront différentes métriques à analyser : la latence, le nombre de canaux de communication, le nombre de messages échangés, etc.

Nous avons fait le choix de Prometheus et de Grafana pour monitorer le système, ce sont des outils puissants pour la capture de métriques et qui sont par ailleurs hautement recommandés avec RabbitMQ.

Prometheus

Prometheus [Tur18] est outil open-source de surveillance et d'alerte qui enregistre des métriques dans une base de données temps-réel (voir figure 3.19) et fournit un langage de

requête appelé *PromQL*. Ce langage permet de faire des requêtes sophistiquées dans la base de données. Prometheus offre quatre types de métriques :

- *Counter* : Il s'agit d'une métrique cumulative qui ne peut qu'augmenter ou être remise à zéro, ce qui permet de suivre des compteurs d'événements ou de nombre de messages échangés.
- *Gauge* : Cette métrique représente une valeur numérique pouvant augmenter ou diminuer, ce qui permet de suivre des mesures variables dans le temps comme la latence ou le débit.
- *Histogram* et *Summary* : Ces métriques permettent de calculer des quantiles, de découper les données en intervalles, et de fournir des informations statistiques sur la distribution des valeurs observées.

De plus, Prometheus offre des fonctionnalités avancées telles que l'utilisation d'expressions régulières *regex* pour modifier le nom des métriques sortantes, la collecte de valeurs sur des intervalles de temps définis et leur stockage dans des vecteurs, ainsi que l'utilisation d'opérations d'agrégation (sum, min, max) et d'autres fonctions pour manipuler les données.

```
# HELP TestRabbitMQ_debit_Producteur_4
# TYPE TestRabbitMQ_debit_Producteur_4 gauge
TestRabbitMQ_debit_Producteur_4 694.0
```

FIGURE 3.19 – Exemple de métrique *Gauge* au format d'exportation de Prometheus.

Grafana

Grafana est un logiciel de management et d'analyse de traces qui offre la possibilité de visualiser des données sous forme graphique dans des panneaux appelés *dashboards*. Ce logiciel tire parti de bases de données temporelles telles que celles stockées sur le serveur Prometheus dans notre cas. Ces données sont régulièrement mises à jour à des intervalles de quelques secondes et présentées en temps réel sur le *dashboard*. L'un des points forts de Grafana réside dans sa capacité à personnaliser les *dashboards* pour répondre aux besoins spécifiques des utilisateurs. Ils peuvent choisir parmi une variété de types de graphes, y compris des graphes temporels, des jauges, et des histogrammes, pour mettre en évidence les tendances et les données pertinentes, afin de comprendre les limites du système et de

le modifier si besoin. De plus, l'utilisation de variables permet de maintenir la cohérence d'un *dashboard* pour différentes configurations, offrant ainsi une flexibilité d'utilisation et une adaptabilité à divers scénarios. En outre, Grafana offre des fonctionnalités avancées telles que l'utilisation d'expressions régulières, ce qui facilite la mise à l'échelle et les changements de configurations du système sans nécessiter de modifications majeures sur l'ensemble du *dashboard*.

L'intégration de Grafana, Prometheus et RabbitMQ dans le système offre plusieurs avantages significatifs. En surveillant en temps réel l'état des échanges entre les agents, ces outils permettent de détecter rapidement les points bloquants, les goulets d'étranglement et les limites du système. Cette visibilité facilite l'identification des problèmes potentiels. De plus, en collectant et en analysant des données détaillées sur les performances du système, Grafana fournit des informations sur les tendances et le comportement du système. Ces éléments permettent de prendre des décisions dans le but d'améliorer les performances du système en modifiant les interfaces de communication, et en dimensionnant les ressources, les services et les agents nécessaires pour répondre aux besoins opérationnels.

3.6 Complexité de la solution

3.6.1 Complexité des communications

Le complexité des communications vise à étudier la quantité d'informations échangée entre les participants d'un système afin d'accomplir certaines tâches [Kus97]. Les mesures de complexité incluent le nombre de messages envoyés, le nombre total de bits et le nombre total de phases de calcul [Bro14]. Dans notre système multi-agent, nous considérons uniquement les communications entre les plateformes. Lors d'un échange, p producteurs envoient k messages de n bits à c consommateurs. La mesure de complexité d'une interface de communication s'intéresse au pire cas de bits n échangés pour k messages.

Le débit n'est pas considéré dans l'étude des complexités pour plusieurs raisons. Tout d'abord, le débit est indépendant de la complexité en nombre de messages. Le nombre de messages, la taille des messages et la fréquence d'envoi des messages supposent un débit théorique qui ne tient pas compte de la latence, du traitement des informations ou de la congestion du réseau, éléments qui peuvent avoir un impact sur le débit réellement observé dans un système. Cependant, d'autres paramètres font que le débit varie, tels que la bande passante du réseau utilisé, le volume des messages échangés et la capacité de traitement des processus avant l'émission et lors de la réception des messages. De plus, le débit peut être modifié et optimisé en utilisant des files d'attente ou du parallélisme.

Échange de données entre plateformes

Prenons le cas simple d'un système avec trois plateformes qui se transmettent des données. Chaque plateforme enverra un message par plateformes alliées, soit un total de 2 messages. Ces messages comprennent des données de position pour la fonction *sendMessagePosition*, des informations de ressources avec la fonction *sendMessageResource* et les requêtes opérationnelles avec la fonction *sendMessageRequest*. Au total, cela fera $2 * 3 * 3 = 18$ messages à transmettre, car les trois plateformes enverront chacune aux deux autres plateformes trois messages : de position, de ressource et de requêtes. Ces messages ne sont pas envoyés continuellement, le partage des requêtes opérationnelles se fait en général une fois à l'arrivée de la plateforme dans le réseau et ensuite uniquement en cas de modification de celles-ci. Le partage des ressources se fait également au niveau de chaque plateforme quand des caractéristiques ou des paramètres de la ressource sont modifiés. Enfin, l'échange de position se fait par période fixe qui dépend de la vitesse de

déplacement de la plateforme. La position d'une plateforme aérienne devra être rafraichis plus fréquemment que la position d'un navire.

Dans un système avec M plateformes, chaque agent plateforme envoie des messages de n bits à $M-1$ agents plateformes. Par exemple, la taille d'un paquet contenant une position fera une taille de 24 octets, avec trois *double* qui font chacun 8 octets. La taille d'un paquet transitant sur un réseau ne peut être qu'estimée sans connaissance des méthodes d'encodage et de compression. La complexité en nombre de messages de ces 3 fonctions correspond à l'envoi de $M-1$ messages par M agents, ce qui fait donc $M*(M-1)$ messages, soit une complexité en nombre de messages de l'ordre de $O(M^2)$. La complexité binaire pour des messages de n bits peut se rapprocher de $O(n*M^2)$.

Communication de pré-allocation

Pour le même cas avec trois plateformes, les seules communications possibles seront entre les trois agents services et les trois agents plateformes. Ainsi, la première étape de pré-allocation nécessite l'envoi de 3 messages comprenant une liste de plans, dont ces derniers ont une taille de n octets. L'envoi est effectué par la fonction *sendMessagePre-Reservation*. Dans le pire des cas, la liste aura autant d'éléments que le nombre d'agents tactiques actifs sur la plateforme, car l'agent service accepte au maximum un plan par agent tactique non satisfait. Cette action est également effectuée par les deux autres agents services, ce qui fait un total de 9 messages à envoyer dans le pire des cas (certaines plateformes peuvent ne recevoir aucune demande).

En reprenant la logique sur la taille d'un paquet vue précédemment, nous pouvons approximer la taille en octets d'un plan et donc celle d'une liste. Un plan se compose de différents champs : un ID unique, un statut de plan, la date de création du plan, la date d'exécution du plan, le nom de l'agent tactique auquel il est rattaché (un *String* d'environ 8 octets dans notre cas). Ainsi, un plan aura une taille minimale de 36 octets environ sans compression ni encodage. Dans le cas d'une plateforme possédant 10 agents tactiques, la liste de plan envoyés pourra faire 360 octets maximum.

Pour chaque envoi de message, un retour sera fait sur chaque plan avec la fonction *replyToPreReservation* de l'agent plateforme. Les plans proposés pourront être acceptés ou refusés, mais dans tous les l'agent service en sera informé.

Dans un système avec M plateformes, chaque agent service envoie des messages de tailles variables (en fonction du nombre de plans envoyés) aux agents plateformes concernés, soit $M-1$ messages dans le pire des cas. L'agent plateforme répond à l'agent service

correspondant pour chaque proposition qu'il reçoit. Les fonctions de communication *replyToPreReservation* et *sendMessagePreReservation* ont une complexité en nombre de messages de l'ordre de $O(M^2)$, mais $O(t * M^2)$ si les plans ne sont pas regroupés dans un même message selon leur destinataire, où t correspond au nombre d'agents tactiques.

La complexité binaire dépend du nombre d'agents tactiques qui ont proposé des plans et du nombre de ressources concernées par le service. Un service collaboratif dupliquera son message pour l'envoyer à deux plateformes, ce qui augmente la quantité de bits nécessaires. En considérant ces paramètres, la complexité binaire des messages est de l'ordre de $O(2 * n * t)$ où $n * t$ correspond à la taille du message qui comprend les propositions de plans d'un nombre total de t agents tactiques. Ce message est envoyé deux fois lorsqu'il s'agit d'un service utilisant deux ressources.

Communication d'allocation réelle et mise à jours des services en cours

La demande d'allocation réelle des ressources par la fonction *sendMessageRealReservation* envoie le même nombre de messages dans le pire des cas. Le nombre de plans acceptés globalement pourra être moins important que le nombre de plans acceptés localement, sauf dans le pire des cas où il pourra être égal (lorsque tous les plans proposés sont finalement acceptés).

Une fois les services acceptés par les agents plateformes, les ressources sont en cours d'exécution lorsque de nouveaux messages de réservation se présentent. Chaque demande d'utilisation de la ressource pourra être à l'origine d'un message de l'agent plateforme via la fonction *sendMessageServiceUpdate* :

- La réservation de la ressource n'est pas possible, l'agent plateforme répond par un message de refus du plan.
- La réservation de la ressource est possible, si cette réservation annule un plan en cours alors un message sera envoyé uniquement pour l'annulation.
- Le plan est arrivé à son terme sans aucune annulation, l'agent plateforme informe par message l'agent service que son plan est terminé.

Comme dans les cas précédents, la communication effectuée par les fonctions *sendMessageRealReservation* et *sendMessageServiceUpdate* a une complexité en nombre de messages de l'ordre de $O(M^2)$ et une complexité binaire de l'ordre de $O(2 * n * t)$.

Exemple d'un échange complet

Prenons le cas où nous avons exactement cinq plateformes avec dix agents tactiques sur chacune d'entre elles. Chacune pourra proposer au maximum dix plans, nous admettons que cela correspondra à des messages de 360 octets dans le pire des cas. Cependant, ces plans ne seront envoyés qu'aux plateformes concernées. En considérant des services collaboratifs utilisant deux ressources, nous pouvons estimer qu'il y aura 720 octets de plans répartis sur quatre messages différents. Ce nombre est multiplié par cinq, car chaque plateforme pourra envoyer le même nombre de messages, soit un total de 3600 octets. Avec les quatre interfaces, cela représente 14,4 ko.

Au niveau du nombre de messages échangés, chacune des cinq plateformes enverra au maximum quatre messages par interface, soit un total de 64 messages. Si nous considérons que l'agent service attend 100 millisecondes entre chaque enchère, nous obtenons une bande passante théorique de 720 ko/s pour l'ensemble des plateformes (144 ko de données à envoyer toutes les secondes pour chaque plateforme). A titre de comparaison, si nous portons notre exemple à dix plateformes, nous passons à un total de 360 messages envoyés et une bande passante de l'ordre de 1,44 mo/s.

La taille des messages dépend principalement du nombre de plans et donc du nombre d'agents tactiques. Nous noterons par ailleurs que, dans l'exemple, le nombre d'agents plateformes a doublé, tout comme le nombre d'agents tactiques, car chaque agent plateforme en possède toujours dix. Si nous considérions toujours le même nombre total d'agents tactiques, il y aurait donc cinq agents tactiques par plateforme, ce qui allège le contenu des messages en le divisant par deux. Nous pouvons donc conclure que le nombre de messages dépend du nombre de plateformes, mais que la bande passante dépend uniquement du nombre d'agents tactiques. En effet, si une plateforme ne possède aucun agent tactique, aucun message ne sera transmis. En pratique, la complexité en bits des interfaces de communication de l'enchère est donc bien linéaire comme les plans sont envoyés aux plateformes concernées uniquement. Ce n'est pas le cas du partage des informations de positions ou de ressources qui lui dépend du nombre de plateformes.

3.6.2 Complexité algorithmique

La complexité computationnelle analyse le temps de calcul d'un algorithme avant de trouver une solution dans le pire des cas ou en moyenne [Pap03]. Il est important de prendre en compte cette complexité qui s'ajoute à la latence globale du système. En effet,

les agents calculent des services ou proposent des plans, ce qui ajoute du délai en plus de la latence des communications.

Algorithme 1 : *checkIfResourceIsAvailable*

L'algorithme *checkIfResourceIsAvailable* vérifie la disponibilité des ressources en fonction des plans $\mathcal{P}$ proposés par l'agent service. Une itération sur la liste de plans est effectuée à la ligne 2, l'agent plateforme regarde chaque plan qu'il a reçu et compare l'énergie requise pour le service avec l'énergie disponible. La ligne 6 propose de récupérer les ressources en charge du plan. Sa complexité est négligeable, car il n'y a qu'une seule ressource par plan pour une plateforme dans notre cas. La ligne 13 vérifie chaque plan actif sur la ressource considérée. Cette boucle dépend donc principalement de l'énergie maximale initiale E_m de la ressource, plus celle-ci est élevée et plus la complexité augmente. Les fonctions *setPlanStatus* et *Add*, en ligne 17 et 18, sont en $O(1)$. L'algorithme *checkIfResourceIsAvailable* a donc une complexité en $O(n * m)$. La variable n correspond au nombre de plans proposés par l'agent service, qui sera dans le pire des cas égale au nombre d'agents tactiques de toutes les plateformes. La variable m correspond au nombre de plans actifs sur la ressource et dépend de la capacité de celle-ci à gérer plusieurs tâches en parallèle.

Algorithme 2 : *updateServices*

L'algorithme *updateServices* met à jour la liste des services. À la ligne 4, le nombre de ressources n_r pour réaliser un type de service peut être obtenu grâce à la fonction $getListOfResourceForEachServiceType$ qui n'est pas détaillée ici. Cette fonction prend en entrées chaque type de services et l'ensemble des ressources. Sa complexité est en $O(m * n)$, avec m le nombre de types de services (S1, S2, etc.) et n le nombre total de ressources, qui dépend donc du nombre de plateforme et du nombre de ressources sur chacune d'entre elles. En général, le nombre de types de services sera toujours négligeable par rapport au nombre de ressources dans le système, donc nous pouvons approximer la complexité à $O(n)$.

Ensuite, nous itérons à la ligne 3 sur chaque type de service t avec une première boucle *for*. Selon le nombre de ressources nécessaires à l'accomplissement du type de service, nous entrons dans des cas différents. Nous considérons le pire cas, celui où un type de service utilise deux ressources pour sa collaboration. Une double itération s'effectue sur la liste des ressources pour obtenir l'ensemble de combinaisons possibles afin de créer

la liste des services, ce qui fait une complexité en $O(n_r^2)$. La désignation des ressources en ligne 16 est en $O(1)$ en accédant directement à l'indice dans la liste des ressources. La performance en ligne 17 a une complexité qui dépend du nombre de ressources n_s pour réaliser le service qui se résume à un maximum de deux opérations dans notre cas. La fonction $getPerformance$ possède donc une complexité $O(1)$. L'opération *Add* consiste simplement à ajouter le service à la fin de la liste des services. La complexité de l'algorithme 2 peut donc être approximée à $O(t * n_r^2) + O(n)$ avec t qui dépend du nombre de types de services, n_r qui correspond au nombre de ressources pouvant réaliser chaque type de service et n le nombre total de ressources.

Algorithme 3 : *resourceAllocation*

L'algorithme *resourceAllocation* alloue les ressources localement au niveau d'un agent service. La fonction *Sort* de la ligne 2 sert à trier la liste selon le score des plans proposés par les agents tactiques et possède une complexité temporelle en $O(n * logn)$. Cela permet ensuite, en ligne 3, de prendre directement les meilleurs plans de la liste dans l'ordre. La boucle s'arrête quand l'ensemble des plans proposés a été parcouru ou lorsque tous les agents tactiques sont satisfaits, c'est-à-dire qu'ils ont un plan d'accepté. Les opérations au sein de la boucle ont une complexité en $O(1)$ et la boucle de la ligne 7 de l'algorithme se limite à deux itérations au maximum, car il n'y a besoin que de deux ressources dans notre cas pour effectuer un service. La boucle en ligne 3 a donc une complexité en $O(n)$ et l'algorithme *resourceAllocation* a une complexité en $O(n * logn)$ dans son ensemble, car la complexité du tri domine la complexité des autres opérations.

L'entrée de l'algorithme étant le nombre de plans proposés par les agents tactiques, chacun d'entre eux doit limiter et trier ses propositions de plans. Le fait de proposer tous les plans possibles permet de s'assurer que toutes les ressources disponibles seront utilisées au maximum, même pour des plans avec un score plus faible. Cependant, si l'agent service propose de nombreux services et que chaque agent tactique propose un score pour chaque plan calculé, alors l'algorithme *resourceAllocation* devra trier un nombre de plans total proportionnel au nombre d'agents tactiques.

Algorithme 4 : *calculatePlan*

L'algorithme *calculatePlan* permet à l'agent tactique de calculer le score des plans en fonction de la liste de services $\mathcal{S}$ qu'il a à disposition. La première ligne permet de sélectionner les services selon leurs performances et s'ils répondent aux objectifs de l'agent

tactique. Cette fonction a une complexité en $O(n)$, où n correspond au nombre de services dans la liste en entrée. Suite à la sélection des services, la liste $\mathcal{S}$ voit sa taille diminuer, l'itération de la ligne 3 s'effectue sur cette nouvelle liste. Les fonctions *getScore* et *Add* ont des complexités constantes. Ce qui fait que la complexité au pire cas de l'algorithme *calculatePlan* est en $O(n)$ et dépend de la taille de la liste de services.

Algorithme 5 : *checkPlanStatus*

L'algorithme *checkPlanStatus* permet de vérifier le statut des plans envoyés par l'agent tactique à l'agent service. Cette vérification détermine si l'agent tactique est satisfait et s'il doit de nouveau planifier dans le cas contraire. L'algorithme a une complexité en $O(n)$, où n correspond au nombre de plans que l'agent a envoyé, ce nombre dépend également du nombre de ressources et de types de services.

Interfaces	**Entrées**	**Expéditeurs/Destinataires**	**Complexité en bits**	**Complexité en message**
sendMessagePosition	Position *pos*	agent plateforme→M agents plateformes	$O(n * M^2)$	$O(M^2)$
sendMessageResource	Resources $\mathcal{R}$	agent plateforme→M agents plateformes	$O(n * M^2)$	$O(M^2)$
sendMessageRequest	OperationalRequest *or*	agent plateforme→M agents plateformes	$O(n * M^2)$	$O(M^2)$
sendMessagePreReservation	Plans $\mathcal{P}_{local}$	agent service→M agents plateformes	$O(2 * n * t)$	$O(M^2)$
replyToPreReservation	Plans $\mathcal{P}_{return}$	agent plateforme→M agents services	$O(2 * n * t)$	$O(M^2)$
sendMessageRealReservation	Plans $\mathcal{P}_{global}$	agent service→M agents plateformes	$O(2 * n * t)$	$O(M^2)$
sendMessageServiceUpdate	Plans $\mathcal{P}_{return}$	agent plateforme→M agents services	$O(2 * n * t)$	$O(M^2)$

TABLE 3.6 – Liste des interfaces de communication.

Algorithme	**Entrées**	**Sorties**	**Fonctions**	**Complexité**
checkIfResource IsAvailable	Plans $\mathcal{P}$	Plans $\mathcal{P}_{return}$	Vérification de la disponibilité des ressources	$O(n * m)$
updateServices	Ressources $\mathcal{R}$	Services $\mathcal{S}$	Mise à jour de la liste des services de l'agent	$O(t{*}n_r^2){+}O(n)$
resourceAllocation	Plans $\mathcal{P}$	Plans $\mathcal{P}_{accept}$	Allocation locale des ressources	$O(n * logn)$
calculatePlan	Services $\mathcal{S}$	Plans $\mathcal{P}$	Calcul de plans par l'agent tactique	$O(n)$
checkPlanStatus	Plans $\mathcal{P}$		Vérification du statut des plans en cours	$O(n)$

TABLE 3.7 – Liste des algorithmes.

3.6.3 Conclusion et synthèse

Dans cette section, nous avons présenté les complexités des différents algorithmes et interfaces. D'une part, nous avons analysé les interfaces de communication et leur complexité en terme de nombre de messages et de bits envoyés. D'autre part, nous avons détaillé les algorithmes intégrés dans les agents et leurs complexités temporelles.

Dans le contexte d'un système où des agents communiquent et calculent à l'aide de fonctions internes, les algorithmes, les communications et leurs complexités impactent directement les performances du système. Des algorithmes efficaces et des interfaces de communication optimisées peuvent améliorer les performances globales du système en réduisant les temps d'exécution et les délais de transmission.

Nous avons vu que les complexités dépendaient fortement de plusieurs paramètres : le nombre de services dans le système, le nombre de plateformes et le nombre de cibles. Ces éléments sont par ailleurs impactés par le nombre total de ressources, le nombre de types de services différents et le nombre de ressources locales à chaque plateforme. Ces paramètres d'entrée affectent les différentes complexités qui pourront être observées en les faisant varier lors de simulations.

En termes d'optimisation, comprendre comment ces paramètres influencent les complexités nous permet de concevoir des stratégies pour optimiser le système. Par exemple, en optimisant les algorithmes pour une utilisation plus efficace des ressources disponibles et en choisissant des interfaces de communication adaptées aux besoins spécifiques du système, nous pouvons minimiser les temps d'exécution et maximiser les débits de transmission. De plus, en ajustant ces paramètres en fonction des besoins réels du système, nous pouvons garantir des performances optimales dans les scénarios proposés.

3.7 Conclusion du chapitre

Dans ce troisième chapitre, nous nous sommes focalisé sur la conception de notre système, en charge de répondre à la problématique de l'allocation de tâches à des senseurs dans un environnement distribué. Des scénarios opérationnels ont été décrits, permettant de soulever les contraintes et les besoins du système dans des contextes définis. Notre proposition d'architecture, centrée sur l'utilisation d'agents, a été présentée et détaillée. Une attention particulière a été portée sur les algorithmes et les interfaces de communication qui régissent le fonctionnement de ces agents. Nous avons analysé leurs complexité, mettant en évidence l'impact de la latence sur les échanges entre les agents et l'allocation efficace des tâches à des senseurs. Ces différentes contributions constituent les fondements de notre approche pour résoudre la problématique d'allocation de tâches dans un contexte distribué.

Dans une première section, nous avons exposé les approches possibles pour le développement de concepts d'architecture pour le combat collaboratif naval. Nous avons présenté des scénarios dans une deuxième section, ceux-ci ont permis d'appréhender la problématique et de comprendre les exigences du système en termes de fonctionnalités. Ces scénarios ont permis de donner des cadres et contextes d'emploi de l'architecture proposée. Ensuite, nous avons détaillé les étapes de conception de l'architecture multi-agent dans une troisième section. Nous avons justifié l'intérêt des SMA pour la résolution de notre problématique avant d'analyser le système et sa structure pour identifier des rôles à faire correspondre à des types d'agents. Dans une quatrième section, nous avons expliqué le fonctionnement de trois agents en présentant les principaux algorithmes et en détaillant les points critiques comme l'interblocage, les famines ou le partage des ressources. Nous avons proposé et précisé l'utilisation d'une méthode d'allocation des tâches par enchère.

Dans une cinquième section, nous avons développé l'aspect distribué du système en nous intéressant aux échanges nécessaires entre les agents pour atteindre une solution lors du mécanisme d'enchère. Pour cela, nous avons schématisé les échanges afin de démontrer que des problématiques de cohérence entre les agents existent en raison de la latence. En pratique, nous avons proposé l'utilisation du middleware RabbitMQ pour simuler les échanges entre les agents et observer ces phénomènes. Enfin, nous avons détaillé les complexités des interfaces de communications et des algorithmes de traitement utilisés par les agents dans une sixième et dernière section. Nous avons vu que ces complexités dépendaient étroitement de différents paramètres du système et qu'elles pouvaient avoir un

impact significatif sur le passage à l'échelle de la solution. Ce dernier point nous contraint à expérimenter le système et à observer ses limites réelles tout en proposant des alternatives pour optimiser l'utilisation des algorithmes et des interfaces de communication dans certaines situations.

Chapitre 4

Experimentation et Evaluation

4.1 Simulation

Le département de la défense américain (*US DoD*) définit la simulation comme : « Une méthode d'implémentation d'un modèle au cours du temps. Également, une technique pour tester, analyser, ou entrainer dans laquelle le monde réel et les concepts de systèmes sont reproduits par des modèles. »[HM17].

Dans le cadre de nos travaux, la simulation joue un rôle essentiel en nous permettant d'explorer différents scénarios opérationnels et d'évaluer l'impact d'événements spécifiques sur le comportement des plateformes militaires et de leurs architectures. En effet, la mise en œuvre de nouveaux concepts dans des conditions réelles, sans avoir évalué préalablement le modèle, peut s'avérer excessivement coûteuse et risquée. La simulation offre ainsi la possibilité de comprendre le fonctionnement du système à travers des modèles qui peuvent être plus ou moins proches de la réalité, avant même leur déploiement effectif.

Une simulation repose généralement sur un ensemble de modèles qui représentent de manière simplifiée les entités simulées. Par exemple, dans le cas de la trajectoire de missiles dans un théâtre d'opérations, un modèle de trajectoire est influencé par divers facteurs environnementaux tels que les conditions de propagation dans l'air, le vent, la pluie, et d'autres paramètres. Ces éléments doivent également être pris en compte et reproduits fidèlement dans la simulation pour assurer la validité des résultats obtenus.

Ainsi, la simulation nous offre un outil puissant pour tester, analyser et valider de nouveaux concepts et stratégies militaires, tout en minimisant les risques et les coûts associés à des essais en conditions réelles. Elle nous permet d'anticiper les performances et les comportements des systèmes, et de prendre des décisions en amont de leur déploiement effectif sur le terrain.

La simulation militaire peut revêtir des formes très abstraites, utilisant des modèles analytiques qui sont plus facilement accessibles que la mise en place d'exercices militaires en conditions réelles (voir figure 4.1). Cependant, il convient de noter que plus la simu-

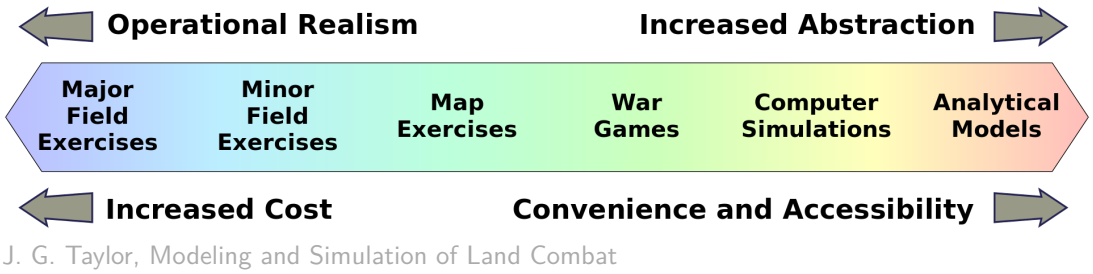

FIGURE 4.1 – Diagramme du spectre de la simulation militaire démontrant la corrélation entre le type de simulation, le niveau d'abstraction et la faisabilité.

lation est réaliste et plus son coût sera important. Idéalement, une simulation militaire devrait être la plus fidèle possible à la réalité, bien que cette aspiration soit souvent difficile à atteindre en pratique. Ces simulations doivent s'appuyer sur des critères préétablis, définissant notamment les performances attendues pour un nouveau système par rapport aux systèmes existants. La preuve de concept s'effectue lors d'une première phase de simulation abstraite, et les concepts prometteurs seront affinés en simulation pour se rapprocher d'un modèle réaliste.

Dans le cadre de nos travaux, la simulation est employée comme un outil de validation indispensable en l'absence d'autres moyens pour évaluer les travaux d'architecture. Un simulateur opérationnel a été développé par Thales pour exécuter et enregistrer des scénarios en temps réel. Le comportement des plateformes alliées et ennemies y est simulé en intégrant des informations sur les plateformes et les modèles de senseurs provenant de la base de données « cmano » [cem].

Nos premières expérimentations consistaient à développer une interface entre ce simulateur et un logiciel à base d'agents servant à ordonnancer des services senseurs. L'objectif était de pouvoir observer le comportement des plateformes dans le simulateur en leur injectant une intelligence artificielle qui décide des actions senseurs à effectuer dans une boucle, où les actions apportent de nouvelles connaissances pour choisir les actions suivantes à accomplir. D'un côté, les senseurs effectuent des tâches demandées par des agents. D'un autre côté, les agents reçoivent les données des senseurs et les états des plateformes pour prendre des décisions. En raison des contraintes de temps et des défis inhérents à la modification du logiciel à base d'agents existant, nous avons pris la décision de développer un nouveau *framework* pour intégrer et tester efficacement nos nouveaux concepts d'agents pour l'allocation de tâches à des senseurs, dans l'objectif de pouvoir l'intégrer

dans un simulateur. Les expérimentations présentées dans ce chapitre permettront de démontrer la pertinence des concepts d'architecture proposés au regard des besoins spécifiés en amont. Nous pourrons montrer l'impact des différents paramètres sur les performances du système et discuter des résultats au regard des attentes.

4.2 Démonstrateurs de concepts d'agents : Ramses II

Dans sa thèse [Gri18], L. Grivault a développé deux démonstrateurs pour présenter ses nouveaux concepts d'agents tactiques : Ramses I et Ramses II. La première simulation utilisait le framework multi-agent JACK [Win05] qui permet d'implémenter des agents BDI, ce framework est comparable à JADE dans ce contexte d'utilisation. Nous présentons un exemple de développement multi-agent avec JADE en annexe. La première version de Ramses II utilisait la plateforme multi-agent JIAC-V [LKK+13], mais la création d'une plateforme multi-agent spécifique a été décidée en raison de besoins imposés par le contexte : la gestion du temps, la mise en pause des agents, la simplification de la gestion du nœud d'agents, un contrôle bas-niveau des agents, une simplification des agents et une mise à l'échelle plus performante. Le logiciel propriétaire que nous utilisons pour l'interfaçage dans la partie suivante récupère l'implémentation multi-agent de Ramses II. Nos problématiques étant proches de ces travaux connexes, nous développerons notre propre système multi-agent en faisant évoluer les modèles d'agents fournis par le framework existant. Plus particulièrement, nous développerons de nouvelles interfaces de communication pour les agents.

L'agent tactique proposé par L. Grivault se comporte selon les événements d'une machine à états (voir figure 4.2). Les agents exécutent une succession d'actions qui amènent à proposer continuellement de nouveaux plans senseurs à un algorithme d'ordonnancement qui les acceptera ou les refusera en fonction des besoins des différents agents tactiques.

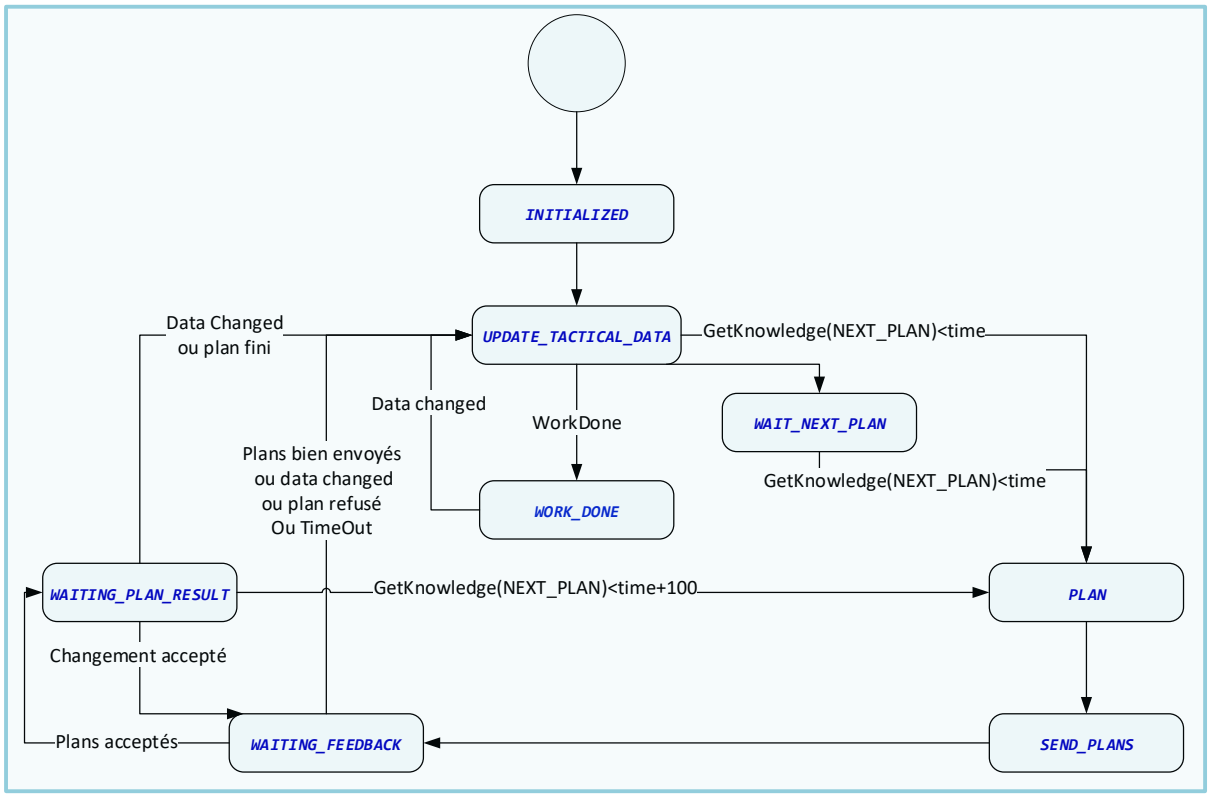

FIGURE 4.2 – Machine à état de l'agent tactique de Ludovic Grivault.

4.3 Interfaçage d'un simulateur avec un logiciel propriétaire

Le simulateur opérationnel permet l'intégration de nouveaux concepts appliqués au domaine d'utilisation, tout en offrant un moyen de valider l'architecture mise en place. Pour pouvoir interfacer le simulateur avec le logiciel à base d'agents, nous avons opté pour le format de sérialisation *Protocol Buffers* (Protobuf), un langage de description d'interface développé par Google. Ce dernier permet de générer des interfaces dans divers langages. Parallèlement, nous avons utilisé le middleware ZeroMQ pour faciliter l'échange de messages entre nos deux applications. Ces deux outils sont décrits en annexe.

4.3.1 Contexte

Dans le cadre de nos recherches, nous avons utilisé des logiciels Thales afin d'expérimenter et de proposer une première approche d'architecture. D'une part, un programme

Java utilise des agents logiciels pour ordonnancer des tâches tactiques à des plateformes sur la base d'un annuaire statique de services et de modèles de performances à gros grain (moins détaillés). D'autre part, un simulateur opérationnel écrit en C++ intègre différents acteurs au sein d'un environnement opérationnel et peut exécuter des scénarios en temps réel. Dans le cadre de nos travaux, nous voulons simuler les scénarios présentés en section 3.2, selon différents points de vue de l'architecture, puis vérifier que cette dernière répond aux besoins du système (voir 1.6.1).

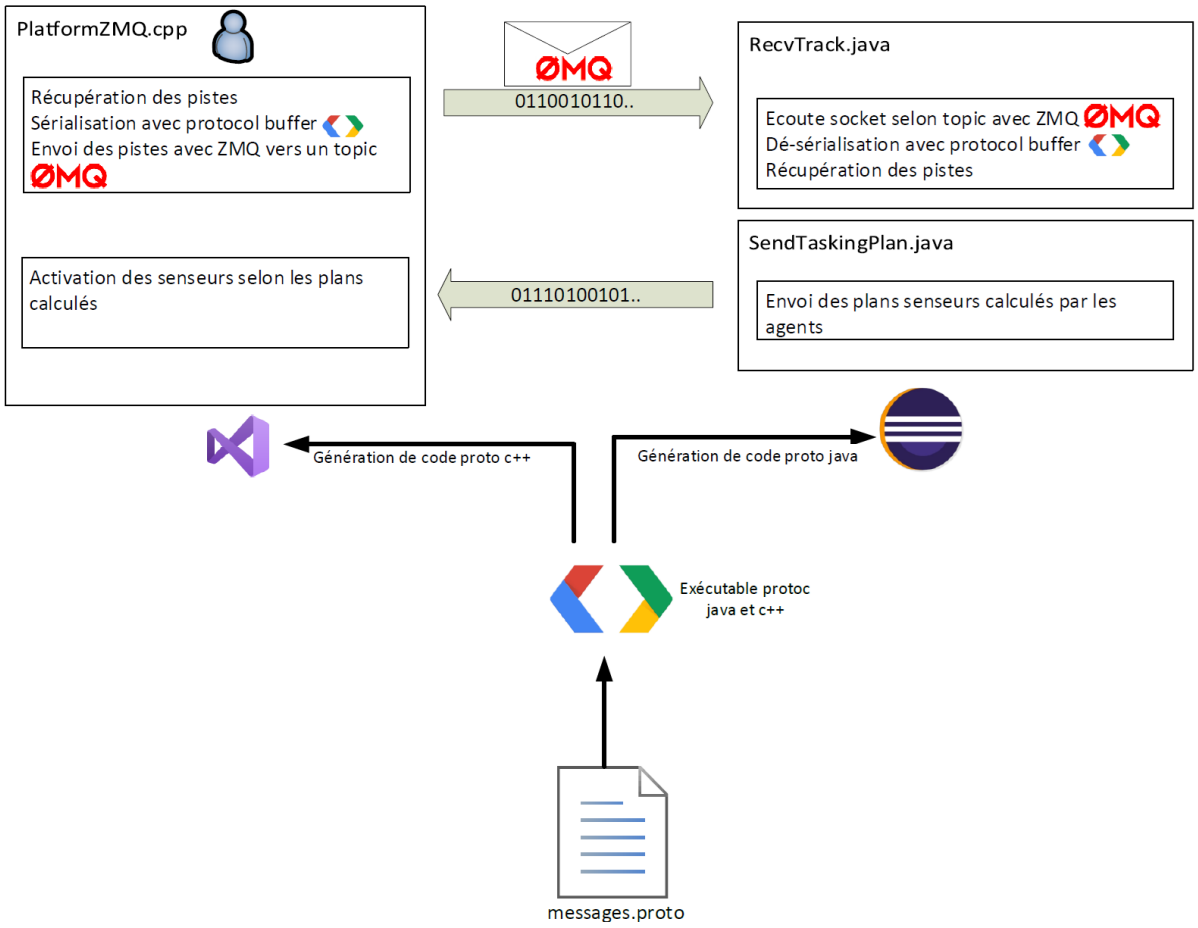

FIGURE 4.3 – Schéma des communications entre les applications avec Protocol Buffers et ZMQ.

Le scénario d'usage, ainsi que les modèles des senseurs et des plateformes seront intégrés dans le simulateur. Le logiciel multi-agent apportera l'intelligence embarquée sur les plateformes simulées. Les agents pourront ainsi capturer l'environnement pour reconstruire un état global. Avec ces données provenant de plateformes et de leurs senseurs, les

agents pourront proposer un ordonnancement de tâches senseurs. Pour nos tests, une seule plateforme de la flotte avait le rôle d'envoyer ses données de pistes. Comme nous avons pu le voir précédemment, l'architecture du logiciel multi-agent est centralisée, toutes les informations sont regroupées sur une plateforme maîtresse.

Le schéma 4.3 présente les échanges entre les deux applications. La plateforme maîtresse récupère les pistes, les sérialise avec *Protocol Buffers* et les envoie avec ZMQ sur le *topic* « Track ». Les informations sur les plateformes sont envoyées sur le *topic* « Platform ». Le logiciel multi-agent écoute sur le *socket* selon les *topics* d'intérêt et récupère la donnée binaire. Il désérialise à l'aide des classes Java générées par *Protobuf*, ce qui permet aux agents de récupérer les pistes mises à jour, puis de calculer de nouveaux plans senseurs. Les plans sont alors envoyés par le même processus vers la plateforme du simulateur pour utiliser les senseurs selon l'ordonnancement proposé. Ces senseurs vont alors enrichir la situation tactique sur les menaces existantes et envoyer les mises à jour aux agents concernés.

4.3.2 Résultats

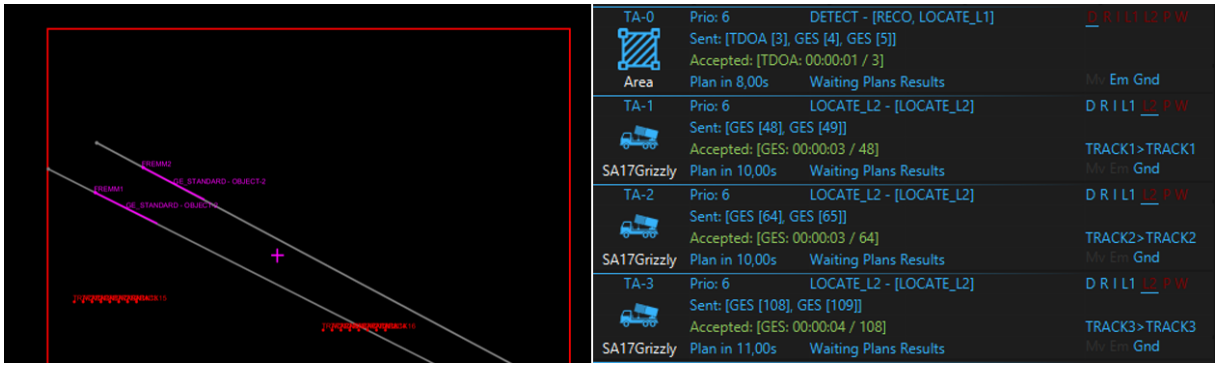

FIGURE 4.4 – Exemple d'un scénario simple de localisation par TDOA.

Ces premiers travaux ont permis de tester le fonctionnement des agents lors d'un scénario opérationnel. Dans celui-ci, nous avons inséré deux plateformes navales et une barrière de batteries sol-air en charge de la défense antiaérienne. Les plateformes ont pour mission de localiser toutes les cibles dans la zone opérationnelle représentée par un rectangle rouge dans la figure 4.4. Lorsque les plateformes sont à portée de détection des menaces, chacune est prise en charge et caractérisée par un agent tactique. Sur la figure, un agent est représenté sur une ligne de l'IHM avec son ID, sa priorité, son objectif, et

les plans senseurs futurs ou en cours. Ces agents vont envoyer les plans acceptés à la plateforme qui sera chargée d'allouer de la ressource senseur pour les accomplir. Nous avons également testé de mettre des cibles en dehors du rectangle rouge et des cibles ne respectant pas les requêtes opérationnelles. La requête opérationnelle est représentée par un agent zone (le premier sur la figure) qui considère uniquement des cibles au sol dans la zone. Ainsi, les cibles aériennes ou les cibles hors zone ne seront pas considérées tant qu'il n'y a pas une nouvelle requête opérationnelle adaptée.

4.3.3 Conclusion

L'interfaçage réussi entre nos deux applications, le logiciel à base d'agents et le simulateur opérationnel, a permis d'analyser les capacités et les limitations de l'approche multi-agent existante. Le système multi-agent permet d'ordonnancer des tâches à des senseurs de manière centralisée depuis une plateforme. Cette centralisation pose problème pour l'adaptation du système et de son réseau, en plus de fournir un unique point de défaillance, elle réduit la capacité de mise à l'échelle et peut provoquer des goulets d'étranglement. De plus, certains éléments sont spécifiques au domaine aérien, comme certains services, ce qui limite l'adaptabilité du système à d'autres domaines d'applications, dont le contexte naval. La complexité du code existant, caractérisée par une architecture monolithique, ainsi que les contraintes de temps de développement ont mis en lumière les défis liés à l'évolution du modèle multi-agent existant vers un modèle distribué et multi-plateforme. Bien que la compréhension de celui-ci soit une étape nécessaire, la modification de l'architecture actuelle s'avère être un défi de taille.

Dans cette optique, nous avons pris la décision de concevoir un banc d'essai. Ce dernier nous permettra d'explorer de nouveaux développements d'agents et de monter en abstraction, offrant ainsi une plateforme idéale pour démontrer les évolutions potentielles de l'architecture actuelle. En parallèle, l'interfaçage de notre logiciel multi-agent avec le simulateur demeure l'un de nos objectifs, d'autant plus que les composants logiciels ont déjà été développés au cours de cette première phase de travail. Ces éléments sont prêts à être intégrés et adaptés dans le cadre d'une nouvelle architecture, permettant ainsi d'évaluer celle-ci d'un point de vue opérationnel.

4.4 Travaux sur RabbitMQ

L'interfaçage des deux logiciels nous permet de connecter une plateforme navale au système multi-agent. L'architecture est donc centralisée, car la plateforme doit définir dans sa base de donnée toutes les plateformes alliées pour connaître les senseurs pouvant être utilisés. De plus, l'allocation de tâches aux senseurs reste centralisée sur cette plateforme, qui doit alors commander les autres. Cette architecture a l'avantage de proposer une planification cohérente, mais la plateforme maîtresse représente un goulot d'étranglement avec toute les nouvelles données des senseurs qui doivent parvenir aux agents pour prendre les bonnes décisions. D'autres inconvénients sont notables, le système n'est pas dynamique, car il doit connaître ses équipements avant le début de la mission. De plus, le passage à l'échelle peut se complexifier, si la plateforme centrale n'en a pas les capacités. Enfin, la défaillance de la plateforme principale pourrait interrompre le fonctionnement du système entier.

Une solution envisagée était d'utiliser un système multi-agent sur chaque plateforme. En distribuant la conception du système, des limitations majeures liées aux capacités du réseau peuvent être rencontrées. En pratique, nous devons composer avec des restrictions de bande passante qui nous empêchent d'échanger d'importantes quantités de données à travers le réseau. Cette contrainte peut entraîner des retards dans la transmission des données, des temps de latence accrus et une diminution des performances globales du système. Le développement d'un banc d'essai avec RabbitMQ permet de tester et d'évaluer nos concepts d'architectures distribuées d'un point de vue réseau, en utilisant un niveau d'abstraction significatif par rapport à l'exploitation d'un logiciel industriel complexe. Les agents seront amenés à communiquer en passant par le serveur RabbitMQ et à publier des messages selon des *topics* permettant de transmettre les informations aux bons destinataires.

Dans un premier temps, nous avons expérimenté l'échange de messages avec RabbitMQ [Rab23]. Nous avons opté pour le langage Java, déjà utilisé pour l'implémentation du modèle des agents de Ramses II. Suite à la prise en main du middleware, nous avons exploité PerfTest, un outil de test de charge pour RabbitMQ, dont nous détaillons les fonctionnalités d'exportation et d'affichage de métriques en annexe. Enfin, nous avons développé un banc d'essai configurable à base de producteurs et de consommateurs, assimilables à des agents communicants, en y ajoutant une partie de gestion du réseau avec les outils Prometheus et Grafana.

4.4.1 *PerfTest* : un outil pour des tests de charge

« Perftest » est un outil développé en Java pour effectuer des tests de performance de RabbitMQ [Per23]. Il simule de la charge et du trafic réseau selon plusieurs configurations et paramètres, tels que le nombre de consommateurs, le nombre de producteurs, les débits d'envoi et de réception, les tailles des files d'attente et des messages (en octets). L'outil peut également générer de la charge aléatoire et ajuster la fréquence de publication des messages selon des intervalles de temps définis.

À titre d'exemple, la commande ci-dessous permet de configurer quatre producteurs et deux consommateurs, qui vont publier et consommer des messages dans la file d'attente « queue1 ». Les producteurs diffusent en mode *fanout* (ou *broadcast*) 100 messages chaque seconde, tandis que les consommateurs se partagent la charge entre eux.

```
java -jar perftest.jar -x 4 -y 2 -r 100 -u "queue1" --metrics-prometheus
    -t fanout
```

Perftest repose sur la librairie *Micrometer* qui permet l'observation et la surveillance des systèmes. *Micrometer* facilite la collecte de métriques provenant de diverses sources et les exporte vers différents systèmes de monitoring.

L'option « metrics-prometheus » permet d'activer l'export de métriques vers un serveur Prometheus. L'outil exporte ses propres métriques en plus des métriques activables sur RabbitMQ, nous permettant ainsi d'implémenter notre propre exportation de métriques dans notre programme. Le tableau 4.1 résume les différentes métriques observables.

	Métriques		
Prometheus PerfTest	Message/s (consommés/-produits)	Latence (en percentiles)	Somme de la latence /latence max
Clients PerfTest	Messages non routés	Nombre de canaux/connection	Total messages publiés, rejetés, consommés
Mémoire	Nombre buffers	Quantité mémoire JVM	Capacité totale des buffers
Threads	Nombre de threads état bloquant	Nombre de threads actifs	
CPU	Nombre de processeurs disponibles JVM	% utilisation CPU JVM	% utilisation CPU globale

TABLE 4.1 – Exemples de métriques exportées vers Prometheus.

Les expériences menées sur PerfTest ont mis en lumière certains points importants : l'outil, conçu à l'origine pour des tests de charge réseau, ne répond pas entièrement à nos besoins en raison de ses limitations et de son manque de flexibilité. PerfTest permet de générer une latence, fixe ou variable, un débit, ainsi qu'une taille pour les messages. Cependant, ces paramètres sont appliqués globalement et ne peuvent pas être ajustés individuellement pour chaque producteur ou chaque consommateur. Ces observations nous ont permis de conclure que PerfTest ne correspondait pas pleinement à nos objectifs de recherches, à savoir l'évaluation des échanges entre les agents de notre système. Le tableau 4.2 résume cette évaluation tout en identifiant les fonctionnalités que nous aimerions voir intégrées dans un tel outil.

Cependant, il est important de noter que PerfTest offre l'avantage d'être un projet open-source, ce qui le rend accessible à la communauté. Nous avons également trouvé intéressante l'intégration de Prometheus pour l'exportation de métriques, offrant ainsi la possibilité d'obtenir des résultats visuels. Par ailleurs, un tableau de bord Grafana a été spécialement conçu pour mettre en évidence les résultats des tests de charge. L'utilisation conjointe de Grafana et de Prometheus dans le cadre de PerfTest constitue donc un excellent exemple pour illustrer leurs fonctionnements (détaillé en annexe). La disponibilité du code source nous permet d'adapter la partie liée à l'exportation de métriques selon nos besoins spécifiques.

Points positifs	Points négatifs	Mécanismes souhaités
Des métriques déjà en place	Taille des messages identique pour chaque producteur	Taille des messages différente d'un producteur à un autre
Nombreux paramètres	Difficulté de connecter les files d'attente aux producteurs voulus	Choisir un nombre de producteurs qui envoient des messages à N consommateurs
Possibilité de lancer plusieurs PerfTest sur plusieurs ports	Globalité des métriques (latences, débit, taille message)	métriques pour chaque producteur/consommateur
Outils de visualisation graphique utilisant JS/HTML/JSON		Pouvoir définir un nombre de producteurs qui envoient à d'autres consommateurs via d'autres files d'attente

TABLE 4.2 – Evaluation de PerfTest par rapport aux fonctionnalités désirées.

Ces premier travaux sur RabbitMQ et l'outil PerfTest se révèlent être constructifs pour nos travaux à plusieurs égards. Tout d'abord, PerfTest représente un exemple concret et pratique de l'utilisation de RabbitMQ dans un contexte de tests de charge. En permettant l'exportation des métriques sur Prometheus, il offre un visuel synthétique sur les performances et le comportement du système. De plus, la disponibilité d'un *dashboard* sur Grafana pour visualiser ces métriques en fait un modèle pour l'exportation et l'analyse de données dans nos propres développements. En outre, l'aspect open source de PerfTest nous permet de réutiliser certains éléments de code dans la suite de nos travaux, offrant ainsi une base solide pour le développement et l'amélioration de notre application. En somme, PerfTest constitue une ressource précieuse qui enrichit notre compréhension de RabbitMQ et de ses applications potentielles.

4.4.2 Développement d'un banc d'essai

Dans la continuité des travaux précédents et inspirés par l'outil PerfTest, nous entamons le développement d'un banc d'essai dédié. Axé sur l'utilisation de RabbitMQ ainsi que sur les outils de surveillance Grafana et Prometheus, ces nouveaux travaux ont pour but de fournir un environnement pour l'analyse et la mesure de performances sur notre future architecture. Nous cherchons à mettre en place une plateforme d'évaluation flexible, permettant l'exportation de métriques via Prometheus et leur visualisation dynamique grâce à Grafana. Cette approche est un atout pour la suite de nos recherches, offrant un framework pour tester nos solutions d'un point de vue du réseau. L'architecture du banc d'essai développé comprend donc RabbitMQ pour la communication, Prometheus pour l'exportation des métriques du système, et Grafana pour la visualisation des métriques sur des graphes.

Conditions de l'expérience

Le banc d'essai permet la transmission de messages entre des producteurs et des consommateurs en fonction des différents *topics* utilisés par le paradigme *publish/subscribe*. L'objectif premier du banc d'essai était de créer un environnement de simulation permettant l'observation et la surveillance des métriques du réseau. Par la suite, nous avons configuré la taille des messages émis ainsi que la fréquence de leur distribution. Enfin, nous avons collecté des métriques et des traces pour analyser le réseau en exploitant les fonctionnalités offertes par Prometheus et Grafana.

Notre configuration repose sur l'utilisation des serveurs Prometheus et Grafana. La flexibilité de notre solution est illustrée par l'exemple de configuration présenté dans la figure 4.5, qui peut être facilement modifiée à l'aide d'un fichier CSV. Nos tests sont effectués sur un ordinateur portable sous *Windows 10 Pro*, avec *16 Go de RAM* et un processeur *Intel Core i7-6820HQ*. En ce qui concerne l'exportation des métriques, nous utilisons la librairie *Micrometer*, qui assure la prise en charge de Prometheus.

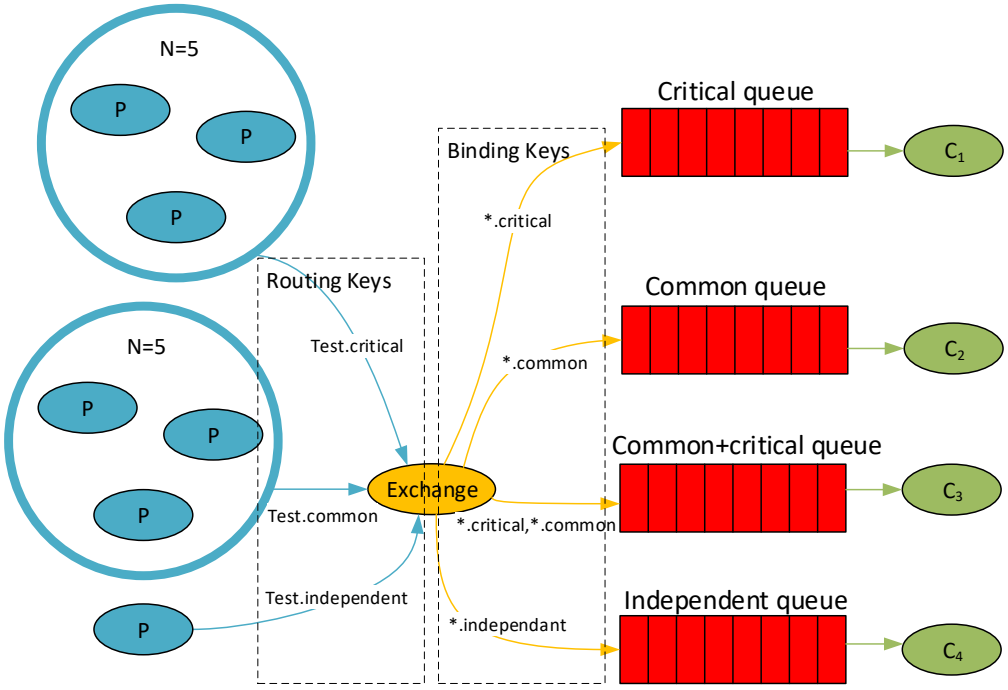

FIGURE 4.5 – Exemple de configuration avec 11 producteurs, 4 consommateurs et 3 topics différents.

Résultats

A cette étape de l'étude, nous produisons des métriques et nous les présentons dans des *dashboards* qui regroupent de nombreuses informations sur RabbitMQ et les échanges entre les producteurs et les consommateurs. Ce banc d'essai nous permettra d'évaluer l'architecture proposée, en considérant les producteurs et les consommateurs comme des entités devant échanger via le réseau. Dans le cadre de nos études, nos plateformes militaires peuvent être représentées par des producteurs et des consommateurs de messages.

Nous envisageons de tester le passage à l'échelle des architectures centralisées et distribuées avec le banc d'essai.

Nous voulions obtenir des analyses du trafic avec les *dashboards* de Grafana sur : le débit, la congestion et la latence. Dans le cadre du banc d'essai, nous avons fait des tests selon certaines configurations, dont l'exemple de la figure 4.5. Plusieurs *raws* ont été créés, ce sont en quelques sortes des onglets comprenant différents graphes. Le premier regroupe les informations générales comme le débit entrant et sortant du serveur RabbitMQ, et la quantité de producteurs, de consommateurs, de files d'attentes, de canaux et de connexions que nous avons configurée. Ces informations sont illustrées par la figure 4.6, le débit sortant est plus important que le débit entrant, car les messages sont dupliqués sur plusieurs files d'attente. Par exemple, les consommateurs C_1 et C_3 récupèrent tous les deux les messages de cinq producteurs.

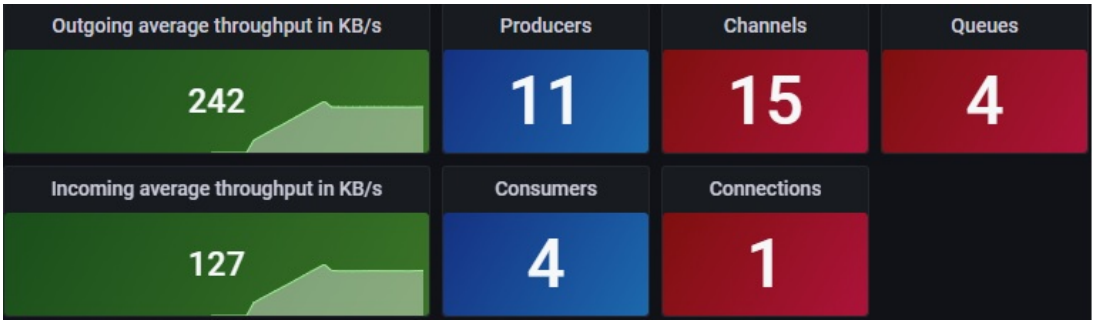

FIGURE 4.6 – Dashboard comprenant des informations de RabbitMQ.

Nous avons également exporté et présenté de nouvelles métriques, telles que les débits de chaque consommateur et de chaque producteur. Ces métriques permettront d'observer la quantité d'information à échanger avec l'architecture proposée. Le cas échéant, il conviendra d'adapter la fréquence des échanges pour maintenir son fonctionnement. RabbitMQ gère également la congestion. Lorsque la vitesse de consommation des messages par le consommateur est limitée en raison d'une latence de celui-ci, nous pouvons observer, sur les graphes (figure 4.7), que le producteur réduit également son débit d'envoi de messages.

Conclusion

L'utilisation de RabbitMQ dans ces travaux préliminaires nous a permis de simuler des échanges entre des entités logicielles. Les travaux suivants intègrent les concepts de

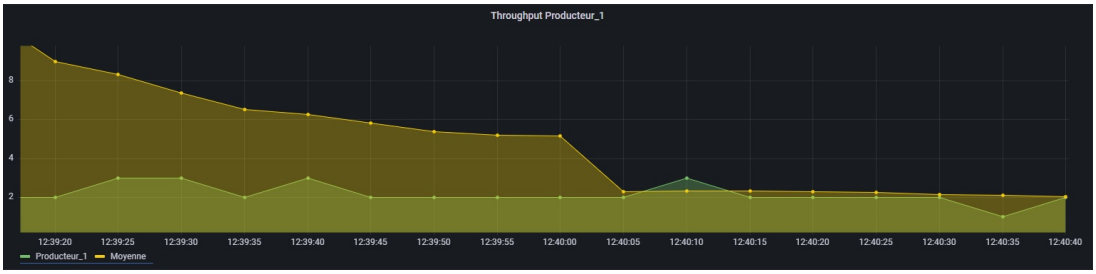

FIGURE 4.7 – Graphe avec le débit d'un producteur.

producteurs et de consommateurs à des agents d'un SMA. Ces agents pourront communiquer avec des agents situés sur d'autres plateformes. La complexité du système augmente selon le nombre de plateformes présentes sur le théâtre des opérations. Ainsi, nous utiliserons RabbitMQ pour les échanges de données entre les agents qui sont physiquement séparés sur des plateformes distinctes. Les agents coopèreront en produisant et consommant des messages. Les outils de monitoring nous permettront d'analyser et d'évaluer l'approche multi-agent proposée. Nous observerons les limites de l'architecture proposée, principalement du point de vue du réseau, dans le but de faire évoluer et d'améliorer ces concepts.

Une première version d'un framework à base d'agents fut proposée en suivant les concepts proposés dans le chapitre 3, elle intègre une première expérimentation d'échanges entre les agents avec RabbitMQ. Les agents sont des *threads* Java et la gestion des échanges se fait de manière asynchrone. Cette version a l'avantage de fonctionner avec le simulateur opérationnel, c'est-à-dire que chaque plateforme et chaque cible sont modélisées par des agents plateformes et des agents tactiques en Java, les agents plateformes utilisent RabbitMQ pour échanger leurs ressources. L'architecture logicielle de ce premier développement non achevé est présentée sur la figure 4.8.

Cette simulation opérationnel ayant quelques défauts, notamment des problèmes d'accès concurrentiels aux ressources, nous l'avons mise en pause pour développer un modèle séquentiel, sans thread, et sans connexion avec un simulateur, qui étaient deux sources de problèmes nous contraignant durant nos démonstrations de concepts d'agents. Cette version du framework se voulait assez concrète, avec des services et des ressources réalistes, ce qui amenait des problématiques d'utilisation des services complexes et peu utiles pour la preuve de concepts de nos agents. Le nouveau banc d'essai corrige également cette complexité en proposant des modèles de ressources et de services simples. Par rapport à

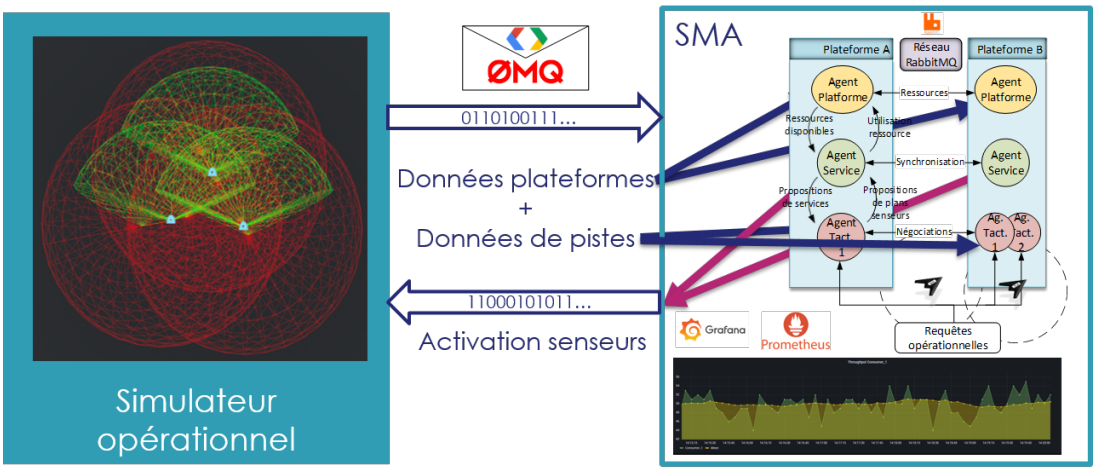

FIGURE 4.8 – Schéma de l'architecture logicielle.

l'architecture présentée en figure 4.8, le simulateur et sa connexion ZMQ sont remplacés par un script Java qui permet directement la création d'agents plateformes et d'agents tactiques. La première version séquentielle n'intègre pas de réseau et de monitoring qui seront ajoutés dans la version distribuée avec RabbitMQ. Nous présentons ce nouveau framework et son évolution dans la section suivante.

4.5 Framework multi-agent pour démonstration de concepts CCN

4.5.1 Première version séquentielle

L'intérêt de cette nouvelle version était de simplifier le modèle pour valider les concepts d'agents proposés. Pour cela, nous avons implémenté une boucle qui exécute la machine à état de chaque agent à chaque pas de simulation. Ce modèle rend le système totalement synchrone, les agents ne pourront ainsi pas accéder à une même ressource en même temps, car leurs actions ne font que se succéder. Avec ce programme, nous pouvons intégrer de nouveaux agents en cours de simulation, en définissant la création d'un agent à un certain pas de la simulation, et créer autant d'agents que nécessaire. Cette approche nous a donc permis de tester la mise à l'échelle de nos algorithmes. L'ajout d'une nouvelle plateforme a un impact sur le nombre de services proposés et donc sur les différents calculs. L'exécution d'un algorithme dans la chaine de décision doit se faire assez rapidement, car l'information

ainsi que les services proposés peuvent devenir rapidement obsolètes.

Conditions de l'expérience

Le framework a été développé sous Java 17. Nos tests sont effectués sur un ordinateur portable sous *Windows 11 Pro*, avec *32 Go de RAM* et un processeur *Intel Core i7-1185G7*.

Nous pouvons paramétrer le nombre de plateformes dans la simulation qui conditionnera la création d'agents plateformes et d'agents services, ainsi que le nombre de ressources disponibles sur chaque plateforme. L'énergie de la ressource peut-être modifiée, elle correspond aux nombres de services que la ressource peut accomplir en simultanée. Nous modifierons ces différents paramètres selon l'algorithme considéré.

Le nombre de cibles est également paramétrable en début de simulation. Il est possible de définir le pas exact de l'apparition d'une nouvelle cible, ainsi que d'une nouvelle plateforme alliée dans le but d'observer le comportement du système lors de l'allocation de tâches. Les propriétés des ressources peuvent être modifiées : les contraintes, les types de services que la ressource peut accomplir et l'énergie totale qu'un service peut utiliser pour la ressource. Il en est de même pour les propriétés d'un service, comme le nombre de ressources qu'il nécessite, les objectifs qu'il peut accomplir et le coût en énergie de l'utilisation du service. De nouvelles ressources et de nouveaux services peuvent être ajoutés à la liste. L'augmentation du nombre de plateformes aura un impact sur le nombre de ressources.

Le simulateur ne modélise pas les communications entre les plateformes, qui peuvent communiquer directement sans latence. Les ressources et les services possèdent un niveau d'abstraction élevé ne reflétant pas la réalité des théâtres d'opération. Un service peut arriver au bout de son exécution ou être annulé, mais nous ne jugeons pas de la qualité d'exécution des services qui demanderait des modèles plus complexes. Chaque service s'exécute pour une durée fixe qui correspond à un nombre de pas de simulations. Le coût d'annulation d'un service en cours n'est pas considéré, donc un plan avec une performance plus importante peut annuler un autre même si le gain est négatif en raison du coût de l'annulation.

Observation du système multi-agent

Afin de faciliter l'observation du système en fonctionnement et de vérifier le comportement des agents, nous avons développé une IHM simple sous forme de tableaux compre-

nant les connaissances des agents. La figure 4.9 présente l'IHM et ses différents onglets. Le premier onglet permet de définir les paramètres de base de la simulation : le nombre de plateformes et le nombre de menaces dans le théâtre d'opérations. Le paramétrage des services et des ressources se fait directement sur les objets concernés.

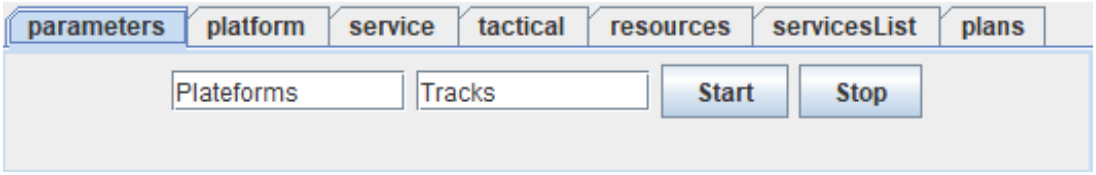

FIGURE 4.9 – IHM permettant de définir les paramètres de la simulation.

Le second onglet, ouvert sur la figure 4.10, présente les plateformes avec leur position, leur liste de ressources après échange avec les autres plateformes, et la liste des requêtes opérationnelles qui contraignent le choix des plans des agents tactiques.

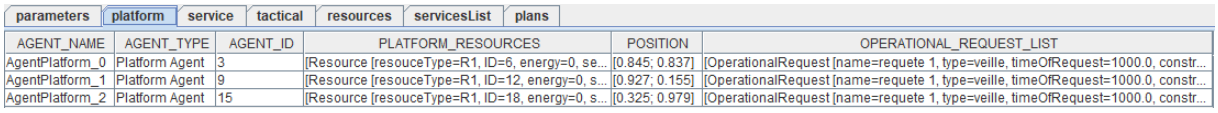

FIGURE 4.10 – Liste des agents plateformes et des données dans leur mémoire.

L'onglet « service » de la figure 4.11 présente la liste des agents services actifs sur les différentes plateformes et l'onglet de la figure 4.12 la liste des services que ces agents proposent. Chaque service possède une performance qui varie au cours du temps. Nous pouvons également observer les ressources qu'il nécessite, ainsi que sa durée d'exécution.

FIGURE 4.11 – Liste des agents services.

Les agents tactiques sont présentés en figure 4.13, avec leur position, la plateforme sur laquelle ils sont exécutés et leur objectif opérationnel actuel. Un agent tactique a terminé tout ses objectifs lorsque l'état de son « CURRENT_OO » est à « TRACK », il effectuera

parameters | platform | service | tactical | resources | servicesList | plans

ServiceName	Performance	Resources	TimeOfService
S2_0	277.2629873889175	[{R1; AgentPlatform_2}, {R1; AgentPlatform_1}]	10000.0
S2_1	277.2629873889175	[{R1; AgentPlatform_2}, {R2; AgentPlatform_1}]	10000.0
S2_2	288.0904885118093	[{R1; AgentPlatform_2}, {R1; AgentPlatform_0}]	10000.0
S2_3	288.0904885118093	[{R1; AgentPlatform_2}, {R2; AgentPlatform_0}]	10000.0
S2_4	277.1629873889175	[{R2; AgentPlatform_2}, {R1; AgentPlatform_1}]	10000.0
S2_5	277.1629873889175	[{R2; AgentPlatform_2}, {R2; AgentPlatform_1}]	10000.0
S2_6	287.99048851180936	[{R2; AgentPlatform_2}, {R1; AgentPlatform_0}]	10000.0
S2_7	287.99048851180936	[{R2; AgentPlatform_2}, {R2; AgentPlatform_0}]	10000.0
S2_8	284.9580138843652	[{R1; AgentPlatform_1}, {R1; AgentPlatform_0}]	10000.0
S2_9	284.9580138843652	[{R1; AgentPlatform_1}, {R2; AgentPlatform_0}]	10000.0
S2_10	284.85801388436516	[{R2; AgentPlatform_1}, {R1; AgentPlatform_0}]	10000.0
S2_11	284.85801388436516	[{R2; AgentPlatform_1}, {R2; AgentPlatform_0}]	10000.0
S4_0	277.3629873889175	[{R3; AgentPlatform_2}, {R3; AgentPlatform_1}]	10000.0
S4_1	288.19048851180935	[{R3; AgentPlatform_2}, {R3; AgentPlatform_0}]	10000.0
S4_2	284.9580138843652	[{R3; AgentPlatform_1}, {R3; AgentPlatform_0}]	10000.0
S1_0	277.2629873889175	[{R1; AgentPlatform_2}, {R1; AgentPlatform_1}]	10000.0
S1_1	288.0904885118093	[{R1; AgentPlatform_2}, {R1; AgentPlatform_0}]	10000.0
S1_2	284.9580138843652	[{R1; AgentPlatform_1}, {R1; AgentPlatform_0}]	10000.0
S3_0	250.0	[{R2; AgentPlatform_2}]	10000.0
S3_1	250.2	[{R3; AgentPlatform_2}]	10000.0
S3_2	250.1	[{R2; AgentPlatform_1}]	10000.0
S3_3	250.2	[{R3; AgentPlatform_1}]	10000.0
S3_4	250.0	[{R2; AgentPlatform_0}]	10000.0
S3_5	250.2	[{R3; AgentPlatform_0}]	10000.0

FIGURE 4.12 – Liste des services proposés par l'agent service.

dans l'ordre « RECOGNIZE, IDENTIFY, LOCATE, TRACK » en se servant de services permettant d'atteindre chaque objectif.

L'onglet de ressources de l'IHM, que nous pouvons observer en figure 4.14, affiche l'état actuel des différentes ressources et des services qui sont en cours d'exécution avec leur pourcentage d'avancement. Par exemple, nous pouvons observer que le service $S2_2$ est en cours sur les ressources 6 et 18, qui sont des ressources de type $R1$. Par ailleurs, la ressource 6 est capable d'effectuer deux services qui demandent une énergie chacun, elle effectue donc le service $S2_6$ en parallèle.

Enfin, l'onglet *plans* affiche les plans des agents tactiques proposés à l'agent service, le statut évolue au cours de la simulation ainsi que les scores calculés.

Tests de priorité sur l'utilisation des ressources

Nous avons créé un agent tactique au bout de 100 pas de simulation alors que 500 agents tactiques sont en cours d'exécution et de planification. Cet agent tactique a un paramètre qui permet de le définir comme étant un souhait de l'opérateur d'utiliser de la ressource pour une cible d'intérêt sans considérer l'avis du système. Le paramètre

parameters | platform | service | tactical | resources | servicesList | plans

AGENT_NAME	AGENT_ID	POSITION	PLATFORM_APPARTENANCY	CURRENT_OO
Target0	21	[0.135; 0.298]	[AgentPlatform_0]	RECOGNIZE
Target1	23	[0.529; 0.631]	[AgentPlatform_0]	IDENTIFY
Target2	25	[0.373; 0.636]	[AgentPlatform_0]	IDENTIFY
Target3	27	[0.146; 0.611]	[AgentPlatform_0]	IDENTIFY
Target4	29	[0.943; 0.186]	[AgentPlatform_0]	IDENTIFY
Target5	31	[0.361; 0.124]	[AgentPlatform_1]	RECOGNIZE
Target6	33	[0.013; 0.066]	[AgentPlatform_1]	RECOGNIZE
Target7	35	[0.837; 0.442]	[AgentPlatform_1]	RECOGNIZE
Target8	37	[0.423; 0.51]	[AgentPlatform_1]	IDENTIFY
Target9	39	[0.306; 0.777]	[AgentPlatform_1]	RECOGNIZE
Target10	41	[0.89; 0.125]	[AgentPlatform_2]	RECOGNIZE
Target11	43	[0.388; 0.998]	[AgentPlatform_2]	RECOGNIZE
Target12	45	[0.126; 0.528]	[AgentPlatform_2]	RECOGNIZE
Target13	47	[0.488; 0.708]	[AgentPlatform_2]	RECOGNIZE
Target14	49	[0.76; 0.667]	[AgentPlatform_2]	RECOGNIZE
Target15	51	[0.084; 0.694]	[AgentPlatform_0]	RECOGNIZE

FIGURE 4.13 – Liste des agents tactiques et des données dans leur mémoire.

parameters | platform | service | tactical | resources | servicesList | plans

ResourceName	ServicesRunning	EnergyMax	Energy	ServicesHandled	ResourceType
Resource_6	S2_6 Target10 : 3% || S2_2 Target14 : 3% ||	2	0	[S1, S2]	R1
Resource_7	S2_11 Target12 : 3% || S2_7 Target15 : 50% ||	2	0	[S2, S3]	R2
Resource_8		2	2	[S3, S4]	R3
Resource_12		2	2	[S1, S2]	R1
Resource_13	S3_2 Target4 : 2% || S2_11 Target12 : 3% ||	2	0	[S2, S3]	R2
Resource_14		2	2	[S3, S4]	R3
Resource_18	S2_2 Target14 : 3% ||	2	1	[S1, S2]	R1
Resource_19	S2_6 Target10 : 3% || S2_7 Target15 : 50% ||	2	0	[S2, S3]	R2
Resource_20	S3_1 Target8 : 2% || S3_1 Target3 : 0% ||	2	0	[S3, S4]	R3

FIGURE 4.14 – Liste des ressources et de leurs utilisations.

permet d'élever le niveau de performance des plans proposés par cet agent, ce qui force donc l'annulation d'un service en cours pour passer son meilleur plan. L'opérateur peut décider d'annuler la priorité sur la cible lorsqu'il considère qu'elle n'a plus d'intérêt. Nous observons que la chaine DRIL de la cible se termine avant les autres, par le mécanisme de préemption des plans.

Temps de calcul des algorithmes

Cette première version nous permet d'affirmer, en pratique, la complexité de nos algorithmes en terme de temps de calcul en fonction du nombre de plateformes, et donc du nombre de services. Le framework étant séquentiel, l'augmentation du nombre de pla-

parameters | platform | service | tactical | resources | servicesList | plans

PlatformAppartenancy	TacticalAgent	PlanScore	PlanStatus
[AgentPlatform_2]	Target14	712.7693476604773	REFUSED
[AgentPlatform_2]	Target14	709.4034443039129	REFUSED
[AgentPlatform_2]	Target14	701.8596042468989	REFUSED
[AgentPlatform_2]	Target14	690.2693476604773	REFUSED
[AgentPlatform_2]	Target14	690.2693476604773	REFUSED
[AgentPlatform_2]	Target14	686.9034443039129	REFUSED
[AgentPlatform_2]	Target14	686.9034443039129	REFUSED
[AgentPlatform_2]	Target14	679.3596042468989	REFUSED
[AgentPlatform_2]	Target14	679.3596042468989	REFUSED
[AgentPlatform_2]	Target14	673.7693476604773	ACCEPTED

FIGURE 4.15 – Liste des plans envoyés par les agents tactiques.

teformes impactera la durée totale de simulation et donc le temps mis par chaque agent pour compléter un service. Il n'est donc pas utile de relever cette durée. Par exemple, si nous augmentons le nombre de plateformes de la simulation, en passant de 10 à 100 plateformes, et que nous considérons 100 menaces au total dans les deux cas. Le système passera de 120 agents à 300 agents, ce qui représente trois fois plus d'agents à exécuter à chaque pas de simulation, avec des calculs plus lourds à traiter, car les algorithmes proposeront une plus grande quantité de services. Nous relèverons donc le temps de calcul des différents algorithmes en fonction de différents paramètres, comme le nombre d'agents tactiques, le nombre d'agents plateformes, le nombre de ressources par plateforme ou les types de services disponibles.

Pour ces simulations, les agents plateformes, tactiques et services s'exécutent tous au démarrage. Nous avons testé les algorithmes vus en session 3.6.2 et numérotés respectivement de 1 à 5 : *checkIfResourceIsAvailable*, *updatesServices*, *resourceAllocation*, *calculatePlan* et *checkPlanStatus*. Les pistes sont actives sur une seule plateforme et nous avons fait la moyenne sur 100 simulations. Un seul agent service sera actif, car une plateforme sans piste ne planifiera rien, cependant la plateforme en charge des menaces pourra allouer des tâches à des ressources sur d'autres plateformes.

Algorithme *checkIfResourceIsAvailable.* Le temps de calcul de l'algorithme 1 dépend du nombre de plans proposés et de l'énergie d'une ressource, car l'algorithme regarde également les plans en cours pour savoir si les nouveaux plans peuvent récupérer les ressources. Pour nos tests, nous définissons le nombre d'agents tactiques cinq fois supérieur à l'énergie pour s'assurer que la ressource sera pleinement utilisée. Nous paramétrons une

unique ressource sur chacune des deux plateformes, ainsi qu'un seul type de service. Les agents tactiques restent sur le même objectif opérationnel pendant toute la durée de l'exécution, et se situent tous sur une seule des deux plateformes. Nous faisons varier l'énergie jusqu'à 200 pour observer l'évolution du temps de calcul.

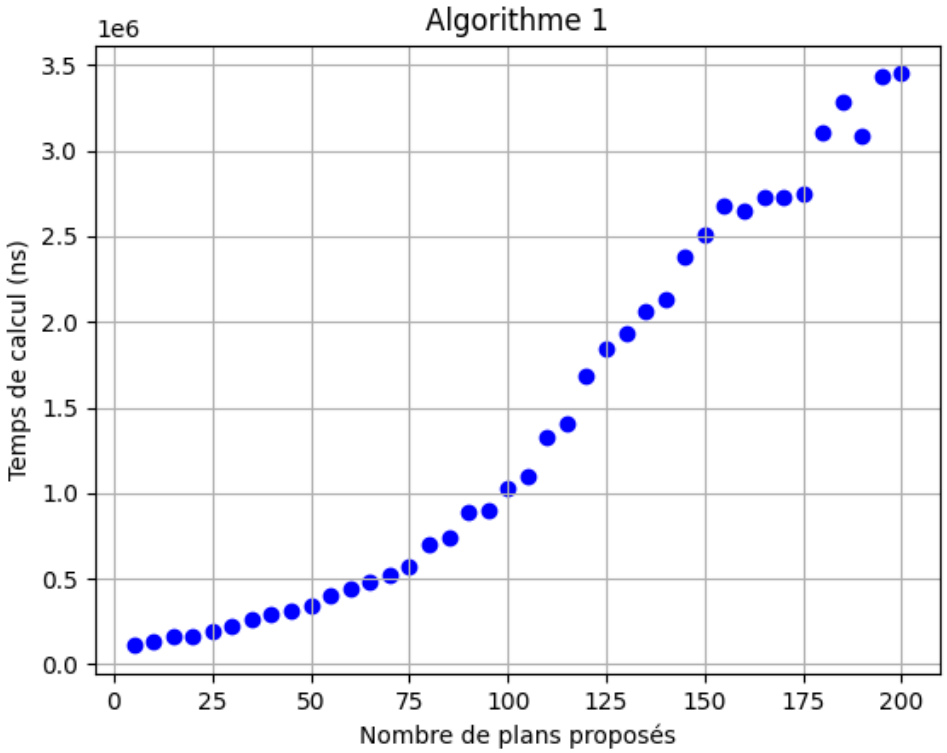

FIGURE 4.16 – Temps de calcul de l'algorithme 1 en fonction du nombre de plans proposés.

D'après les résultats de la figure 4.16, l'algorithme présente un temps de calcul de quelques millisecondes. Cette fonction est appelée par l'agent plateforme à chaque fois qu'il reçoit de nouvelles propositions de plans de l'agent service. La complexité, dans le cas où le nombre d'agents tactiques est largement supérieur aux ressources disponibles pour effectuer les services, devient polynomiale. Le nombre de plans proposés sera au maximum égal à la quantité de ressources disponibles. Dans notre exemple, il y aura jusqu'à 200 plans proposés pour une seule ressource.

Algorithme *updatesServices*. L'algorithme 2 dépend du nombre de types de service, du nombre de ressources pouvant effectuer les services, ainsi que du nombre total de

ressources. Pour faire évoluer le nombre de ressources, nous fixons trois ressources par plateformes et nous faisons varier le nombre de plateformes et de types de services. Chaque service utilise les mêmes ressources et permet d'effectuer les mêmes objectifs. La figure 4.17 présente des résultats pour un nombre de plateformes variant entre 4 et 16, c'est-à-dire que l'agent service peut utiliser entre 12 et 48 ressources disponibles pour chaque type de services.

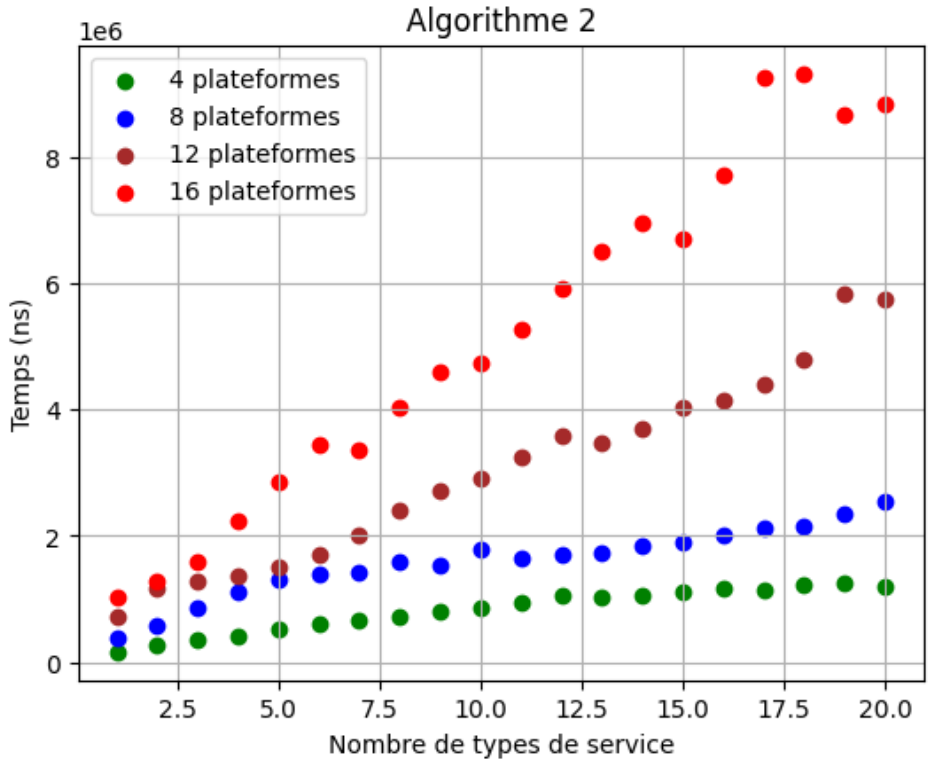

FIGURE 4.17 – Temps de calcul de l'algorithme 2 en fonction du nombre de plateformes et de types de services.

Nous pouvons clairement observer les limites de cette fonction de mise à jour des services, qui a été développée à la base pour tester les limites de l'agent tactique lorsqu'un grand nombre de service lui est proposé. Dans le cas où 16 plateformes possèdent chacune 3 ressources et que chaque ressource peut effectuer ce type de service, la formule 3.1 donne un total de 1080 combinaisons pour un même service. Donc, lorsqu'il y a 20 types de services, l'agent service propose une liste de 21600 services. Dans la réalité, le nombre de services possibles sera nettement inférieur, et la fonction de l'agent service pour les

calculer sera également différente. La liste de services peut être définie en préparation de mission, l'agent service n'aura plus qu'à apposer une performance sur chacun des services.

Algorithme *resourceAllocation*. Cet algorithme dépend du nombre de plans proposés par les agents tactiques à l'agent service pour l'allocation locale des ressources senseurs. Pour nos simulations, nous fixons un seul type de service et le nombre de ressources par plateforme à 3. L'augmentation du nombre de plateformes aura un impact sur le nombre de services possibles, tandis que le nombre de cibles influera sur le nombre de plans en entrée de l'algorithme. Pour nos tests, nous imposons l'exécution des agents tactiques sur une seule plateforme.

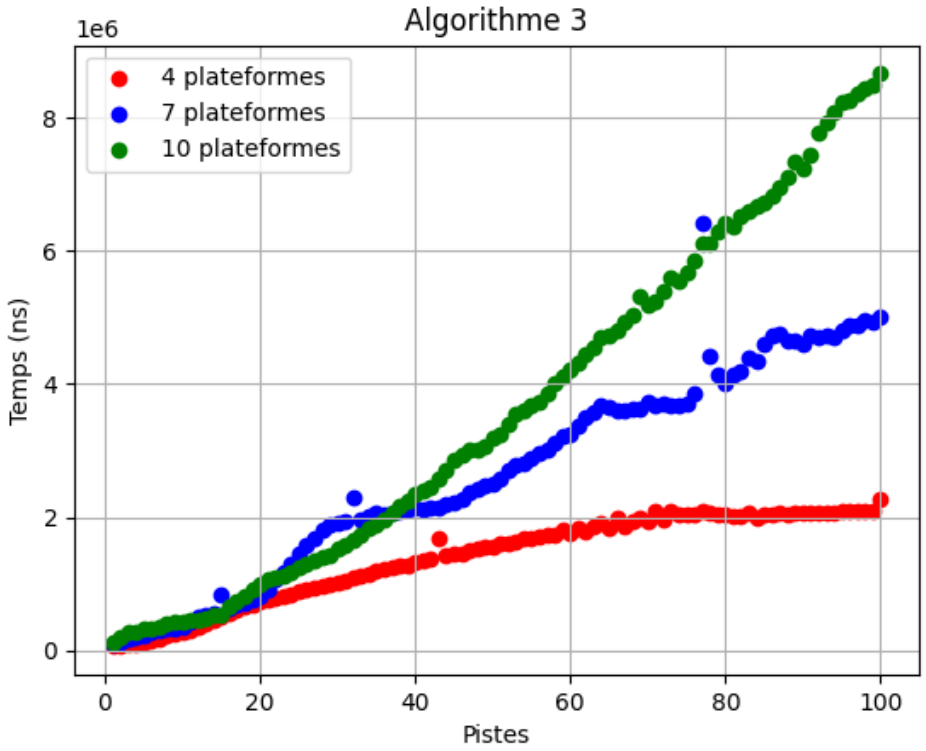

FIGURE 4.18 – Temps de calcul de l'algorithme 3 en fonction du nombre de plateformes et de pistes.

La figure 4.18 présente les résultats pour 4,7 et 10 plateformes, avec le nombre d'agents tactiques variant de 1 à 100. Sur ce graphe, nous ne pouvons pas clairement observer la complexité quasi-linéaire exprimée en théorie dans le chapitre précédent, car celle-ci dé-

pend du nombre de plans proposés. Ce nombre résulte du nombre de plateformes selon la formule 3.1 multiplié par le nombre de pistes, car chaque agent tactique proposera ce nombre de plans dans le pire des cas. Le temps de calcul affiché est de quelques millisecondes, et pour une seule plateforme. Si nous considérons 10 plateformes avec chacune 100 pistes, le temps de calcul restera similaire et le système parviendra donc à traiter 1000 agents tactiques au total. Cette répartition des agents tactiques sur les plateformes permet donc d'équilibrer la charge de calcul pour l'exécution des algorithmes.

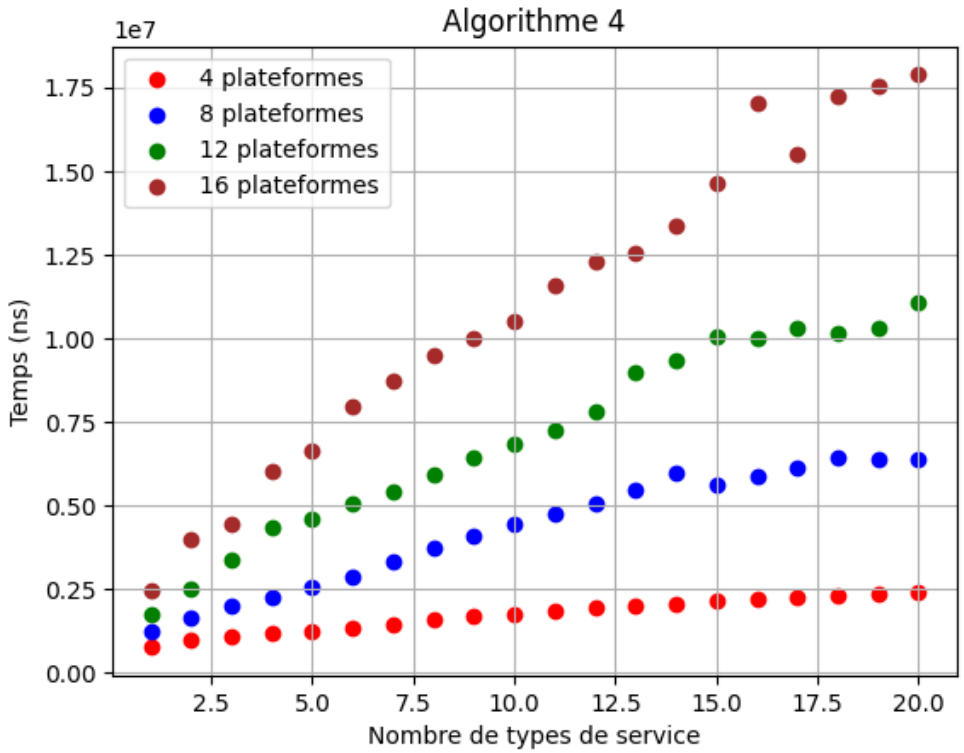

FIGURE 4.19 – Temps de calcul de l'algorithme 4 en fonction du nombre de plateformes et de types de services.

Algorithme *calculatePlan*. Cet algorithme et l'algorithme suivant *checkPlanStatus* dépendent tous deux des mêmes paramètres. Le nombre de services proposés à l'agent tactique, et donc indirectement le nombre de ressources et de plateformes, impacte les temps de calculs de ces deux algorithmes. Pour nos tests, nous proposons de faire varier la quantité de types de services et le nombre de plateformes. Pour accélérer la simulation, nous déclarons un seul agent tactique qui exécute les algorithmes. Les plateformes possèdent 3 ressources chacune, et comme précédemment, chaque ressource permet d'effectuer tous les services proposés.

Dans notre simulation, l'agent tactique propose un nombre de plans quasiment égal au nombre de services car nous supposons que tous les services servent pour les objectifs de l'agent tactique et que les scores des services sont tous acceptables pour être utilisés. Les résultats sur la figure 4.19 démontrent que l'agent tactique peut prendre du temps sur l'exécution de cette fonction s'il doit prendre en compte chaque service qui lui est

proposé. Une réduction du nombre de services en entrée peut réduire cette complexité, notamment en discriminant en amont les services de mauvaises qualités ou les services ne répondant pas aux objectifs opérationnels de l'agent tactique.

Ces résultats valent pour un seul agent tactique, donc il faut multiplier ces valeurs par le nombre d'agents tactiques sur la plateforme pour obtenir la charge de calcul totale nécessaire. Dans notre cas, nous ne parallélisons pas les calculs des différents agents tactiques d'une même plateforme.

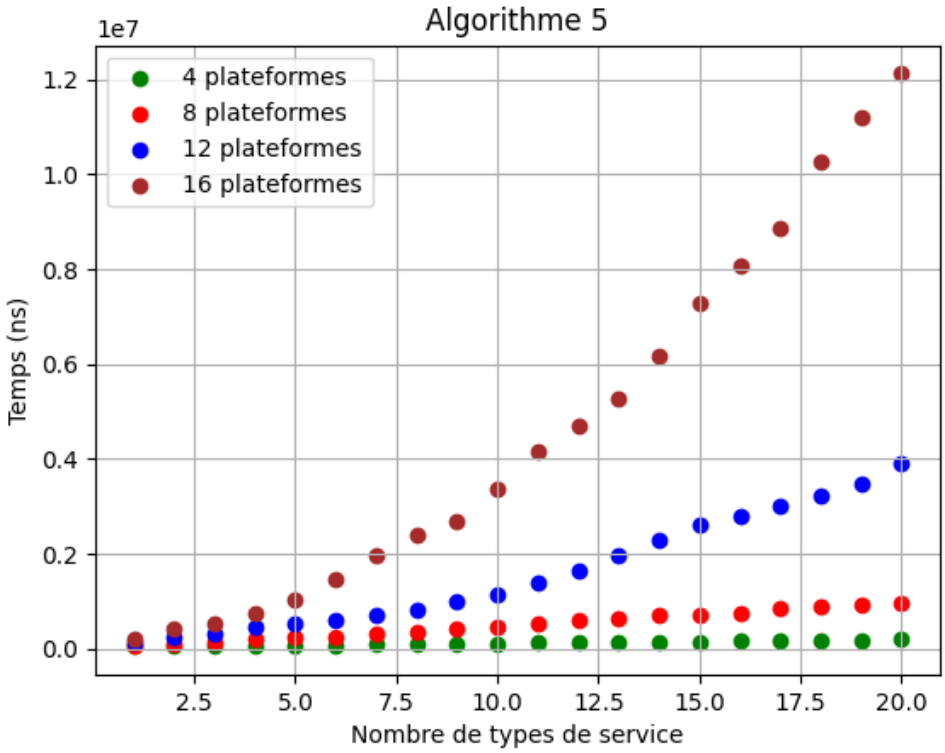

FIGURE 4.20 – Temps de calcul de l'algorithme 5 en fonction du nombre de plateformes et de types de services.

Algorithme *checkPlanStatus*. Les résultats de la figure 4.20 présentent les temps de calcul de la fonction qui vérifie le statut des plans proposés par l'agent tactique en fonction du nombre de types de services et du nombre de plateformes. Les temps de calculs sont en pratique plus courts que pour l'algorithme 4, car la boucle s'arrête dès qu'un plan accepté, annulé ou terminé est trouvé dans la liste des plans envoyés.

Conclusion

Cette première version a permis de démontrer le bon fonctionnement des agents pour l'allocation des tâches et d'observer la complexité des algorithmes pouvant impacter le temps de réaction des systèmes. Nos résultats sont synthéthisés dans le tableau 4.3. Dans notre cas, les deux derniers algorithmes sont exécutés sur chaque agent tactique. Le temps affiché n'est que pour un agent tactique et nous travaillons en séquentiel sur une même plateforme. En parallélisant les calculs des agents tactiques, nous pouvons donc théoriquement obtenir un temps similaire à celui présenté dans le tableau.

Algorithme	**Paramètres**	**Temps mesuré (en ms)**
checkIfResourceIsAvailable	200 plans proposés	3.5
updateServices	20 types de services et 48 ressources	9
resourceAllocation	100 agents tactiques et 10 plateformes (1 type de service et 30 ressources)	8.5
calculatePlan	1 agent tactique, 20 types de service et 48 ressources	17.5
checkPlanStatus	1 agent tactique, 20 types de service et 48 ressources	12

TABLE 4.3 – Temps moyens relevés pour chaque algorithme dans les pires cas en fonction de différents paramètres.

Ces résultats suggèrent que le passage à l'échelle de la solution n'est pas limité par le temps d'exécution des fonctions de décision des différents agents. Les limites de l'approche apparaissent pour des valeurs en dehors du cadre de nos scénarios d'usage.

Cependant, ce modèle ne prend pas en compte les aspects de latence entre les différents composants comme les agents s'exécutent les uns à la suite des autres. De plus, la collaboration multi-plateforme est limitée par les débits disponibles pour la communication entre les plateformes. La première version ne prend pas en compte la quantité de données qui doit être échangée pour le fonctionnement du système multi-agent, ce qui est limitant pour son intégration opérationnelle. La seconde version prend en compte ces aspects.

4.5.2 Deuxième version distribuée avec *eventBus*

La deuxième version du framework propose d'intégrer de la communication entre les agents du système en utilisant un *thread* pour exécuter chaque agent plateforme. Le fonctionnement des agents services et tactiques d'une même plateforme reste séquentiel, mais

les agents plateformes interagissent de manière asynchrone. Nous utilisons tout d'abord une bibliothèque *eventBus* permettant des communications *publish-subcribe* entre les plateformes, à la place de partager les informations via un *blackboard* qui n'est plus adapté en raison de la concurrence d'accès aux données des agents. La bibliothèque *eventBus* permet d'échanger des données entre les agents qui s'exécutent sur des *threads* différents, de manière instantanée et sans les latences que peut apporter RabbitMQ. Le framework reste similaire dans son ensemble, avec la partie IHM et les algorithmes utilisés par les agents.

Modifications par rapport à la première version

Le framework se place dans des conditions d'expérience similaires à la première version. L'utilisation de *threads* et l'intégration de RabbitMQ ou de l'*eventBus* dans le système apportent une modification majeure qui impacte plusieurs éléments. Tout d'abord, nous avons dû modifier la mémoire partagée. Dans la première version cette mémoire était accessible par tous les agents pour simplifier le partage de la donnée. Dans notre seconde version, nous créons de multiples mémoires partagées, une par plateforme. Tous les agents d'une même plateforme (un agent service, un agent plateforme et N agents tactiques) peuvent accéder à la donnée. Cependant, le système doit récupérer des données provenant d'autres plateformes et il doit également échanger pour prendre des décisions. Tout ces échanges vont alors passer par un intermédiaire, RabbitMQ ou l'*eventBus*, selon un paradigme *publish/subscribe*, ce qui permettra de mettre à jour les connaissances des agents. Les données qui seront échangées sont : les ressources, les positions, les agents tactiques pour la redondance, les propositions de plans, et les retours sur ces propositions. Nous pouvons observer ces échanges sur la figure 3.12, présentée au chapitre précédent, où le réseau maillé correspond à l'*eventBus* dans la première version et RabbitMQ dans la seconde.

Dans la première version, le déroulement d'un plan se faisait à chaque pas de simulation. Les différents *threads* étant indépendants temporellement, il n'est plus possible de se baser sur des pas de simulation, mais plutôt sur le temps du système, tel que mesuré par la fonction « System.currentTimeMillis() » en Java. Chaque service possède une durée d'exécution qui déterminera l'avancement d'un service ainsi que l'évolution de son score.

Le fonctionnement des agents reste séquentiel sur une même plateforme, c'est-à-dire que chaque agent exécutera ses algorithmes à tour de rôle. C'est une étape intermédiaire vers l'implémentation complète du SMA, où chaque agent d'une même plateforme pourra

s'exécuter en parallèle. Cette première approche permet de réduire le nombre de conflits et de modifications à apporter. La manière d'allouer les tâches localement entre l'agent service et les agents tactiques reste la même. L'allocation globale est plus complexe, les agents services font des demandes de ressources en parallèle et l'ordre d'arrivée des messages n'est plus assuré comme pour le modèle séquentiel. De plus, dans la première version, l'avancement des services était fait au niveau de l'agent service. A présent, chaque plateforme possédant la ressource met à jour l'avancement des services collaboratifs en cours, donc les ressources participants à un même service doivent synchroniser leurs actions. Nous considérons cette synchronisation imparfaite en raison des latences possibles dans le réseau, nous supposons que l'utilisation des différentes ressources pour l'exécution d'un service collaboratif à environ 1 seconde d'intervalle est suffisante pour que la collaboration soit effective.

Enfin, l'agent plateforme est le plus à même de connaître l'avancement des services sur ses ressources. C'est pour cette raison qu'il écoutera les demandes de services des agents services de toutes les plateformes. Il est donc nécessaire d'avoir un canal de communication ouvert en parallèle pour écouter la réception de messages à tout moment.

Complexité des interfaces de communication

Cette deuxième version nous permet d'observer en pratique la complexité des interfaces de communication utilisés dans le cadre de l'allocation de tâches distribuée. Cette complexité dépend de plusieurs paramètres tels que le nombre de plateformes, le nombre de pistes ou le nombre de services proposés. Pour toutes nos simulations, nous fixons 4 types de services, un pour chaque objectif opérationnel. Chaque plateforme possède le même nombre d'agents tactiques qui constitue un des paramètres de la simulation. Le programme s'arrête lorsque tous les agents ont terminés, c'est-à-dire qu'ils ont atteint leur dernier objectif opérationnel. Nous nous intéresserons à la quantité de messages et d'octets échangés lors de la simulation. On pourra alors observer ces métriques en fonction des paramètres que nous ferons varier.

Les fonctions *sendMessagePosition* et *sendMessageResource*. Pour ces deux interfaces, un message de position est envoyé par chaque plateforme toutes les cinq secondes et un message pour échanger les ressources également (avec une mise à jour plus rapide en cas de modification d'une ressource). Cela nous permet donc d'observer un débit en octet par seconde. Les résultats, présentés en figure 4.21, montrent bien une évolution

quadratique du débit en fonction du nombre de plateformes, comme le suggérait la théorie. La fréquence d'envoi des messages pourra être modifiée selon la situation, des plateformes aériennes ont besoin d'un taux de rafraichissement plus important que des plateformes navales. Nous pouvons également noter que le débit affiché est un débit global, et non la quantité d'information que chaque plateforme recevra.

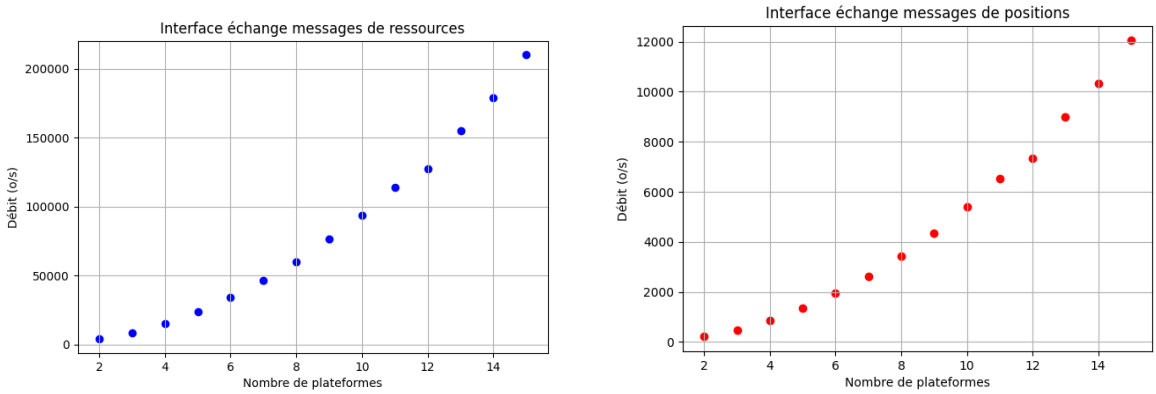

FIGURE 4.21 – Quantité de données échangée par les fonctions d'échange des ressources et des positions en fonction du nombre de plateformes.

Les interfaces de pré-allocation. Dans cette version du framework, les messages sont échangés instantanément (sans latence apparente), ce qui rend difficile l'établissement d'un débit réaliste pour chaque interface d'échange. À la place, nous relevons le nombre d'octets et de messages échangés au cours de la simulation pour en observer l'évolution.

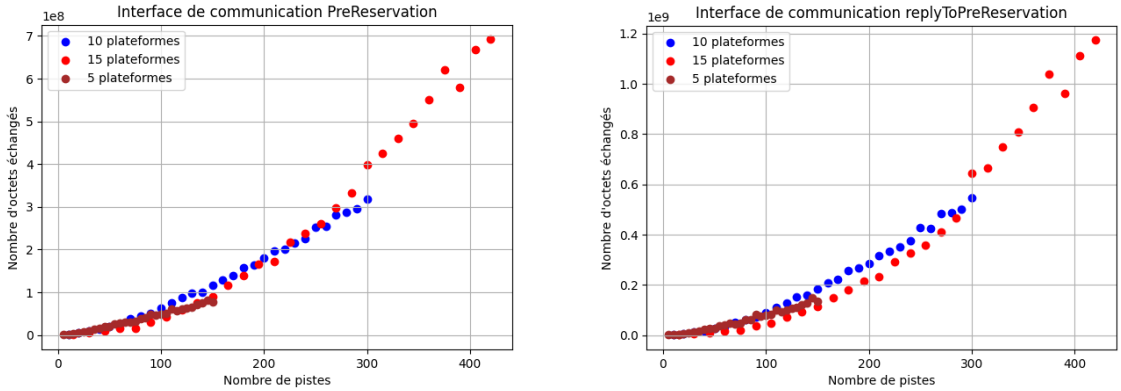

FIGURE 4.22 – Quantité de données échangée par les fonctions de pré-réservation des ressources en fonction du nombre de pistes et du nombre de plateformes.

Sur la figure 4.22, nous pouvons observer que le nombre d'octets augmente avec le nombre de pistes et que le nombre de plateformes a peu d'impact sur cette métrique. L'agent service fait une pré-sélection des plans avant de les envoyer et choisit au maximum un plan par agent tactique, ce qui réduit fortement la quantité d'information échangée par rapport à une architecture où chaque agent tactique enverrait toutes les propositions de plans à un agent service en passant par le réseau.

Les fonctions d'allocation réelle et de mise à jours des services en cours. Dans notre programme, nous proposons de fixer une durée de temporisation, positionnée à 100 ms dans notre simulation, avant d'envoyer les plans sélectionnés par l'agent service. En modulant cette valeur, l'agent service peut recevoir plus de plans avant de les envoyer aux agents plateformes, ce qui permet de les regrouper en un seul message, réduisant ainsi le nombre de messages à échanger. Cependant, du côté de l'agent plateforme, nous proposons de répondre directement à la réception d'une demande d'allocation, ce qui explique que le nombre de messages est plus important lors de la réponse, comme nous pouvons le voir sur la figure 4.23. Les réponses pourraient également être regroupées selon leurs destinataires avant envoi, dans ce cas on se retrouverait avec autant de messages des deux côtés. Le nombre d'octets pourra cependant différer, car certaines propriétés des plans ont pu être modifiées (par exemple, le statut du plan).

Chaque message de l'allocation réelle ne contient qu'un seul plan, qui est envoyé aux plateformes possédant les ressources nécessaires. Le nombre de messages reste cependant nettement inférieur par rapport à la phase de pré-allocation, car celle ci n'a lieu que lorsque les deux plateformes répondent positivement pour l'utilisation de leurs ressources, et une bonne partie des plans proposés peuvent être refusés. L'agent plateforme retournera toujours un message, lorsque le service est terminé ou annulé, nous observons donc le même nombre de messages pour la mise à jour de l'avancement des services. Nous pourrons également noter qu'il y a plus de messages lorsque le nombre de cibles augmente, étant donné qu'il y a plus d'échanges à effectuer pour que chaque agent puisse accomplir ses objectifs.

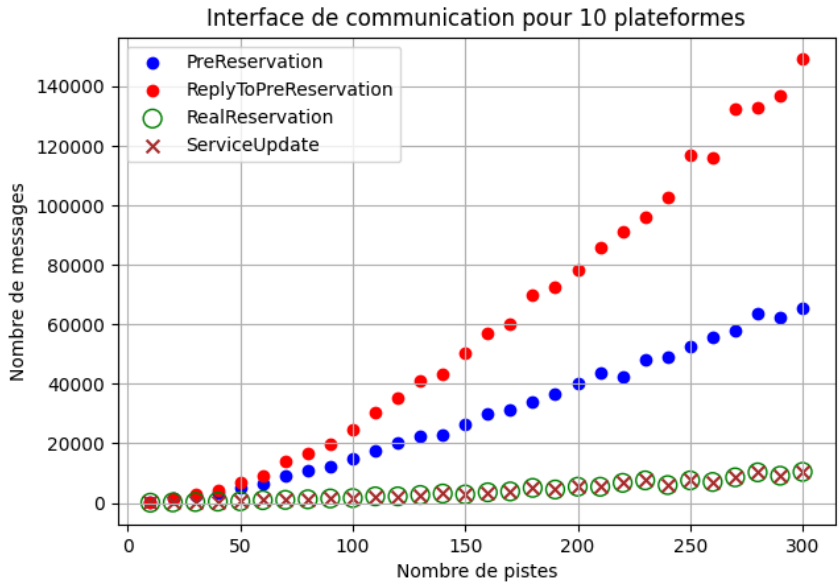

FIGURE 4.23 – Nombre de messages échangés par les différentes interfaces en fonction du nombre de pistes.

Conclusion sur la version distribuée avec *eventBus*

Nous avons pu observer l'évolution de la quantité de données échangés jusqu'à ce que tous les agents tactiques aient atteint leur objectif final en fonction de différents paramètres : le nombre de plateformes, le nombre d'agents tactiques, le nombre de ressources et le nombre de services. Nous avons ainsi pu comparer avec les complexités que nous avions présentées dans le chapitre précédent. Nous aurions également voulu présenter le déroulement complet d'une étape de l'algorithme d'enchère. Cependant, l'aspect asynchrone de nos agents ne nous permet pas de délimiter clairement une séquence d'échanges de messages pour obtenir une première allocation de ressources.

Néanmoins, cette simulation présente une limite, elle ne permet pas de préciser la quantité d'information qui est échangée au cours du temps, étant donné que les échanges sont instantanés et sans latence. La métrique de bande passante est importante pour déterminer si le système pourra fonctionner dans un contexte opérationnel. L'intégration de RabbitMQ à la place de l'*eventBus* permet d'obtenir des résultats plus précis sur les latences et les débits en fonction de l'évolution des paramètres. La version avec *eventBus* demeure plus performante et permet d'aller plus loin dans le nombre de paramètres et de *threads*, ce qui explique pourquoi nous l'avons utilisé pour ces tests d'interfaces plutôt que RabbitMQ.

4.5.3 Deuxième version distribuée avec *RabbitMQ*

Nous proposons à présent d'intégrer le middleware RabbitMQ pour gérer la communication entre les agents du système. L'utilisation de Grafana et de Prometheus permettent d'observer les différentes métriques du système au cours de la simulation.

Interface graphique Grafana

Dans la figure 4.24, nous affichons tout d'abord des informations générales concernant l'ensemble des plateformes et RabbitMQ, puis des informations plus spécifiques à chaque plateforme :

- Le nombre de plateformes.
- Le nombre de canaux.
- Le nombre de files d'attente.
- Le nombre d'agents tactiques en général et pour chaque plateforme.
- Le débit entrant et sortant du serveur RabbitMQ.
- Le débit entrant et sortant de chaque plateforme.
- Le nombre de messages sortant par minute.

Nous utilisons le nom de chaque agent plateforme comme variable pour obtenir des informations précises sur les interfaces de communication de chacune d'entre elles, ce qui n'était pas possible avec l'*eventBus*.

Sur la figure 4.24, nous avons également des onglets spécifiques à chaque interface de communication. Si nous ouvrons un onglet, nous pouvons voir sur la figure 4.25 des informations ciblées uniquement sur une des interfaces. La ligne supérieure concerne la plateforme désignée par la variable « query0 », la ligne inférieure concerne toutes les plateformes. Nous affichons ainsi des métriques sur la latence moyenne de l'interface, les débits d'envoi et de réception, ainsi que le nombre de messages échangés par minute. Dans le cas de la fonction de mise à jour des services, nous affichons également le nombre de messages annulés et le nombre de messages terminés. Il est donc possible de définir des métriques d'intérêt propres à chaque interface si nécessaire.

Observation des métriques selon plusieurs paramètres

Dans le cadre de l'exemple de dashboard vu précédemment, nous avons enchainé des simulations de cinq minutes chacune, en faisant varier le nombre de plateformes et de cibles. Nous avons modifié le comportement des agents tactiques pour qu'ils continuent

FIGURE 4.24 – Vue d'ensemble du dashboard sur Grafana.

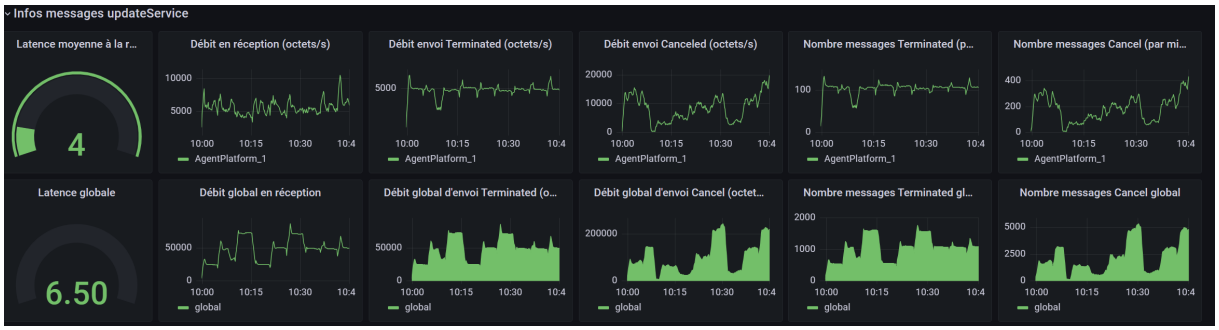

FIGURE 4.25 – Onglet sur les interfaces de mise à jour des services.

de planifier pendant toute la durée de la simulation, c'est une manière artificielle de tester la charge.

La figure 4.26 présente le débit global en réception par les plateformes dans le cadre de l'échange de données de position. Nous avons délimité chaque simulation par des lignes en pointillés bleues. Nous observons que le nombre d'agents tactiques n'a pas d'impact sur ce débit, comme nous l'avons expliqué en théorie, seul le nombre de plateformes a un impact ici. Nous pouvons aussi noter l'augmentation quadratique du débit global en

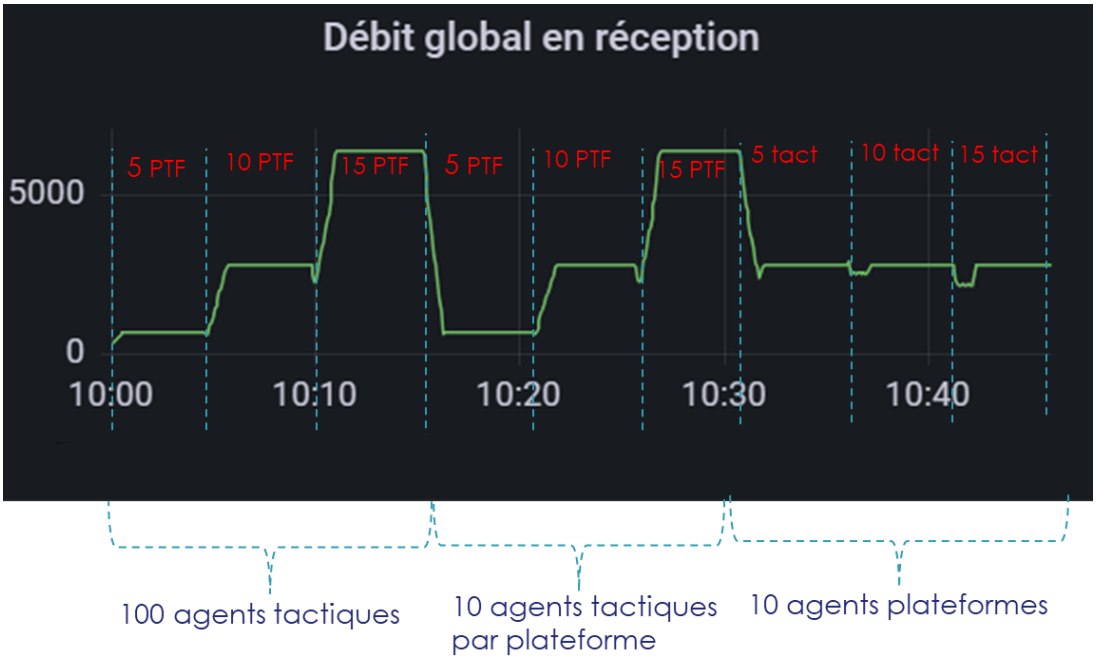

FIGURE 4.26 – Débit global en réception lors de simulations où le nombre de plateformes et de cibles varie.

fonction du nombre de plateformes.

La figure 4.27 présent le nombre de messages qui sort de l'ensemble des plateformes à chaque minute. Pour un même nombre d'agents plateformes, le nombre de messages sortants est plus élevé. Cette augmentation peut être expliquée par le fait qu'il y a plus de pistes disponibles pour le même nombre de ressources. Par conséquent, l'agent service doit échanger davantage de messages pour répondre aux demandes des agents tactiques. De plus, le nombre de messages de refus pour les services proposés augmente également. À mesure que le nombre d'agents satisfaits augmente, le nombre de messages diminue, car ces agents ne planifient rien tant que leur plan en cours n'est pas terminé. Cela allège également la taille des messages échangés par l'agent service.

Le nombre de ressources disponibles impacte la satisfaction des agents tactiques, ce qui réduit le besoin de négociation sur l'utilisation des ressources, et qui peut donc également réduire le nombre de messages. Cela permet d'expliquer les valeurs observées pour 100 agents tactiques, seulement deux ressources sont disponibles sur chaque plateforme pour accomplir un service. Pour 5 plateformes, la probabilité de satisfaire les agents est faible, mais le nombre de messages à échanger est également moins important. Avec 15

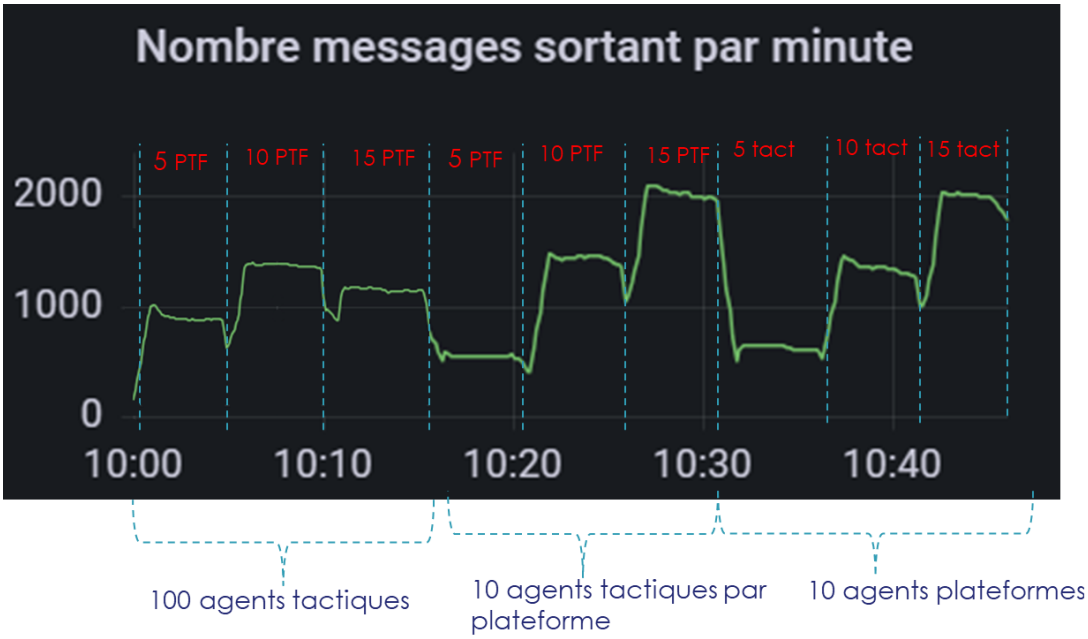

FIGURE 4.27 – Nombre de messages sortant par minute lors de simulations où le nombre de plateformes et de cibles varie.

plateformes, le nombre de messages devrait augmenter, cependant, les agents tactiques sont plus rapidement satisfaits, et lorsqu'ils ont des plans en cours ils ne planifient plus et donc moins de messages sont échangés.

Conclusion

L'intégration de RabbitMQ et d'outils de monitoring nous permet d'observer le fonctionnement de notre système au cours de simulations. Notre framework devient alors un outil adapté pour cibler les points névralgiques de l'architecture et optimiser l'utilisation de la bande passante. Il est également modulable, de nouvelles métriques peuvent être exportées et d'autres paramètres peuvent être utilisés.

4.5.4 Réponse aux besoins opérationnels et industriels

Dans notre premier chapitre, nous avions précisé que l'architecture devait répondre à certaines caractéristiques nécessaires d'un point de vue opérationnel. Le tableau 4.4 résume la manière dont nous répondons à chaque besoin.

Besoins	**Réponses**
Commande par ordre haut niveau	Les requêtes opérationnelles et la notion de services permettent d'abstraire le fonctionnement du système pour l'utilisateur. Celui-ci n'a pas besoin de connaître le fonctionnement des senseurs, ni leurs caractéristiques pour choisir le meilleur en fonction de la situation.
Rapidité de prises de décisions	Nous avons relevé les temps de calcul des algorithmes et les latences sur les interfaces qui sont des éléments pouvant retarder la prise de décision. Nous avons pu observer des valeurs satisfaisantes pour des scénarios de charge. Le temps de prise de décision augmente à mesure que le scénario se densifie. Les agents doivent intégrer des mécanismes pour gérer la congestion et prioriser certaines communications.
Utilisation de l'IA pour apporter de l'aide à la décision	Le SMA propose des plans senseurs qui pourront être validés par un opérateur sans qu'il n'ai besoin de connaître parfaitement l'utilisation des senseurs.
Décentralisation de la conception	Nous avons proposé de distribuer les agents sur les différentes plateformes pour faciliter la mise à l'échelle, pour créer de la redondance et pour décentraliser la conception.
Robustesse	Le système fonctionne en mode dégradé : lorsque des plateformes quittent le réseau ou que des senseurs arrêtent de fonctionner, le système multi-agent arrête les services impactés et replanifie si nécessaire.
Priorité de l'utilisateur	Nous avons vu que l'intégration d'un agent tactique particulier avec une priorité maximale impose au système d'utiliser le service proposé par l'agent, ce qui prouve qu'un utilisateur peut imposer l'utilisation de services s'il le souhaite.

Modularité et adaptabilité	L'abstraction des concepts de services, de ressources, des plateformes et les hypothèses de conception permettent à l'architecture d'être modulaire. En respectant les normalisations proposées, il est ainsi facile d'ajouter de nouveaux services par exemple, et de nouvelles ressources.
Capacité « plug-and-play »	Nous avons vu que le système pouvait s'adapter en cas d'ajout d'une nouvelle plateforme ou d'une ressource dans le réseau. Ces éléments peuvent s'intégrer dans le réseau, tout en préservant le fonctionnement de celui-ci.
Quantification des besoins en bande passante	Nous avons obtenu des résultats sur les besoins en bande passante pour les échanges entre les entités de notre système. Nous avons également proposé un monitoring qui permet d'observer le comportement du système et des agents en temps réel d'un point de vu du réseau.
Mise à l'échelle	Nous avons montré qu'il était possible d'intégrer de nouvelles plateformes dans le système et de nouvelles pistes dans le cadre d'un scénario dense. La performance du système dépend des ressources disponibles et du dimensionnement du réseau (entre autres).

TABLE 4.4 – Résumé de la manière dont le système répond aux besoins spécifiés.

4.6 Conclusion et développements futurs

Dans ce chapitre, nous avons réalisé des expérimentations à base de simulations dans le but d'obtenir des résultats sur l'architecture multi-agent proposée. Nous avons présenté les travaux amonts d'interfaçage d'un simulateur avec un logiciel propriétaire. Ces premiers travaux nous ont amené à concevoir un banc d'essai puis un framework multi-agent afin de démontrer le bon fonctionnement de nos concepts présentés dans le chapitre 3. Ce framework a subit de nombreuses évolutions, en passant par 3 versions majeures, chacune apportant des résultats spécifiques. Ainsi, nous avons pu observer l'impact de

divers paramètres sur l'exécution des algorithmes des agents. Nous avons ensuite relevé des informations sur la quantité de données échangées par les plateformes pour allouer les tâches aux senseurs. Enfin, nous avons intégré ces résultats sous forme de graphes dans un outil de monitoring, afin de voir la quantité de données échangée au cours de la simulation.

Dans cette partie expérimentale, notre framework s'est avéré être un atout majeur dans l'évaluation et la validation de notre architecture multi-agent. Grâce à sa flexibilité et à sa capacité à reproduire des scénarios variés, nous avons pu analyser les performances du système développé. L'utilisation conjointe de Grafana et de Prometheus nous a permis d'observer en temps réel le fonctionnement du réseau ainsi que les échanges entre les différents agents, offrant ainsi une vue globale et détaillée du système. De plus, notre framework a permis de tester les concepts des agents, avec leurs algorithmes, leurs interfaces de communication et leurs mécanismes. En résumé, le framework constitue un outil précieux, permettant non seulement de tester et d'optimiser notre architecture, mais également de valider son fonctionnement.

Le framework nous a permis d'obtenir des résultats, qui ont montré que l'architecture proposée fonctionne dans le cas d'un scénario de charge. Nous avons observé que le débit nécessaire au global pouvait devenir assez important, de l'ordre de quelques mégaoctets par seconde. La latence évolue également à mesure que le nombre de plateformes ou de pistes augmente. Il conviendra donc de mesurer le débit maximum sur un scénario opérationnel complet, pour obtenir plus de précisions sur les besoins réels. L'environnement de simulation comprend des limites, mais permet d'améliorer les interfaces et les algorithmes dans le but de garder un mécanisme d'allocation efficace. Pour obtenir des résultats en phase avec la réalité, le framework doit évoluer, en incluant des services spécifiques, en posant des règles sur les interfaces pour optimiser l'utilisation de la bande passante, etc.

Comme mentionné précédemment, le framework a de nombreuses possibilités d'évolution. D'un côté, nous pouvons améliorer le fonctionnement de nos agents, en optimisant les algorithmes ou en précisant les règles des interfaces de communication pour réduire l'utilisation de la bande passante et la latence. Nous pouvons également développer un mécanisme d'élection pour déterminer quelles plateformes exécutent les agents. Le partage des agents tactiques et son coût en bande passante n'ont pas été traités dans ces expérimentations. D'autre part, nous pouvons également intervenir sur le monitoring du système, en exportant de nouvelles métriques, comme l'utilité globale, le taux d'occupation de la ressource, les charges de calcul de chaque algorithme, le taux de satisfaction

des agents, etc. Ces nouvelles métriques permettraient d'observer le comportement des agents et de voir lesquels sont moins sollicités et pour quelles raisons.

Enfin, nous avions prévu d'intégrer le framework dans le simulateur opérationnel que nous avons présenté, afin de proposer un scénario en temps réel plus proche d'un contexte opérationnel pour observer le bon fonctionnement du système multi-agent. Cette évolution nécessite un travail sur l'utilisation de véritables ressources pour accomplir des services spécifiques. Les temps d'exécution des services seraient différents, tout comme le changement de l'état opérationnel des agents tactiques (DRIL-P). Les informations des senseurs seront également plus précises, complétant ainsi les informations des agents tactiques et leurs objectifs.

Conclusion générale et perspectives

Retour sur la problématique

L'élaboration d'une architecture multi-agent distribuée et collaborative pour les systèmes navals constitue un défi majeur dans le contexte actuel de l'évolution des technologies et des exigences opérationnelles. Dans cette perspective, cette thèse s'est penchée sur les problématiques d'allocation de tâches au sein des systèmes distribués, confrontés à des contraintes de communication dans un environnement militaire souvent limité et instable. Un accent particulier a été accordé sur la nécessité des collaborations multi-plateformes, multi-milieux et multi-senseurs, où les senseurs et les plateformes sont interconnectés au sein d'un même réseau distribué. Par ailleurs, cette architecture de collaboration devait être capable de s'adapter à des caractéristiques essentielles des systèmes navals telles que la mise à l'échelle et la robustesse. Le premier chapitre de cette thèse offre un aperçu complet des enjeux et des défis liés à cette problématique.

Nous avons proposé d'utiliser les systèmes multi-agents afin de résoudre le défi complexe et distribué de l'allocation de tâches à des senseurs au sein d'un modèle collaboratif multi-plateforme, en réponse à notre première question de recherche *QR1*. Ensuite, nous avons étudié les stratégies de communication entre les plateformes militaires, car la coordination entre des agents d'un système distribué repose sur un échange d'informations dépendant de l'état du réseau et des priorités en fonction de la criticité des données échangées. L'analyse des interfaces de communication utilisés par les agents, ainsi que leurs complexités a permis de répondre à notre deuxième question de recherche *QR2*. Enfin, nous avons présenté plusieurs scénarios opérationnels pour valider le système multi-agent face aux besoins opérationnels et aux mécanismes exigés par le système. Ces scénarios, en observant différentes métriques aux cours de simulations, permettent de répondre à la troisième question de recherche *QR3*.

Travaux réalisés

QR1 : Une approche multi-agent pour l'allocation de tâches à des senseurs

Pour répondre à la première question de recherche, nous nous sommes intéressés aux systèmes multi-agents et aux méthodes d'allocation de tâches qu'ils utilisent. Parmi ces méthodes, les algorithmes d'enchères se sont démarqués par leur efficacité, bien qu'ils ne soient pas toujours optimaux. Ces algorithmes offrent une approche décentralisée et flexible pour l'allocation des tâches, permettant aux agents autonomes de négocier et de prendre des décisions localement, tout en contribuant au fonctionnement global du système.

Dans ce contexte, nous avons présenté des réflexions sur les choix de conception de l'architecture et sur l'attribution de rôles à des agents, détaillant les responsabilités et les fonctions qui leurs sont attribuées. Chaque agent possède une architecture commune lui permettant de communiquer et de prendre des décisions grâce à des algorithmes internes et aux informations qu'il accumule dans sa mémoire. En se basant sur les rôles et les besoins spécifiés, nous avons proposé un système multi-agent composé de trois types d'agents : tactique, service et plateforme. L'agent plateforme maintient une base de connaissance sur une plateforme de la flotte, il a connaissance des ressources et des positions de toutes les plateformes qu'il a pu obtenir par échange de messages. C'est également lui qui répond aux demandes d'allocation lorsqu'il s'agit de ses propres ressources. Ainsi, les plateformes peuvent coopérer et partager leurs situations tactiques. Ensuite, l'agent service permet de générer le catalogue des services de la plateforme en se basant sur l'ensemble des senseurs à sa disposition et les types de services que ces senseurs peuvent accomplir. Son rôle est plus large, grâce à un mécanisme d'enchère, il propose ses services à des agents tactiques et décide des gagnants. Les ressources des plateformes sont nécessaires à l'exécution de ces services. Pour chaque cible détectée sur le théâtre des opérations, un agent tactique est généré. Cet agent cherche à compléter sa base de connaissance sur la cible qu'il poursuit en utilisant les services qui lui sont proposés. Ainsi, chaque agent tactique proposera des offres sur les services qui dépendront de paramètres relatifs à la cible, aux objectifs qu'il poursuit et aux conditions de son environnement. L'agent service sera en charge de sélectionner les meilleurs services en faisant attention à leurs priorités respectives. Il devra ensuite négocier avec les autres agents services qui sont en concurrence sur l'obtention des ressources. Ce système multi-agent possède une capacité d'adaptation forte qui lui

permet de répondre dynamiquement aux variations de l'environnement, que ce soit par une augmentation du nombre de menaces, des modifications sur les ressources ou bien l'insertion de nouvelles plateformes dans la coalition.

La complexité de la solution réside principalement dans l'aspect distribué de l'environnement. Logiquement, les plateformes communiquent par le biais de liaisons de communication, les agents sont donc soumis aux mêmes contraintes liés à la latence. Les agents plateformes cherchent un état global cohérent avec les autres plateformes afin d'éviter des problèmes d'interblocages ou de famines. Ainsi, les communications entre agents revêtent une importance capitale pour le bon fonctionnement du système. Elles constituent le socle sur lequel repose la coordination des tâches affectées aux ressources senseurs, tout en permettant d'assurer la cohérence d'un état global du système. Dans la section suivante, nous aborderons les réponses à la question de recherche *QR2* portant sur les interactions entre les agents, en mettant en lumière les défis liés aux problématiques de latences, de priorités des messages et d'optimisation des échanges, dans le cadre de notre environnement distribué.

QR2 : Interfaces d'échanges multi-agents pour la collaboration multi-plateforme

La seconde question de recherche considère les problématiques des communications entre les agents. Les agents services doivent échanger des informations avec les agents plateformes pour convenir de l'utilisation des ressources pour les différents services nécessités. Dans ce cadre, nous avons proposé d'utiliser une double enchère pour allouer des tâches aux ressources possédées par les agents plateformes. La première enchère s'effectue entre les agents tactiques et l'agent service de chaque plateforme. Les propositions par séquence des agents tactiques permettent une cohérence sur le choix des vainqueurs par l'agent service, une mémoire partagée permet d'échanger des données entre les agents locaux. À la suite de cette enchère, l'agent service reçoit une liste de plans senseurs qu'il communiquera aux agents plateformes concernées par les ressources recherchées lors d'une deuxième enchère. Cette fois, il est en concurrence avec d'autres agents services qui proposeront des plans dont l'agent plateforme décidera de leur acceptation ou de leur refus.

L'aspect critique du système et le besoin de prendre en compte des priorités sur les tâches à mener nécessitent que le système puisse préempter des ressources en cours d'utilisation par des plans moins prioritaires. Ces échanges constants qui évoluent à mesure que

le nombre d'agents augmente soulève un vrai défi pour le système. Nous avons présenté les différentes interfaces de communication utilisées dans le cadre de cette allocation de tâches en précisant leur complexité en terme de messages et de bits échangés.

QR3 : Validation de l'architecture par scénarios opérationnels

La dernière question de recherche *QR3* met en pratique les concepts multi-agents et les problématiques de communications présentés dans cette thèse. Nous avons présenté différents scénarios opérationnels permettant de stimuler l'architecture selon différentes configurations et événements pouvant survenir dans un environnement militaire.

Dans un premier temps, nous avons proposer d'interfacer un système multi-agent existant avec un simulateur opérationnel. La complexité du système ne permettait pas de faire évoluer l'architecture de manière distribuée, nous avons donc changé d'approche en proposant de monter en abstraction. Ainsi, nous avons proposé dans un premier temps un *framework* avec RabbitMQ et des outils de *monitoring* pour faire face à la complexité des systèmes de senseurs navals [QKLB+23]. Nous l'avons ensuite développé pour intégrer les concepts d'agents que nous proposons dans ce manuscrit.

Ce nouveau framework à base d'agent a permis de démontrer l'application de ces concepts et d'obtenir des résultats sur la complexité des algorithmes utilisés par les agents en fonction de l'évolution du contexte en terme de nombre de menaces, de plateformes alliés, de ressources et de services associés. Nous avons également proposé de mettre en évidence la complexité des communications et la nécessité de surveiller et de contrôler le système en optimisant l'utilisation de la bande passante dans des scénarios denses.

Perspectives

Les travaux présentés dans ce manuscrit offrent de nombreuses ouvertures potentielles et de nouvelles problématiques non étudiées à l'heure actuelle. De futurs travaux pourraient viser à approfondir les solutions développées dans le cadre de cette thèse, en intégrant la solution dans des systèmes réels qui nécessitera une levée des abstractions proposées.

Dans un premier temps, l'intégration de la solution dans un environnement plus complet pourrait consister à lever l'abstraction sur les notions de services et de ressources considérés. Chaque ressource et chaque service pourraient avoir des spécificités,

des contraintes et des utilisations différentes nécessitant un grain plus fin pour permettre aux agents de les sélectionner et de les utiliser avec une précision plus importante. Cela permettrait de refléter fidèlement les contraintes et les défis rencontrés d'un environnement opérationnel réel. Les travaux d'interfaçage du système multi-agent avec un simulateur sont des points d'entrée pour l'évolution du système futur dans le cadre de la simulation opérationnelle. Après des simulations plus dimensionnantes, un maquettage sur architecture physique pourra être envisagé.

Une comparaison approfondie de la solution développée vis-à-vis d'autres architectures existantes pourrait également constituer un axe de recherche pertinent, permettant d'évaluer les avantages et les limitations de chaque approche, ainsi que les opportunités d'optimisation algorithmique et de modification des interfaces de communication pour améliorer les performances du système. Cette comparaison semble toutefois complexe dans les cadres fermés du combat collaboratif et des opérations militaires, où des aspects tels que la reconfiguration, la tolérance aux pannes, la perte de connectivité et l'incertitude jouent un rôle crucial. Néanmoins, une utilisation de l'architecture dans des domaines possédant des similitudes pourra être envisagée.

En outre, l'intégration de mécanismes d'ordonnancement et d'ontologies contribuerait à enrichir le système, en améliorant les connaissances des agents et en renforçant la pertinence des actions entreprises par les senseurs au cours du temps. Cette démarche permettrait notamment d'optimiser les propositions d'allocation, en tenant compte du caractère dynamique du contexte opérationnel et de la nécessité de solutions rapides et efficaces.

En parallèle, une attention particulière devrait être accordée à l'aspect cybersécurité des échanges au sein du système multi-agent, en vue de garantir l'intégrité, la confidentialité et la disponibilité des informations sensibles échangées entre les entités distribuées.

Enfin, il serait également intéressant d'examiner les aspects de confiance de l'intelligence artificielle distribuée du système multi-agent, en particulier dans son rôle en tant que système d'aide à la décision. Une analyse approfondie des mécanismes de confiance et de validation des décisions prises par le système pourrait contribuer à renforcer la fiabilité et la crédibilité de ses actions par des acteurs humains.

Annexe A

Plateformes navales et aériennes

Frégate de défense aérienne FDA

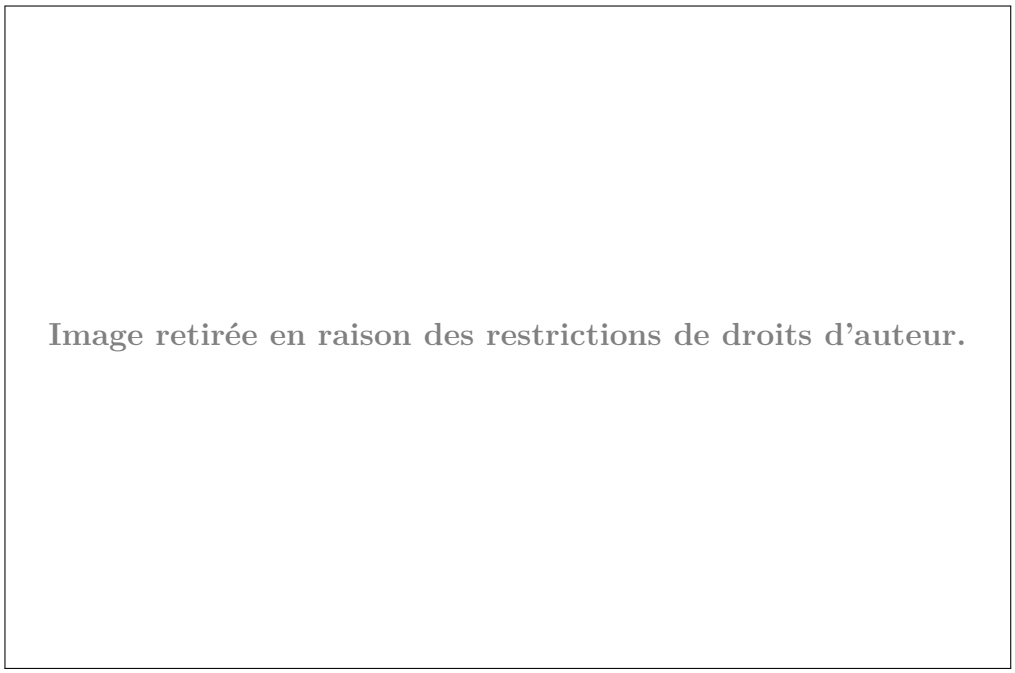

Figure A.1 – Frégate de défense aérienne.

La frégate de défense aérienne (FDA, figure A.1) est responsable de la lutte anti-aérienne ainsi que de la gestion des opérations aériennes. Elle maintient une situation tactique dans les airs et coordonne les tirs alliés.

Une FDA utilise deux moteurs de propulsion diesel, deux turbines à gaz de propulsion et un propulseur d'entrave pour atteindre une vitesse maximale de 30 noeuds. Sa défense antiaérienne est composée du système d'arme antiaérien PAAMS qui emploie des missiles Aster. Les missiles Aster 15 sont capables d'intercepter à 360° des avions, des drones et des missiles de croisière jusqu'à plus de 30 km de distance et une altitude de 13 km. Les missiles Aster 30 augmentent leur portée à 120 km pour 20 km d'altitude, ce sont des

missiles plus adaptés à la défense d'une zone et donc des plateformes alliées. Il possède également des missiles antinavires, des tourelles, des canons et des mitrailleuses. Enfin, six radars sont embarqués à bord, dont un dédié à la conduite de tir et un autre à la veille-air 3D. La Marine nationale possède deux FDA en service, le Chevalier Paul et le Forbin.

Frégate défense et d'intervention FDI

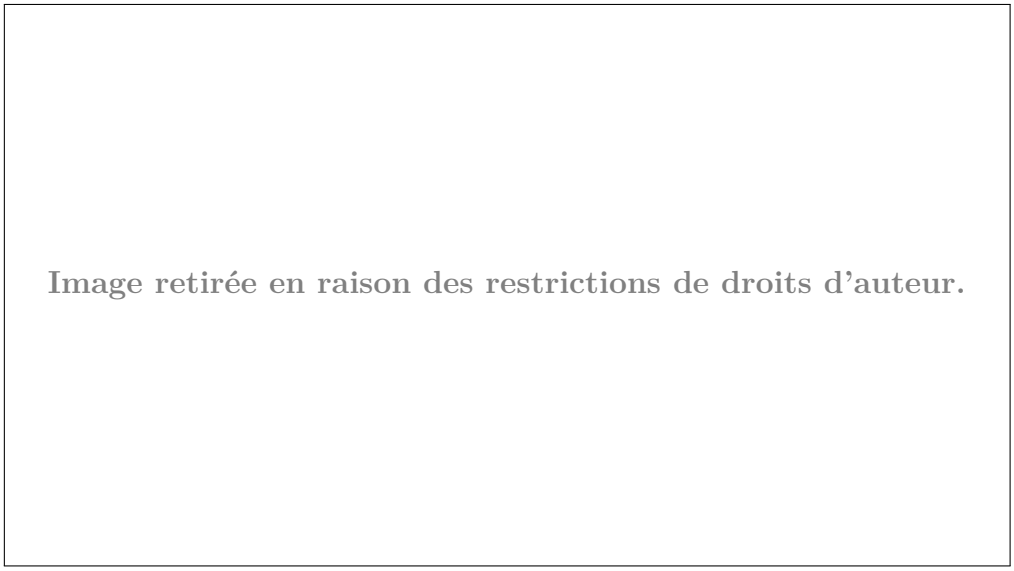

FIGURE A.2 – Frégate de défense et d'intervention Amiral Ronarc'h en visualisation 3D (premier plan).

La fréqate de défense et d'intervention (FDI, figure A.2), en cours de construction, sera livré en plusieurs exemplaires à la Marine nationale d'ici 2030. La vitesse maximale de la FDI pourrait dépasser les 27 noeuds. La frégate sera capable d'agir seule ou en coopération au sein d'une force navale pendant 45 jours. Sa polyvalence lui permet de gérer tous les domaines de luttes (aérienne, surface, sous-marine). Du point de vue des innovations, elle embarquera un hélicoptère et un drone. De plus, ce sera le premier navire a posséder un mât unique contenant tous les senseurs responsables de la veille aérienne, des innovations de cyberdéfense seront également intégrées dans le système.

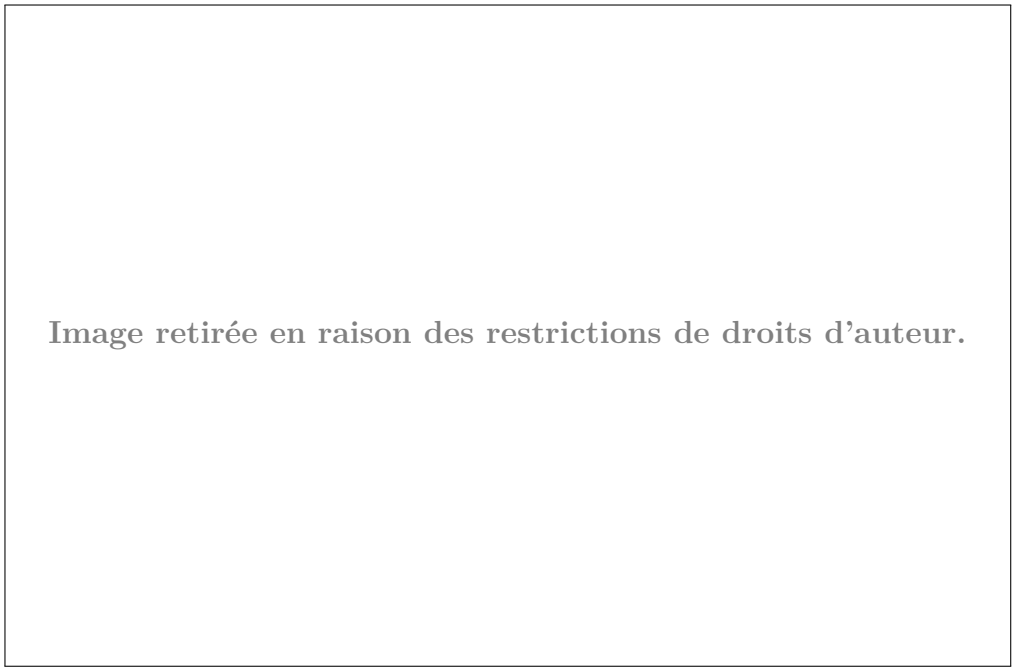

FIGURE A.3 – Frégate multi-missions Bretagne.

Frégate multi-missions FREMM

La frégate multi-missions (figure A.3) est un navire polyvalent qui permet les opérations de lutte aérienne, de surface, sous-marine et également terrestre lorsqu'elle est proche des côtes. La Marine nationale possède en tout six FREMM.

Le missile de croisière naval (MdCN) est une arme précise pouvant être lancée à 1000km de distance qui permet à la FREMM des frappes rapides sur des cibles terrestres stratégiques. En outre, le navire détient également des missiles antinavires et antiaériens (Aster 15), ainsi qu'une tourelle, deux canons et deux mitrailleuses. Pour la partie senseurs, la FREMM possède deux radars de navigation, un radar de veille multifonctions, un sonar de coque, un sonar remorqué et un brouilleur. Tout comme la FDI, le navire peut embarquer un Caïman Marine et un drone.

L'E-2C Hawkeye

L'E-2C Hawkeye est un aéronef embarqué sur le porte-avions Charles de Gaulle, il permet une détection avancée et un commandement sous contrôle. D'une part, la détection est effective grâce au radar AN/APS 145 capable de détecter et de poursuivre jusqu'à 2000 cibles aériennes simultanément à plus de 500km de distance, couplé au système de

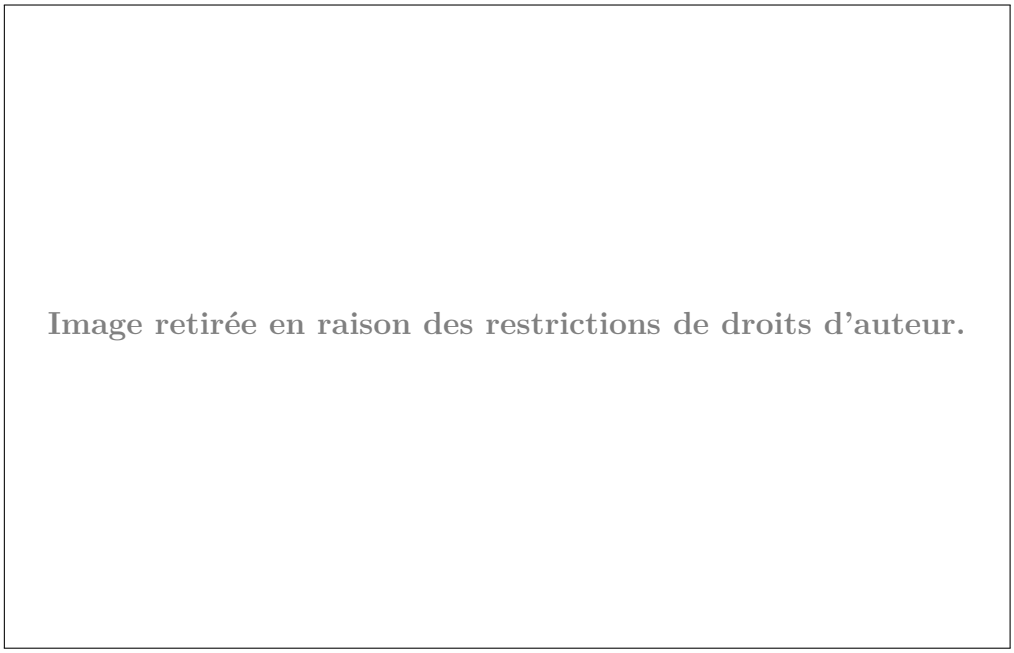

FIGURE A.4 – E-2C Hawkeye.

détection passif ALR-73 qui complète les informations pour obtenir plus de précisions sur les radars ennemis et leurs porteurs. L'Hawkeye détecte et piste toutes les cibles aériennes et maritimes grâce à son radôme. D'autre part, le commandement est possible grâce aux moyens de communication, les liaisons de données tactiques LDT L11 et L16, ainsi que la liaison satellite FLEET-SATCOM.

Annexe B

Exemples de senseurs supplémentaires

Le radar naval multifonction Herakles

Le radar Herakles (figure B.1) possède une antenne tournante (1 tour/s) positionné en haut d'un mât secondaire [BCC+14]. Il permet d'établir la veille aérienne et de surface, et peut également détecter, poursuivre et identifier les cibles. Il peut également guider les missiles Aster et évaluer le succès du tir. Il possède donc plusieurs modes de fonctionnement selon la situation. Il fonctionne en bande S, ce qui lui donne une portée de détection de 250 km pour les cibles aériennes et 80 km pour les cibles de surface. Le radar

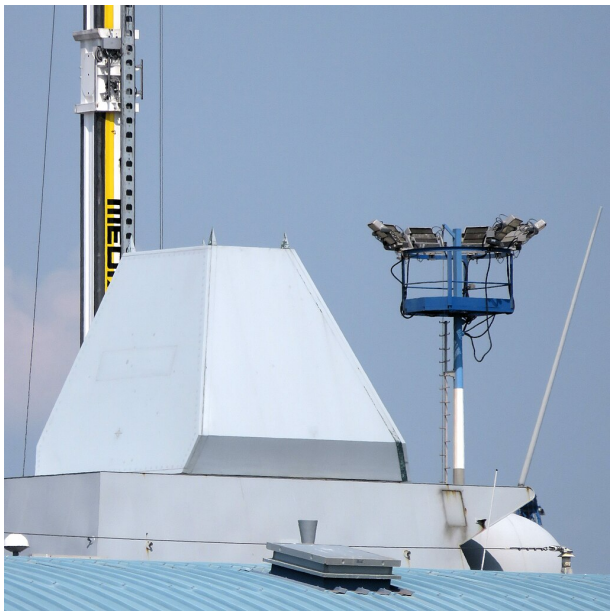

Figure B.1 – Radar Herakles.

multifonction remplit plusieurs tâches, il permet de remplacer plusieurs radars en un seul

plus polyvalent. Ainsi, la FREMM ne possède qu'un seul radar Herakles capable de faire de la veille et la poursuite de cibles. Le radar possède des mécanismes de redondance au niveau des émetteurs et des récepteurs, une panne sur un des modules n'altère pas le fonctionnement du radar.

Les radars à synthèse d'ouverture

Les radars à synthèse d'ouverture, ou SAR (*Synthetic Aperture Radar*), sont des dispositifs électroniques permettant de retranscrire sous la forme d'une image la réflectivité des objets de l'environnement [FF13]. Cela est réalisé par le déplacement du radar au cours du temps afin d'effectuer des mesures de signaux à différentes positions. La combinaison des différentes mesures permet d'estimer plus précisément la topographie de l'environnement sous forme d'images électromagnétiques 3D, là où une seule image d'un radar fixe ne permettrait pas de discriminer deux objets dont leur distance au radar serait identique. L'information relevée se présente sous la forme d'un signal avec une amplitude et une phase. L'amplitude renseigne la réflectivité de l'objet tandis que la phase permet de mesurer les variations de positions de l'objet par rapport au radar. Le radar utilise une fréquence de porteuse comprise entre les bandes P et W (225 MHz à 110 GHz). La diffusion électromagnétique d'un objet dépend de sa taille par rapport à la longueur d'onde du signal qu'il réfléchit. Les fréquences plus faibles sont utilisées pour analyser les zones volumineuses comme des forêts, les fréquences intermédiaires sont utilisées pour des zones moyennement volumineuses comme des champs et enfin les fréquences plus élevées (bande X et K en pratique) permettent d'obtenir des images de haute résolution pour des cibles plus petites. Dans la figure B.2, nous pouvons voir à quoi ressemble une image SAR prise depuis le ciel.

Le SAR peut être utilisé en mission afin d'illuminer une zone du théâtre d'opérations et d'obtenir une image de la zone. L'avantage du SAR est qu'il peut être utilisé dans des conditions d'environnement et de météo difficile, il peut voir à travers la brume, la pluie, la neige et peut fonctionner de nuit. La prise d'image SAR prend un temps non négligeable, il y a une phase d'acquisition puis une phase de traitement. Nous pouvons observer sur la figure B.3 le besoin de la plateforme de se déplacer et de modifier l'angle de son radar afin d'obtenir assez de données pour produire une image convenable.

D'après [Gri18], un radar d'environ un mètre fonctionnant en bande X à une distance de 10 km de la zone d'intérêt mettrait environ 36 secondes à obtenir une image avec une résolution de 15 cm, en considérant une vitesse moyenne de 277 $m.s^{-1}$. Le radar est

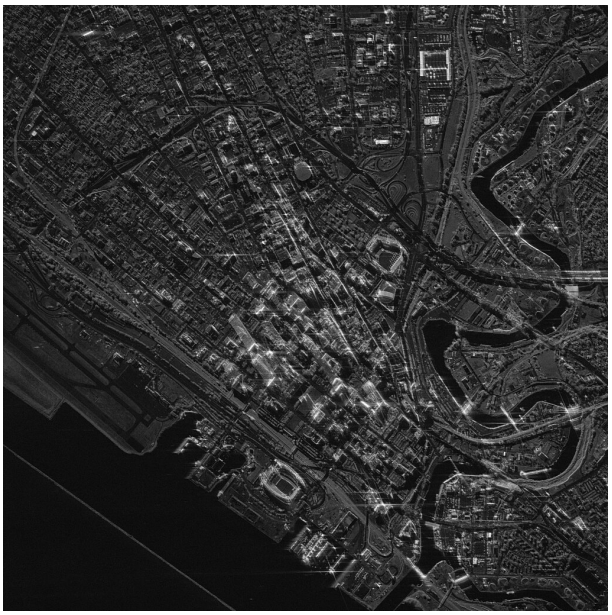

FIGURE B.2 – Imagerie SAR de Cleveland dans l'Ohio.

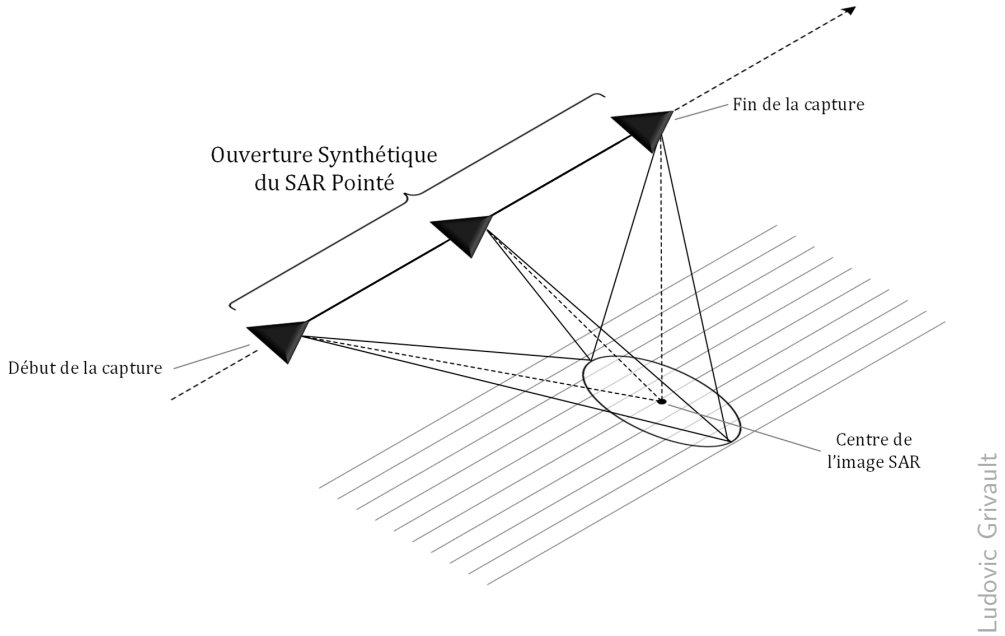

FIGURE B.3 – Schéma d'une acquisition d'une image SAR-Spot [Gri18] p. 114.

donc utilisé pendant cet interval de temps sans pouvoir être employé pour autre chose. Cependant, la zone d'observation du SAR pouvant être plus ou moins large, le radar peut

identifier plusieurs cibles à la fois. De plus, le bon fonctionnement du SAR est soumise à des conditions de positions de la plateforme et d'angle de visée du radar s'il est utilisé pour une cible particulière. Selon les conditions de la mission, le SAR n'est pas toujours la meilleure solution, et si le SAR est choisi, le plan de vol de la plateforme doit être adapté.

Le système optronique PASEO XLR

Le PASEO XLR est un équipement optronique passif développé par la société Safran [Saf]. Il permet d'imager l'environnement de nuit comme de jour, et même dans des conditions de météo difficile. Intégré dans le CMS, le senseur identifie et piste des cibles à longue portée dans un mode manuel ou automatique. Il peut également contrôler quelques armes embarquées.

D'après Safran, il a été conçu pour opérer dans des environnements navals extrêmes et possède les spécifications techniques suivantes :

- Caméra de troisième génération MWIR (MidWare InfraRed) avec zoom et option HD
- TV HD avec zoom et observateur
- Télémètre laser
- Observateur SWIR (Short Wave InfraRed)
- Plateforme gyrostabilisée haute performance
- Mesures de haute précision en LOS (Line of Sight)
- Répond à la norme américaine MIL-SPEC

Annexe C

Exemple d'utilisation de la plateforme multi-agent JADE

Dans cet exemple, un voyageur souhaite aller d'un point A à un point B, il est représenté par un agent voyageur. Un agent est utilisé pour chaque moyen de transport : agent train, agent bus et agent voiture. Une IHM permet à l'utilisateur de sélectionner son besoin, il décide d'où il part et où il veut aller (voir figure C.1).

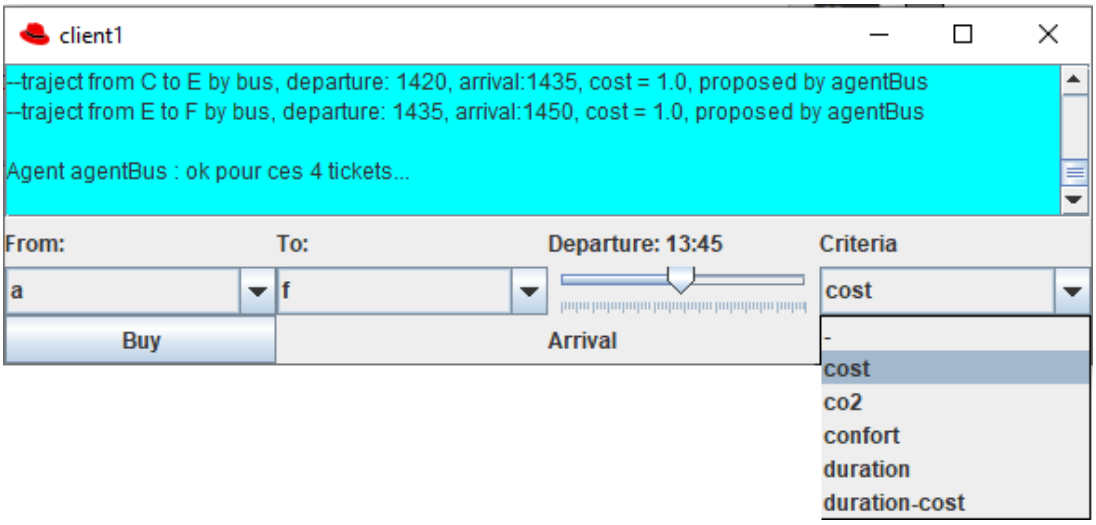

Figure C.1 – IHM du client voyageur

Un fichier CSV comprend la base de données des horaires. Ensuite, l'utilisateur précise un horaire de départ puis un critère qui déterminera le trajet le plus approprié pour lui selon le coût, le temps ou le confort. Un système d'enchère simple, le « Contract Net Protocol » (CNP), est mis en place, un score est calculé pour chaque trajet proposé et le voyage avec le score le plus élevé gagne l'enchère. L'utilisateur peut donc dire à l'agent qui a gagné l'enchère qu'il accepte son offre et confirme l'achat. Le programme est très simplifié et de nombreuses fonctionnalités pourraient être ajoutées pour le rendre plus réaliste. Les

coûts sont tous les mêmes chaque catégorie de transport, le temps de trajet est fixe et ne change pas d'un horaire à l'autre. Au niveau des fonctionnalités, il pourrait y avoir un agent qui alerte en cas d'annulation d'un trajet, dans ce cas il alerte l'agent voyageur qui va sélectionner le trajet suivant disponible avec le meilleur score. De plus, il serait possible de mettre en vente un certain nombre de tickets et de décrémenter ce nombre à chaque achat dans la base de connaissance de l'agent qui vend un voyage. Plusieurs agents voyageurs pourraient demander des voyages et certains trajets pourraient ne plus être disponibles.

Cet exemple permet d'observer les fonctionnalités de JADE qui propose une interface graphique avec un *sniffer*. La figure C.2 présente les interactions entre les agents. Les échanges s'opèrent lorsque l'utilisateur appuie sur le bouton « Buy » qu'on peut voir sur la figure C.1. Nous pouvons observer que l'agent client fait une demande de proposition (*Call For Proposal* : CFP) à tous les agents de transports, qui en retour proposent des trajets. L'agent bus gagne l'enchère, car c'est son trajet que le client choisit, il informe qu'il a bien pris en compte sa demande.

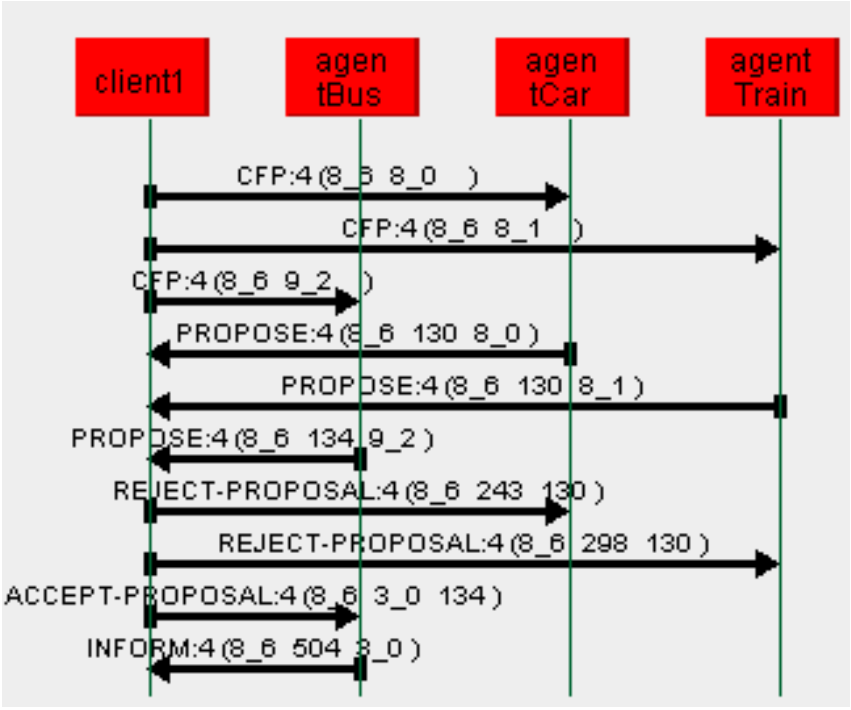

FIGURE C.2 – Vue du Sniffer de l'IHM de JADE : Echanges entre les agents.

L'exemple de l'agence de voyages peut être vu comme une source d'inspiration pour la

conception de notre système multi-agent. Dans notre contexte, l'utilisateur doit également exprimer une requête qui conditionnera les propositions faites par les agents. Les critères de choix d'un trajet par l'utilisateur de l'agence de voyages peuvent correspondre de manière analogue aux critères de discrétion, de précision, de rapidité, de fiabilité ou de disponibilité demandés par l'opérateur. Les agents définissent ensuite un score sur chaque proposition en fonction de la cohérence avec les différents critères stipulés. Dans le cas de l'agence de voyages, les agents ne coopèrent pas, ils proposent chacun des voyages avec des scores au client et c'est au client de décider. Les agents voyages ont tous une base de connaissance sur les trajets et leurs coûts, ce qui permet de définir le meilleur choix en fonction des critères, cela correspond à la vision qu'ils ont de leur environnement.

Annexe D

Méthodologies et plateformes de conception des SMA

Le développement et la création d'applications à base de systèmes multi-agents amènent des problématiques de conceptions, et des outils sont alors nécessaires. Différents langages de modélisation des interactions sont utilisés pour développer des SMA (AUML [OVDPB00], COOL [BF95]) et de nombreuses méthodologies ont été créées pour développer efficacement un SMA.

Selon Wooldridge [Woo09], les méthodologies d'analyse et de conception permettent de comprendre un peu plus le système pour ensuite le concevoir. Ces méthodologies se composent de collections de modèles accompagnés de lignes directrices. Les modèles de départs sont très abstraits, puis, au fil des analyses et des processus de conception, les modèles deviennent de plus en plus concrets, détaillés et proches de l'implémentation finale. Nous pouvons énumérer une liste non exhaustive de méthodologies décrites dans la littérature : MAS-commonKADS [IGGV98], GAIA [WJK00, ZJW03], MaSE [DWS01], TROPOS [BPG+04], ADELFE [PG04], Agile-PASSI [CCS+06], DIAMOND [Jam16], ASEME [SM22], etc. Dans le chapitre 3, nous avons utilisé la méthodologie GAIA en suivant des étapes d'analyse et de conception. L'étape suivante, qui ne fait pas partie de cette méthodologie, consiste à implémenter et développer les concepts dans un langage de programmation pour ensuite les tester et les valider.

Dans cet objectif, des plateformes pour simuler les systèmes à base d'agents sont développées depuis des années, nous en avons recensé un certain nombre en les comparant à la manière de l'article [PLPG20] qui compare les plateformes selon les critères suivants : open source, dernière mise à jour et spécialisation de la plateforme (modélisation comportement humain et modèles sociaux, recherche pour l'intelligence artificielle, simulation et modélisation d'environnement, simulation de transports). D'autres critères de comparaison sont possibles, par exemple : la représentation des agents, le langage de développement, la reproductibilité, les communications utilisées, la documentation, les exemples, la facilité

de développement et la communauté [RHLP16].

L'article [KB15] propose une analyse approfondie des plateformes à base d'agents, mettant en lumière les critères essentiels pour leur comparaison. Les auteurs examinent divers aspects, tels que les propriétés de la plateforme, sa facilité d'utilisation, sa capacité opérationnelle, la gestion de la sécurité et un point de vue pragmatique. Les propriétés des plateformes incluent des informations sur les développeurs, le domaine d'utilisation de l'outil, la licence, la dernière mise à jour et si la plateforme est *open source*. La facilité d'utilisation découle de sa simplicité, de l'apprentissage de l'outil, de sa mise à l'échelle, des compatibilités avec les standards, et des communications qu'intègrent les agents. La capacité opérationnelle de la plateforme considère les critères de performance, de stabilité ou de robustesse de la plateforme, incluant également le langage de programmation utilisé et le système d'exploitation disponible. L'aspect pragmatique des plateformes s'intéresse à divers critères supplémentaires tels que la manière de l'installer, les guides et supports à son utilisation, sa popularité ou son cout d'exploitation.

Selon ces modèles de comparaison, nous avons pu établir un tableau comparatif des différentes plateformes, ce tableau est subjectif et correspond aux informations que nous avons pu trouver sur ces plateformes (*Tableau mis à jour : 11/2023*). De plus, beaucoup de plateformes servent à faire de la modélisation, ce qui n'est pas le but de nos travaux. JADE (Java Agent Development) est populaire par les nombreuses librairies incluses, l'utilisation de standards FIPA (Foundation for Intelligent Physical Agents), la possibilité de simuler des systèmes distribués, l'interface graphique pour le développement des agents, l'aspect *open source* et l'abstraction de la complexité de développement des SMA [DKJ18]. JADE s'avère être une plateforme complète pour un premier développement d'agents, cette plateforme est utilisée pour expérimenter des SMA dans de nombreux projets [dNVvSL17, EH17]. De plus, de multiples tutoriels sont mis à disposition de la communauté. Un exemple concret, que nous détaillons en annexe, est présenté dans [emm19].

Plateforme	Langage de programmation	Mise à jour	Tutoriels, exemples, documentation [1]	Projets, articles	Communauté
GAMA	GAML (Java)	04/2023	++++	++++	++++
JaCaMo	Jason	05/2023	++++	+++	++
MASON	Java	07/2019	++++	+++	++
D-MASON	Java	07/2019	++	+	++
MASS	Java/C++, CUDA	2015	+	+	++
SPADE	Python >=3.8	06/2023	+++	++	++
SeSam	Java	2012	++	+	+
JIAC-V	Java	12/2017	+++	+++	++
Repast, RepastHPC	ReLogo/Java	10/2023	+++	++++	+++
JADE	Java	12/2022	++++	++++	++++
Flame	XMML/C	2014	++	++	+
NetLogo	Logo	11/2023	++++	++++	++++
PADE	Python	01/2021	++	+	+

TABLE D.1 – Différence entre les domaines naval et aéronautique

1. + : faible, ++ : moyen, +++ : bon, ++++ : excellent

Annexe E

Outils de sérialisation

E.1 ZeroMQ : transfert de la donnée

ZeroMQ est une bibliothèque qui facilite l'échange de messages entre différents programmes en utilisant des *sockets*. Elle se distingue par sa portabilité sur différents systèmes d'exploitation et langages de programmation, notamment Java et C++, utilisés dans notre contexte. Le préfixe « zero » indique l'absence d'un intermédiaire appelé « message broker », ce qui la différencie d'autres middlwares orientés messages (MOM). ZeroMQ se distingue par sa capacité à offrir une interopérabilité remarquable entre différentes parties de code, quel que soit leur emplacement. Grâce à sa légèreté, sa rapidité et sa faible latence, ZeroMQ permet une communication efficace et réactive entre les composants logiciels. Cette bibliothèque propose notamment le modèle *publish/subscribe*, offrant une flexibilité accrue dans la communication entre les composants.

ZeroMQ ne gère pas les données envoyées, à l'exception de leur taille en octets. Cela implique que les données doivent être formatées de manière appropriée pour être lues par l'application lors de la réception. Les librairies spécialisées comme *Protocol Buffers* servent à sérialiser la donnée transmise dans un sens et la dé-sérialiser dans l'autre. Si on envoie par exemple un *string* d'un langage à l'autre, il n'est pas encodé de la même façon, un octet *NULL* se met à la fin du *string* dans certains langages tandis que dans d'autres, le *string* commence par un nombre qui correspond à sa taille. Pour utiliser ZMQ, nous avons inclus les librairies *cppzmq* et *JeroMQ* dans nos projets.

E.2 Protocol Buffers : sérialisation de la donnée

Protocol Buffers est un format de sérialisation développé par Google qui se sert d'un langage de description d'interface (IDL) décrivant une structure de données afin d'échanger des messages entre des applications codées dans différents langages. Dans notre cas, la sérialisation est faite entre une application Java et un simulateur en C++. En compilant

deux fois le fichier .proto dans chacun des langages avec le compilateur *protoc*, nous obtenons des « *stubs* » qui permettent de coder les données en fonction des types spécifiés dans le fichier IDL. En C++, proto générera deux fichiers, un « .h » et un « .cc », en Java ce sera un fichier « .java » comprenant des *class* pour chaque message ainsi que des *Builder*.

Nous avions besoin de transmettre des données de pistes (*track*) et des données sur les plateformes alliées afin que les agents puissent travailler avec les informations utiles de l'environnement. Pour cela, nous avons écrit un fichier *proto* définissant chaque message à envoyer. Les données de pistes proviennent des senseurs présents sur les plateformes et la fusion de données de ces pistes permet d'établir la situation tactique. La figure E.1 présente un exemple de message dans un fichier *proto*.

```
41 message SensorStatus{
42     string id                = 1;
43     bool availability        = 2;
44     SensorType type          = 3;
45     double detectionRange    = 4;
46 }
47
48 message Track{
49     string id              = 1;
50     string idTrack         = 2;
51     Location loc           = 3;
52     Velocity vel           = 4;
53     repeated SensorStatus sensors  = 5;
54     TOA toa = 6;
55     string trackAppartenancy = 7;
56 }
```

FIGURE E.1 – Extrait du contenu du fichier track.proto.

D'une part, une interface écrite en C++ sérialise la donnée avant de l'envoyer via ZMQ, et une autre interface reçoit également les tâches à accomplir en retour. D'autre part, une interface Java désérialise des messages reçus et une autre répond par l'envoi d'un ordonnancement.

Annexe F

Export de métriques dans PerfTest

F.1 Utilisation de Prometheus dans PerfTest

PerfTest permet de visualiser des métriques à l'aide d'un système de *monitoring*. Dans notre cas, nous nous intéressons à Prometheus [Tur18], que PerfTest intègre dans ses options. RabbitMQ exporte également des données sur un serveur Prometheus en activant un plugin.

Une fois installé, Prometheus peut être configuré avec un fichier YAML (*Yet Another Markup Language*) « prometheus.yml » localisé dans le dossier d'installation de l'outil dont voici un exemple utilisé pour nos travaux :

```
global:
  scrape_interval:     3s

scrape_configs:
  - job_name: prometheus
    static_configs:
      - targets: ['localhost:9090']

  - job_name: perf-test
    scrape_interval: 3s
    static_configs:
      - targets: ['localhost:8080']

  - job_name: rabbitmq_grafana
    scrape_interval: 3s
    static_configs:
      - targets: ['localhost:8090']

  - job_name: rabbitmq_prometheus
```

```
20      static_configs:
21        - targets: ['localhost:15692']
```

Le premier bloc *global* contient des paramètres généraux concernant le comportement du serveur Prometheus. Le paramètre *scrape_interval* définit l'intervalle de temps entre chaque requête à la base de données du serveur Prometheus par une application externe. Ce paramètre détermine également la résolution des séries temporelles. Une série temporelle représente une collection ordonnée de données qui sont enregistrées et indexées en fonction du temps. Dans notre cas, elle peut représenter le débit, la latence ou le nombre de requêtes dans le réseau. Pour revenir à l'exemple, cela signifie qu'une valeur sera récupérée toutes les 3 secondes. Cet intervalle peut également être spécifié au niveau des *jobs*, qui écrasera le paramètre global.

Le second bloc *scrape_configs* spécifie les cibles (ou *target*) où sont exportées les métriques. Dans l'exemple, le serveur Prometheus va récupérer les données sur quatre *targets*. Le port 9090 correspond aux métriques propres à Prometheus, elles sont automatiquement exportées lorsque le serveur est en fonctionnement et ne nécessite pas de configurations. Le port 8080 présente les métriques de PerfTest à condition d'avoir spécifié en argument « `--metrics-prometheus` ». Le port 8090 est un port utilisé pour exporter nos propres métriques, il ne fonctionne pas avec PerfTest, comme nous pouvons le voir sur la figure F.1. Enfin, le port 15692 expose les métriques du serveur RabbitMQ si le plugin est activé. L'adresse *localhost :9090/metrics* est utilisée pour accéder aux métriques de Prometheus.

Prometheus possède également un moteur de recherche de métriques et un outil graphique. Les métriques pourront donc être affichées dans un tableau ou dans un graphe (voir figure F.2). Le moteur de recherche supporte les requêtes du langage PromQL, ce qui permet, par exemple, de récupérer des vecteurs de données plutôt qu'une valeur discrète. Un exemple de requête, présenté ci-dessous, démontre la richesse de ce langage.

```
1 avg_over_time(label_replace({__name__=~"TestRabbitMQ_debit_Consommateur_
    .*"},"name_label","$1","__name__", "(.+)")[60s:])
```

Dans cet exemple, la requête prend toutes les séries temporelles associées au débit des consommateurs de messages RabbitMQ. Elle remplace l'étiquette par le nom du consommateur, ce qui permettra d'afficher Consommateur_1 plutôt que « TestRabbitMQ_debit_ Consommateur_Consommateur_1 ». Ensuite, la requête fait une moyenne de ces valeurs sur une période de 60 secondes, ce qui permet d'afficher un débit moyen en réception de chaque consommateur.

La partie graphique de Prometheus se veut limitée, elle est peu flexible pour afficher

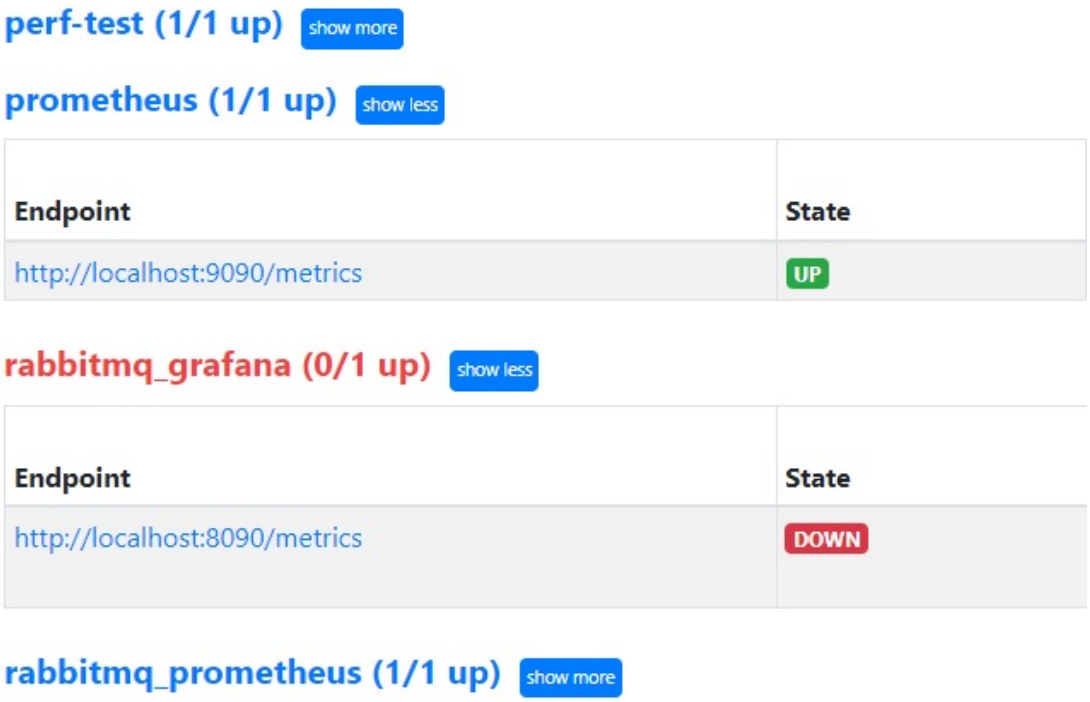

FIGURE F.1 – Représentation des « *targets* » activées lors de l'exécution de PerfTest sur l'interface de Prometheus.

des métriques et sert principalement à débuguer. C'est pour cette raison que Prometheus est combiné à des outils de visualisation plus développés, en particulier Grafana.

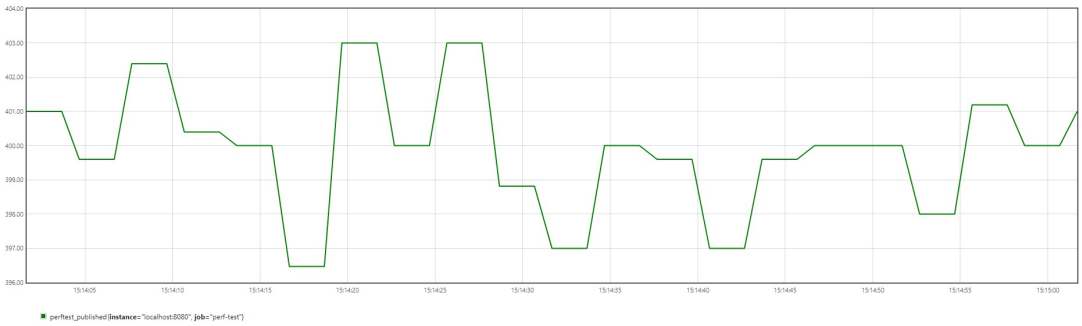

FIGURE F.2 – Représentation graphique Prometheus du nombre de messages publiés au cours du temps.

F.2 Grafana

La documentation de RabbitMQ sur l'utilisation de Prometheus conseille fortement d'utiliser conjointement l'outil Grafana [CK21] afin de visualiser les métriques du système. RabbitMQ propose un plugin pour le management du système en accédant à l'adresse localhost sur le port 15672, celui-ci collecte des métriques pour les afficher dans l'interface graphique. Un autre plugin permet d'activer l'export de métriques au format Prometheus, ce qui permet à RabbitMQ de collecter et d'exposer des métriques, telles que le nombre de messages en file d'attente, pour que le serveur Prometheus puisse les exploiter.

L'interface proposée par RabbitMQ permet la surveillance en temps réel des files d'attente, des *exchanges*, des *channels* en fournissant des graphes temporels simples permettant de détecter rapidement des anomalies dans le système. De plus, cette interface graphique autorise son utilisateur à créer directement des éléments de RabbitMQ en cours de simulation, que ce soit des files d'attentes ou même un test d'envoi d'un message. Cependant, la simplicité de l'interface de management de RabbitMQ limite le stockage de la donnée sur quelques heures. L'interface ne peut pas être personnalisée pour créer des tableaux de bord personnalisés et observer de nouvelles métriques, justifiant ainsi l'utilisation d'un autre outil de monitoring.

Par défaut, Grafana utilise le port 3000 en localhost. Pour récupérer des données, Grafana doit ajouter une source de données, Prometheus dans notre cas. L'adresse du serveur Prometheus sur le port 9090 permet à Grafana de récupérer toutes les métriques des différentes *targets*. Le *dashboard* de la figure F.4 présente un exemple d'acquisition de données pendant deux heures. La figure F.3 présente un *panel* du *dashboard* de PerfTest et affiche les mêmes données que la figure F.2, ce qui nous permet de voir l'évolution des représentations graphiques entre les deux outils.

Dans Grafana, les *panels* peuvent être regroupés dans des *rows*, la taille de chaque graphe ainsi que leur positionnement peuvent être facilement modifiés. Il y a une variété de *panels* disponibles, comprenant des diagrammes à barres, des séries temporelles, des jauges, des tables, des *heatmaps*, des histogrammes, des organigrammes, des listes d'alertes, et bien d'autres.

FIGURE F.3 – Représentation graphique Grafana du nombre de messages publiés au cours du temps.

FIGURE F.4 – Dashboard de PerfTest importé depuis le Grafana Labs.

TABLE DES FIGURES

Liste des publications

[1] Paul Quentel. Architecture distribuée de collaboration pour systèmes critiques : application aux systèmes navals. In *31e journées francophones sur Systèmes multi-agent (JFSMA)*, page 53, 2023.

[2] Paul Quentel, Yvon Kermarrec, Ludovic Grivault, Pierre Le Berre, and Laurent Savy. Approche multi-agent pour la collaboration multi-plateforme dans un contexte de défense navale. In *Rencontres des Jeunes Chercheurs en Intelligence Artificielle (RJCIA 2023)*, 2023.

[3] Paul Quentel, Yvon Kermarrec, Pierre Le Berre, Ludovic Grivault, and Laurent Savy. A rabbitmq-based framework to deal with naval sensor systems design complexity. In *Science and Information Conference*, pages 948–959. Springer, 2023.

BIBLIOGRAPHIE

[ABM22] Advanced Battle Management System(ABMS). *Congressional Research Service*, Février 2022.

[ASD18] TEKEVER ASDS. CAMELOT specifications. Délivrable 740736, Projet Européen CAMELOT, Novembre 2018.

[AYBA20] Noureddine El Abid Amrani, Mohamed Youssfi, Omar Bouattane, and Oum El Kheir Abra. Interoperability between heterogeneous multi-agent systems recommended by FIPA : Towards a weakly coupled approach based on a network of recurrent neurons of the lstm type. In *2020 3rd International Conference on Advanced Communication Technologies and Networking (CommNet)*, pages 1–6. IEEE, 2020.

[Bak01] David Bakken. Middleware. *Encyclopedia of Distributed Computing*, 11, 2001.

[Bar] Frédéric Barbaresco. IA embarquée pour la surveillance appliquée aux domaines radar, sonar et caméra. `https://pfia2021.fr/journees/defense/Session5-AI_for_Surveillance-ONERA_THALES_AIRBUSfinal.pdf`. Consulté le 20/07/2021.

[BCC+14] Philippe Billaud, Claude Chanot, Philippe Condette, Guy Desodt, Alain Jeantet, Thierry Jurand, Michel Moruzzis, and Dominique Peyrard. Radars de surface, radars de défense terrestres et navals. *Techniques de l'ingénieur Technologies radars et applications*, base documentaire : TIP385WEB.(ref. article : te6679), 2014.

[BDS] Ted Bapty, James Davis, and Jason Scott. Future Airborne Capability Environment.

[BF95] Mihai Barbuceanu and Mark S Fox. COOL : A Language for Describing Coordination in Multi Agent Systems. In *ICMAS*, pages 17–24, 1995.

[ble19] Cols bleus. Une marine en pointe. *Le magazine de la Marine Nationale*, 3083, Novembre 2019.

[ble20] Cols bleus. Le porte-avions : Pièce maîtresse de la stratégie française. *Le magazine de la Marine Nationale*, 3092, Décembre 2020.

[ble21] Cols bleus. Dossier d'information. *Le magazine de la Marine Nationale*, Hors-série, Janvier 2021.

[BMS93] Özalp Babaoğlu, Keith Marzullo, and Fred B Schneider. A formalization of priority inversion. *Real-Time Systems*, 5(4) :285–303, 1993.

[BOMJ19] Vicent Botti, Andrea Omicini, Stefano Mariani, and Vicente Julian. *Multi-agent systems*. MDPI-Multidisciplinary Digital Publishing Institute Basel, Switzerland, 2019.

[BPG+04] Paolo Bresciani, Anna Perini, Paolo Giorgini, Fausto Giunchiglia, and John Mylopoulos. Tropos : An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, 8 :203–236, 2004.

[Bro14] Trevor Brown. Distributed Computing and Communication Complexity. `https://www.cs.toronto.edu/~toni/Courses/CommComplexity2014/Lectures/lecture10a.pdf`, 2014. Consulté le 08/02/2024.

[BS10] Parasumanna Gokulan Balaji and Dipti Srinivasan. An introduction to multi-agent systems. *Innovations in multi-agent systems and applications-1*, pages 1–27, 2010.

[BU20] Jérôme Bordellès and Mickaël Ulvoa. Les besoins actuels et futurs en fréquences pour les armées : un défi stratégique pour la france. *Annales des Mines*, 9 :78–88, Mars 2020.

[BUL+18] Jacob Beal, Kyle Usbeck, Joseph Loyall, Mason Rowe, and James Metzler. Adaptive opportunistic airborne sensor sharing. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 13(1) :1–29, 2018.

[CBH09] Han-Lim Choi, Luc Brunet, and Jonathan P How. Consensus-based decentralized auctions for robust task allocation. *IEEE transactions on robotics*, 25(4) :912–926, 2009.

[CCDN+21] Lorenzo Canese, Gian Carlo Cardarilli, Luca Di Nunzio, Rocco Fazzolari, Daniele Giardino, Marco Re, and Sergio Spanò. Multi-agent reinforcement learning : A review of challenges and applications. *Applied Sciences*, 11(11) :4948, 2021.

[CCS+06] Antonio Chella, Massimo Cossentino, Luca Sabatucci, Valeria Seidita, et al. Agile passi : An agile process for designing agents. *International Journal of Computer Systems Science & Engineering*, 21(2) :133–144, 2006.

[CDJM01] Brahim Chaib-Draa, Imed Jarras, and Bernard Moulin. Systèmes multi-agents : principes généraux et applications. *Edition Hermès*, 242 :1030–1044, 2001.

[CDKB05] George F Coulouris, Jean Dollimore, Tim Kindberg, and Gordon Blair. *Distributed systems : concepts and design*. Pearson Education, Addison-Wesley, 5 edition, 2005.

[cem] `https://cmano-db.com/`. Consulté le 20/11/2023.

[Cha07] Yasmine Charif. *Chorégraphie dynamique de services basée sur la coordination d'agents introspectifs*. PhD thesis, Paris 6, 2007.

[CK21] Mainak Chakraborty and Ajit Pratap Kundan. Grafana. In *Monitoring Cloud-Native Applications : Lead Agile Operations Confidently Using Open Source Software*, pages 187–240. Springer, 2021.

[CLL05] Kang Chen, Xue-Shan Luo, and Xiang Liu. Sensor resource management research based on intelligent agent in naval multi-platform cooperative engagement. In *2005 International Conference on Machine Learning and Cybernetics*, volume 1, pages 132–136. IEEE, 2005.

[Cos18] Benjamin Coste. *Détection contextuelle de cyberattaques par gestion de confiance à bord d'un navire*. Theses, Ecole nationale supérieure Mines-Télécom Atlantique, December 2018.

[dAM16] DGITM Direction des Affaires Maritimes. « Cyber Securité » - Evaluer et protéger le navire, Septembre 2016.

[DBGP21] Alaa Daoud, Flavien Balbo, Paolo Gianessi, and Gauthier Picard. Un modèle agent générique pour la comparaison d'approches d'allocation de ressources dans le domaine du transport à la demande. In *JFSMA 2021 : 29ème Journées Francophones sur les Systèmes Multi-Agents*, pages 127–136, Bordeaux, France, June 2021. Cépaduès.

[DKJ18] Ali Dorri, Salil S Kanhere, and Raja Jurdak. Multi-agent systems : A survey. *IEEE Access*, 6 :28573–28593, 2018.

[dNVvSL17] Nathalia Moraes do Nascimento, Carlos Juliano M Viana, Arndt von Staa, and Carlos Lucena. A Publish-Subscribe based Architecture for Testing Multiagent Systems. In *SEKE*, pages 521–526, 2017.

[Dos14] David Dossot. *RabbitMQ essentials*. Packt Publishing Ltd, 2014.

[DPSG19] David Deptula, Heather Penney, Lawrence Stuzriem, and Mark Gunzinger. Restoring America's military competitiveness : Mosaic Warfare. *Mitchell institute for aerospace studies*, September 2019.

[DWS01] Scott A DeLoach, Mark F Wood, and Clint H Sparkman. Multiagent systems engineering. *International Journal of Software Engineering and Knowledge Engineering*, 11(03) :231–258, 2001.

[EH17] Jamal El Hachem. *A Model Driven Method to Design and Analyze Secure System-of-Systems Architectures : Application to Predict Cascading Attacks in Smart Buildings*. PhD thesis, Pau, 2017.

[emm19] Jade appliqué cours 3 : protocoles et diffusion. `http://emmanuel.adam.free.fr/site/spip.php?article175`, 2019. Consulté le 23/11/2023.

[FF13] Laurent Ferro-Famil. Principes de l'imagerie radar à synthèse d'ouverture (rso). *Techniques de l'ingénieur Le traitement du signal et ses applications*, base documentaire : TIP383WEB.(ref. article : te6702), 2013.

[Ger06] Basil Germond. Les forces navales européennes face aux « nouvelles menaces » en mer. *Relations internationales*, 125 :45–58, 2006.

[GHSVE22] Ludovic Grivault, Thibault Hernoust, Daniel Stofer, and Nicolas Van Eenoo. Procédé et dispositif de sélection des ressources capteurs d'un système multi-capteur, 06 2022.

[Gra18] Tim Grayson. Mosaic warfare. *DARPA/STO*, pages 11–13, 2018.

[Gri18] Ludovic Grivault. *Architecture multi-agent pour la conception et l'ordonnancement de systèmes multi-senseur embarqués sur plateformes aéroportées*. PhD thesis, Sorbonne université, 2018.

[Gro19] Vincent Groizeleau. FREMM : la veille coopérative prévue en 2021. https ://www.meretmarine.com/fr/defense/fremm-la-veille-cooperative-prevue-en-2021, Juin 2019.

[Gua21] Davide Andrea Guastella. Scheduling Plans of Tasks. *arXiv preprint arXiv :2102.03555*, 2021.

[GVDPTL18] Marisol García-Valls, Jorge Domínguez-Poblete, Imad Eddine Touahria, and Chenyang Lu. Integration of data distribution service and distributed partitioned systems. *Journal of Systems Architecture*, 83 :23–31, 2018.

[Hen09] Peter Henderson. Modular open systems architecture. *April (2009)*, pages 1–15, 2009.

[HM17] Raymond R Hill and John O Miller. A history of United States military simulation. In *2017 Winter Simulation Conference (WSC)*, pages 346–364. IEEE, 2017.

[HMR93] Jean-Michel Hélary, Achour Mostefaoui, and Michel Raynal. *Déterminer un état global dans un système réparti*. IRISA, 1993.

[Hop95] Johns Hopkins. The Cooperative Engagement Capability. *APL Technical Digest*, 16(4) :377–396, 1995.

[Huh12] Michael N Huhns. *Distributed Artificial Intelligence*, volume 1. Elsevier, 2012.

[IGGV98] Carlos A Iglesias, Mercedes Garijo, José C González, and Juan R Velasco. Analysis and design of multiagent systems using MAS-CommonKADS. In *Intelligent Agents IV Agent Theories, Architectures, and Languages : 4th International Workshop, ATAL'97 Providence, Rhode Island, USA, July 24–26, 1997 Proceedings 4*, pages 313–327. Springer, 1998.

[Jam16] Jean-Paul Jamont. *Démarche, modèles et outils multi-agents pour l'ingénierie des collectifs cyber-physiques*. PhD thesis, Université Grenoble Alpes, 2016.

[Jan24] Janes. *Janes Fighting Ships*. Janes, 2023-2024.

[Jia15] Yichuan Jiang. A survey of task allocation and load balancing in distributed systems. *IEEE Transactions on Parallel and Distributed Systems*, 27(2) :585–599, 2015.

[JPCH11] Luke Johnson, Sameera Ponda, Han-Lim Choi, and Jonathan How. Asynchronous decentralized task allocation for dynamic environments. In *Infotech@ Aerospace 2011*, page 1441. American Institute of Aeronautics and Astronautics, 2011.

[KB15] Kalliopi Kravari and Nick Bassiliades. A survey of agent platforms. *Journal of Artificial Societies and Social Simulation*, 18(1) :11, 2015.

[KNF10] Stéphane Kemkemian and Myriam Nouvel-Fiani. Toward common radar & EW multifunction active arrays. In *2010 IEEE International Symposium on Phased Array Systems and Technology*, pages 777–784. IEEE, 2010.

[Kus97] Eyal Kushilevitz. Communication complexity. In *Advances in Computers*, volume 44, pages 331–360. Elsevier, 1997.

[Lej14] Jonathan Lejeune. *Algorithmique distribuée d'exclusion mutuelle : vers une gestion efficace des ressources*. PhD thesis, Paris 6, 2014.

[LKK+13] Marco Lützenberger, Tobias Küster, Thomas Konnerth, Alexander Thiele, Nils Masuch, Axel Heßler, Jan Keiser, Michael Burkhardt, Silvan Kaiser, and Sahin Albayrak. JIAC V : A MAS framework for industrial applications. pages 1189–1190, 2013.

[LMS+08] Grace Lewis, Edwin Morris, Soumya Simanta, Dennis Smith, and Lutz Wrage. Smart : Analyzing the reuse potential of legacy components in a service-oriented architecture environment. In *AIAA Infotech@ Aerospace 2007 Conference and Exhibit*, page 2865, 2008.

[LT19] Yanjiang Li and Chong Tan. A survey of the consensus for multi-agent systems. *Systems Science & Control Engineering*, 7(1) :468–482, 2019.

[MBL+20] Forrest E Morgan, Benjamin Boudreaux, Andrew J Lohn, Mark Ashby, Christian Curriden, Kelly Klima, and Derek Grossman. Military applications of artificial intelligence. *Santa Monica : RAND Corporation*, 2020.

[MCLC22] Antoine Milot, Estelle Chauveau, Simon Lacroix, and Charles Lesire Cabaniols. Allocation par enchères et planification hiérarchique pour un système multirobot, application au cas de la chasse aux mines. In *JFSMA 2022 : 30èmes Journées Francophones sur les Systèmes Multi-Agents*, 2022.

[Mey15] Jean-Louis Meyzonette. Optronique : paramètres de base. *Techniques de l'ingénieur Optique Photonique*, base documentaire : TIP520WEB.(ref. article : e4000), 2015.

[MIAP19] Milica Matić, Sandra Ivanović, Marija Antić, and Istvan Papp. Health Monitoring and Auto-Scaling RabbitMQ Queues within the Smart Home System. In *2019 IEEE 9th International Conference on Consumer Electronics (ICCE-Berlin)*, pages 380–384. IEEE, 2019.

[ML17] Pedro Merino Laso. *Détection de dysfonctionements et d'actes malveillants basée sur des modèles de qualité de données multi-capteurs.* Theses, Ecole nationale supérieure Mines-Télécom Atlantique, December 2017.

[MPS02] Pavlos Moraïtis, Eleftheria Petraki, and Nikolaos I Spanoudakis. Engineering JADE agents with the Gaia methodology. In *Net. ObjectDays : International Conference on Object-Oriented and Internet-Based Technologies, Concepts, and Applications for a Networked World*, pages 77–91. Springer, 2002.

[MSK+22] Tim Marler, Carra S Sims, Ajay K Kochhar, Christine Kistler LaCoste, Caitlin Lee, Matthew Strawn, and Mark Toukan. *What Is JADC2, and How Does It Relate to Training ? : An Air Force Perspective on Joint All Domain Command and Control.* RAND, 2022.

[Nao22] Douraïd Naouar. *MoRiA Une méthode basée sur les modèles pour l'analyse des risques de cybersécurité : application à un système complexe de défense*

navale. PhD thesis, Ecole nationale supérieure Mines-Télécom Atlantique Bretagne Pays de la Loire, 2022.

[Nata] Marine Nationale. Nos équipements Marine. https ://www.defense.gouv.fr/marine/ nos-equipements-marine. Consulté le 04/05/2023.

[Natb] Marine Nationale. Porte-avions. https ://www.defense.gouv.fr/marine/forces-surface/porte-avions. Consulté le 20/11/2023.

[Natc] Marine Nationale. Rafale Marine. https ://www.defense.gouv.fr/marine/ aeronefs/rafale-marine. Consulté le 20/11/2023.

[OKS20] Michael Otte, Michael J Kuhlman, and Donald Sofge. Auctions for multi-robot task allocation in communication limited environments. *Autonomous Robots*, 44 :547–584, 2020.

[OVDPB00] James J Odell, Harry Van Dyke Parunak, and Bernhard Bauer. Representing agent interaction protocols in UML. In *International Workshop on Agent-Oriented Software Engineering*, pages 121–140. Springer, 2000.

[Pap03] Christos H Papadimitriou. Computational complexity. In *Encyclopedia of computer science*, pages 260–265. 2003.

[Per23] RabbitMQ perftest. https ://rabbitmq.github.io/rabbitmq-perf-test/stable/htmlsingle/, 2023. Consulté le 10/02/2024.

[PG04] Gauthier Picard and Marie-Pierre Gleizes. The ADELFE methodology. In *Methodologies and Software Engineering for Agent Systems : The Agent-oriented Software Engineering Handbook*, pages 157–175. Springer, 2004.

[PIA20] PIAP. CAMELOT test plan. Délivrable, Projet Européen CAMELOTn, 2020.

[PLPG20] Constantin-Valentin Pal, Florin Leon, Marcin Paprzycki, and Maria Ganzha. A review of platforms for the development of agent systems. *arXiv preprint arXiv :2007.08961*, 2020.

[PRC+10] Sameera Ponda, Josh Redding, Han-Lim Choi, Jonathan P How, Matt Vavrina, and John Vian. Decentralized planning for complex missions with dynamic communication constraints. In *Proceedings of the 2010 American Control Conference*, pages 3998–4003. IEEE, 2010.

[QKG+23] Paul Quentel, Yvon Kermarrec, Ludovic Grivault, Pierre Le Berre, and Laurent Savy. Approche multi-agent pour la collaboration multi-plateforme dans un contexte de défense navale. In *Rencontres des Jeunes Chercheurs en Intelligence Artificielle (RJCIA 2023)*, 2023.

[QKLB+23] Paul Quentel, Yvon Kermarrec, Pierre Le Berre, Ludovic Grivault, and Laurent Savy. A rabbitMQ-based framework to deal with naval sensor systems design complexity. In *Science and Information Conference*, pages 948–959. Springer, 2023.

[Rab] RabbitMQ. Server Documentation : Monitoring. https ://rabbitmq-website.pages.dev/docs/monitoring. Consulté le 22/02/2024.

[Rab23] RabbitMQ Tutorials. https ://www.rabbitmq.com/getstarted.html, 2023. Consulté le 10/02/2024.

[Ray12] Michel Raynal. *Concurrent programming : algorithms, principles, and foundations*. Springer Science & Business Media, 2012.

[RB86] Michel Raynal and D Beeson. *Algorithms for mutual exclusion*. MIT press, 1986.

[RB14] Tiago Rosado and Jorge Bernardino. An overview of openstack architecture. In *Proceedings of the 18th International Database Engineering & Applications Symposium*, pages 366–367, 2014.

[RBA05] Wei Ren, Randal W Beard, and Ella M Atkins. A survey of consensus problems in multi-agent coordination. In *Proceedings of the 2005, American Control Conference, 2005.*, pages 1859–1864. IEEE, 2005.

[RHLP16] Alban Rousset, Bénédicte Herrmann, Christophe Lang, and Laurent Philippe. A survey on parallel and distributed multi-agent systems for high performance computing simulations. *Computer Science Review*, 22 :27–46, 2016.

[Ric20] Chet Richards. Boyd's OODA loop. *Necesse 2020*, 5 :142–165, 2020.

[RLLX08] Duncan Russell, Nik Looker, Lu Liu, and Jie Xu. Service-oriented integration of systems for military capability. In *2008 11th IEEE international symposium on object and component-oriented real-time distributed computing (ISORC)*, pages 33–41. IEEE, 2008.

[Saf] Safran. PASEO XLR. https ://www.safran-group.com/fr/produits-services/paseo-xlr-systeme-optronique-didentification-conduite-tir-tres-longue-portee-applications-navales. Consulté le 20/11/2023.

[SB17] Alan N Steinberg and Christopher L Bowman. Revisions to the jdl data fusion model. In *Handbook of multisensor data fusion*, pages 65–88. CRC press, 2017.

[SCL19] Garrett C Sargent, Charles Collier, and Ilya Lipkin. Designing the next generation of sensor systems using the SOSA standard. In *Open Architecture/Open Business Model Net-Centric Systems and Defense Transformation 2019*, volume 11015, pages 113–132. SPIE, 2019.

[SD15] Vaishnavi Singhal and Deepak Dahiya. Distributed task allocation in dynamic multi-agent system. In *International Conference on Computing, Communication & Automation*, pages 643–648, 2015.

[SG20] A El Fallah Seghrouchni and L Grivault. Multi-agent paradigm to design the next generation of airborne platforms. *Aerospace Lab*, pages 1–8, 2020.

[SHR+08] Espen Skjervold, Trude Hafsøe, Kjell Rose, Nils Agne Nordbotten, Ketil Lund, et al. Multinett II : SOA and XML security experiments with Cooperative ESM Operations (CESMO). 2008.

[Sin89] Mukesh Singhal. Deadlock detection in distributed systems. *Computer*, 22(11) :37–48, 1989.

[SM22] Nikolaos I Spanoudakis and Pavlos Moraitis. The ASEME methodology. *International Journal of Agent-Oriented Software Engineering*, 7(2) :79–107, 2022.

[Spe02] Anthony E Spezio. Electronic warfare systems. *IEEE Transactions on Microwave Theory and Techniques*, 50(3) :633–644, 2002.

[SRF20] Hugo Seuté, Laurent Ratton, and Antoine Fagette. Passive sensor planning for TDOA/FDOA geolocation under communication constraints. In *2020 IEEE 23rd International Conference on Information Fusion (FUSION)*, pages 1–8. IEEE, 2020.

[SSFS18] P Sommer, Florian Schellroth, M Fischer, and Jan Schlechtendahl. Message-oriented middleware for industrial production systems. In *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*, pages 1217–1223. IEEE, 2018.

[SST21] George Marios Skaltsis, Hyo-Sang Shin, and Antonios Tsourdos. A survey of task allocation techniques in MAS. In *2021 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 488–497, 2021.

[Sul20] Bastien Sultan. *Maîtrise des correctifs de sécurité pour les systèmes navals*. PhD thesis, Ecole nationale supérieure Mines-Télécom Atlantique, 2020.

[SV00] Peter Stone and Manuela Veloso. Multiagent systems : A survey from a machine learning perspective. *Autonomous Robots*, 8 :345–383, 2000.

[TE20] Itshak Tkach and Yael Edan. *Distributed Heterogeneous Multi Sensor Task Allocation Systems*. Springer, 2020.

[Tha] Thales. Radar à balayage électronique à antenne active RBE2 AESA. https ://www.thalesgroup.com/fr/active-electronically-scanned-array-aesa-rbe2-radar. Consulté le 20/11/2023.

[TLL17] Olivier Thibesard, Eric Louazon, and Emmanuel Latasse. Les liaisons de données tactiques (LDT). *Publication interarmées*, 109, june 2017.

[TMSS] Joanna Turner, Qinggang Meng, Gerald Schaefer, and Andrea Soltoggio. Distributed strategy adaptation with a prediction function in multi-agent task allocation.

[Tok17] Joyce Tokar. An examination of open systems architectures for avionics systems–an update. *Air Force FACETM TIM Paper*, 2017.

[Tur18] James Turnbull. *Monitoring with Prometheus*. Turnbull Press, 2018.

[UPT+18] UPVLC, PIAP, TEK-ASDS, BAES, and KEMEA. Architecture and Data Model. Délivrable, Projet Européen CAMELOT, Novembre 2018.

[Ven08] Pascal Vennesson. Penser les guerres nouvelles : la doctrine militaire en questions. *Pouvoirs*, 125 :81–92, 2008.

[Vog13] Bahtijar Vogel. Towards open architecture system. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, pages 731–734, 2013.

[Voi17] Jean-Luc Voirin. *Model-based system and architecture engineering with the arcadia method*. Elsevier, 2017.

[Win05] Michael Winikoff. JACK™ intelligent agents : an industrial strength platform. *Multi-agent programming : Languages, platforms and applications*, pages 175–193, 2005.

[Wit20] Dr. Thomas Withington. The Cooperative Approach. https ://www.armadainternational.com/2020/06/the-cooperative-approach/, June 3, 2020. Consulté le 10/04/2022.

[WJK00] Michael Wooldridge, Nicholas R Jennings, and David Kinny. The GAIA methodology for agent-oriented analysis and design. *Autonomous Agents and multi-agent systems*, 3 :285–312, 2000.

[Woo09] Michael Wooldridge. *An introduction to multiagent systems*. John Wiley & Sons, 2009.

[Wu17] Jun Wu. Energy-efficient concurrency control for dynamic-priority real-time tasks with abortable critical sections. *Computing and Informatics*, 36(4) :765–792, 2017.

[XCS18] Bing Xie, Jing Chen, and Lincheng Shen. Cooperation Algorithms in Multi-Agent Systems for Dynamic Task Allocation : A Brief Overview. In *2018 37th Chinese Control Conference (CCC)*, pages 6776–6781, 2018.

[YWW+19] Dong Yang, Lingxiao Wu, Shuaian Wang, Haiying Jia, and Kevin X Li. How big data enriches maritime research–a critical review of automatic identification system (AIS) data applications. *Transport Reviews*, 39(6) :755–773, 2019.

[ZJW03] Franco Zambonelli, Nicholas R Jennings, and Michael Wooldridge. Developing multiagent systems : The Gaia methodology. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 12(3) :317–370, 2003.

[ZKA+16] Aboelsood Zidan, Mutaz Khairalla, Ahmed M Abdrabou, Tarek Khalifa, Khaled Shaban, Atef Abdrabou, Ramadan El Shatshat, and Ahmed M Gaouda. Fault detection, isolation, and service restoration in distribution systems : State-of-the-art and future trends. *IEEE Transactions on Smart Grid*, 8(5) :2170–2185, 2016.

COLLEGES SCIENCES
BRETAGNE POUR L'INGENIEUR
LOIRE ET LE NUMERIQUE

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Titre : Architecture multi-agent distribuée et collaborative pour l'allocation de tâches à des senseurs : application aux systèmes navals

Mot clés : Architecture, système multi-agent, allocation de tâches à des senseurs, coopération, communication, partage d'information

Résumé : L'évolution du contexte de défense aéronaval nécessite une modification majeure de l'architecture des systèmes de senseurs actuels afin de maitriser les futures menaces et d'intégrer les nouveaux dispositifs et senseurs à venir. Ces senseurs, hétérogènes, complémentaires et embarqués sur des plateformes navales ou aériennes, sont essentiels pour l'acquisition de données de l'environnement et l'établissement de la situation tactique. Dans ce contexte, les plateformes peuvent collaborer et partager leurs ressources senseurs pour accomplir de nouvelles fonctionnalités et établir un panorama global de la situation.

Dans cette thèse, nous avons conçu et développé un système multi-agent pour l'allocation de tâches à des ressources distribuées sur des plateformes distinctes dans le but d'accomplir des capacités collaboratives. Nous présentons des scénarios illustrant les besoins opérationnels auxquels l'architecture doit répondre, établissant ainsi un cahier des charges. Ensuite, nous détaillons les étapes de la conception et de l'implémentation de cette nouvelle architecture, en décrivant chaque type d'agent et les interactions possibles entre eux. Nous proposons un algorithme d'enchère nécessitant des échanges entre les agents, soumis aux contraintes de bande passante et de latence. Enfin, nous présentons un banc d'essai intégrant des outils de capture et de visualisation de métriques du système, permettant l'évaluation des concepts d'agents et de leurs mécanismes de communication. L'objectif est de démontrer que notre architecture répond aux besoins opérationnels spécifiés, notamment le passage à l'échelle des algorithmes et des interfaces de communications des agents, la résistance aux pannes et la performance du système.

Title: Distributed and collaborative multi-agent architecture for allocating tasks to sensors: application to naval systems

Keywords: Architecture, multi-agent system, task allocation to sensors, cooperation, communication, information sharing

Abstract: The changing context of naval and aerial defense requires a major modification of current sensor system architectures to overcome future threats and to integrate next generation devices and sensors. These sensors, heterogeneous, complementary, and embedded on naval or aerial platforms, are essential for acquiring data from the environment in order to establish the tactical situation. In this context, platforms can collaborate and share their sensor resources to achieve new functionalities and set up a global overview of the situation.

In this thesis, we have designed and developed a multi-agent system for allocating tasks to distributed resources on distinct platforms in order to accomplish collaborative capabilities. We present scenarios illustrating the operational needs that the architecture must meet, thus establishing a set of specifications. Then, we detail the steps involved in designing and implementing this new architecture, describing each type of agent and the possible interactions between them. We propose an auction algorithm requiring exchanges between agents, subject to bandwidth and latency constraints. Finally, we present a test bed integrating tools for capturing and display system metrics, allowing the evaluation of agent concepts and their communication mechanisms. The objective is to demonstrate that our architecture meets the specified operational requirements, in particular the scalability of the agents' algorithms and communication interfaces, fault tolerance, and system performance.