Intégration des systèmes multi-agents aux systèmes embarqués pour la délégation de tâches

Tifaine Inguere

UNIVERSITE BRETAGNE LOIRE / EDUCATION LANGAGES, INTERACTIONS COGNITION, CLINIQUE

Le Mans Université

THESE DE DOCTORAT DE

LE MANS UNIVERSITE
COMUE UNIVERSITE BRETAGNE LOIRE

ECOLE DOCTORALE N° 603
Education, Langages, Interaction, Cognition, Clinique
Spécialité : Informatique

Par
Tifaine INGUERE

Intégration des systèmes multi-agents aux systèmes embarqués pour la délégation de tâches

Thèse présentée et soutenue à Le Mans, le 12 juin 2018
Unité de recherche : CREN Le Mans - Centre de Recherche en Éducation de Nantes
Thèse N° : 2018LEMA3002

Rapporteurs avant soutenance :

Virginie FRESSE Maître de Conférences HDR, Université Jean Monnet, Saint Etienne
Sébastien PICAULT Maître de Conférences HDR, Université de Lille

Composition du Jury :

Président : Dominique HOUZET Professeur, Grenoble-INP
Examinateurs : Virginie FRESSE Maître de Conférences HDR, Université Jean Monnet, Saint Etienne
Sébastien PICAULT Maître de Conférences HDR, Université de Lille
Dir. de thèse : Pascal LEROUX Professeur, Université du Mans

Invité(s)

Leila MERGHEM-BOULAHIA Maître de Conférences HDR, Université de Technologie de Troyes

Co-Encadrants de Thèse : Florent CARLIER Maître de Conférences, Université du Mans
Valérie RENAULT Maître de Conférences, Université du Mans

A ma famille,
A Nicolas,
A Korra et Saphyr

Résumé

Cette thèse présente comment l'intégration de systèmes multi-agents au sein de systèmes embarqués peut permettre d'optimiser la gestion des tâches. Nous relevons un manque de flexibilité pour les systèmes embarqués et posons l'hypothèse d'une solution multi-agents permettant la prise en compte dynamique du contexte d'évolution du système. Les systèmes embarqués, intégrés à l'environnement utilisateur, sont contraints en termes d'espace physique et donc de ressources matérielles. Ces limites impliquent un besoin d'optimisation des ressources. Nous proposons d'expérimenter des algorithmes de négociation multi-agents pour déléguer des tâches entre les différentes ressources. Pour valider nos hypothèses, nous détaillons les caractéristiques des systèmes multi-agents, leurs comportements, leurs modèles, les plateformes sur lesquelles ils évoluent, leurs standards de communication et leurs algorithmes sociaux. Nous avons constaté que la majorité des travaux du domaine agent se concentraient sur d'autres problématiques. Par conséquent, nous avons proposé la formalisation de systèmes multi-agents embarqués et d'une plateforme multi-agents adaptée. Nous avons ensuite expérimenté cette plateforme au sein de systèmes embarqués avec le cas d'étude du traitement d'images, notamment avec le calcul d'une interpolation de pixels. Nous avons mené des études de performances pour estimer le coût administratif d'une solution multi-agents, puis avons considéré ces résultats au regard des gains de capacité de nos systèmes embarqués. Nos dernières expérimentations mettent à l'épreuve notre solution de délégation de tâches entre plusieurs cartes embarquées dans un contexte hétérogène.

Abstract

This thesis shows how the integration of multi-agents systems within embedded systems can optimize tasks management. We notice a lack of flexibility for embedded systems and hypothesize that a multi-agents solution will allow the dynamic consideration of the system context of evolution. Embedded systems, being integrated into the user environment, are limited in terms of physical space and thus hardware resources. These limits involve the necessity to optimize the resources. We suggest experimenting multi-agents negotiation algorithms to delegate tasks between several resources. To validate our hypotheses, we detail the characteristics of multi-agents systems, their behavior, their models, the platforms on which they evolve, their communication standards and their social algorithms. We observed that the majority of the works of the multi-agents domain concentrated on other problems. Therefore, we proposed the formalization of embedded multi-agents systems and of an adapted multi-agents platform. We then experimented this platform within embedded systems with the case study of image processing, especially the calculation of a pixels interpolation. We led performance studies to estimate the administrative cost of a multi-agents solution, then considered these results in relation to the capacity earnings of our embedded systems. Our last experiments put to the test our solution of tasks delegation between several embedded cards within a heterogeneous context.

Remerciements

Avant toute chose, je tiens à remercier *Florent Carlier* et *Valérie Renault*, pour m'avoir proposé cette aventure. Je les remercie profondément pour leur encadrement, leur soutien et leur implication durant ces années de recherche à mes côtés.

Je tiens également à remercier l'entreprise STMicroelectronics et en particulier *Stéphane Henry* pour avoir apporté soutien et ressources à cette thèse. L'expérience d'un travail en partenariat avec le monde professionnel fut particulièrement enrichissante. Je remercie également mon encadrant au sein de l'entreprise, *Emmanuel Grandin*, pour son enthousiasme et son expérience, toujours de bon conseil.

Je remercie l'équipe de travail de la section Multimédia de l'entreprise STMicroelectronics pour m'avoir offert un cadre de travail chaleureux et amical. J'ai toujours pu compter sur l'aide de chacun pour mon travail et j'ai beaucoup appris à leurs côtés. Je remercie par ailleurs l'ensemble des employés de l'entreprise que j'ai pu rencontrer à l'occasion de cette thèse pour leur intérêt et leur amitié.

Je remercie le laboratoire du CREN et en particulier le site du Mans pour m'avoir permis de mener cette thèse à terme. Je remercie en particulier mon directeur de thèse *Pascal Leroux* pour son expérience et sa bienveillance.

Je remercie les membres de mon comité de suivi de thèse pour leur encadrement régulier au cours de ces années de travail. Je remercie en particulier *Nicolas Sabouret* pour m'avoir guidée lors de ma première conférence. Son amitié et ses judicieux conseils m'ont permis d'apprécier pleinement cette conférence et les suivantes.

Je remercie les membres de mon jury d'avoir accepté d'évaluer mon travail. Je remercie notamment *Sébastien Picault* et *Virginie Fresse* pour leur travail de rapporteurs sur ce manuscrit, ainsi que *Dominique Houzet* et *Leila Meghem-Boulahia* pour avoir accepté d'être examinateurs. Leur sagesse et leur expérience au service de l'évaluation de mes recherches m'honorent.

En dernier lieu, mais non des moindres, je remercie mes proches pour leur soutien et leur amitié. Je remercie particulièrement ma famille, mes parents, mes grands-parents, mes sœurs et l'ensemble de mes belle-familles pour leur tendresse, leur énergie et leur présence. Ils ont cru en moi sûrement plus que je n'ai pu le faire moi-même et leur confiance a représenté une source de motivation inépuisable. Je remercie en particulier mon compagnon *Nicolas Jaudronniere*, son amour et ses délicieux chocolats chauds furent des soutiens particulièrement efficaces.

Je remercie également mes amis, toujours présents pour les hauts comme pour les bas. Je remercie les anciens comme les nouveaux, notamment mes camarades doctorants. Les après-midis de travail et surtout des pauses marquées par un humour décapant resterons de précieux souvenirs.

Finalement, je termine cette aventure avec d'une part de précieuses rencontres et d'autres parts l'approfondissement d'anciens liens, sans qui ni ce travail ni moi-même ne serions les mêmes aujourd'hui. Tout ce que je peux dire à tous aujourd'hui, c'est un simple, mais profond, merci.

Table des matières

Liste des Figures

Liste des Tables

Chapitre 1

Introduction

"Un enfant, un professeur, un livre, un crayon peuvent changer le monde."

Malala Yousafzai
Femme politique(1997 -)

1.1 | Motivations

Je vous propose une expérience. Prenez au hasard une personne et demandez lui de définir la notion de Système Embarqué (SE). Il est peu probable que la personne puisse répondre avec précision. La notion, si elle lui est familière, restera floue. Pourtant les SE font partie intégrante de notre quotidien.

Depuis l'époque de la première satellisation autour de la lune en 1967 et de son système de guidage *Apollo Guidance Computer* considéré comme le premier système embarqué reconnaissable, ces ensembles complexes électroniques et informatiques se sont diffusés dans notre vie courante sous autant de formes et d'applications que l'imagination humaine l'a permis.

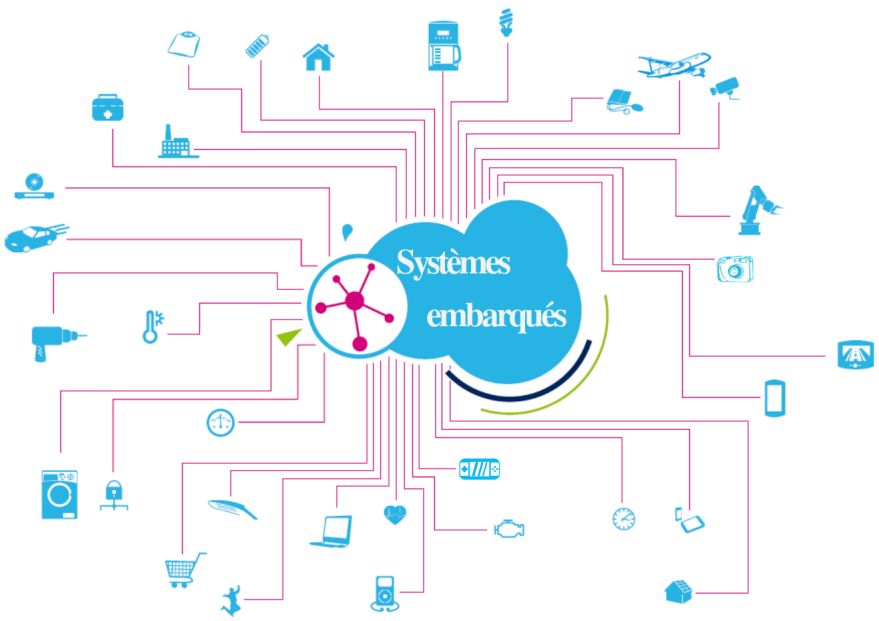


Figure 1.1 – *Exemples d'appareils intégrant un ou plusieurs systèmes embarqués.*

Développés pour assurer des tâches complexes, précises et inaccessibles à l'homme (ex : calculateur dans un avion, équipement médical, sonde spatiale), simplifier et sécuriser notre train de vie (ex : smartphones, four, distributeur

bancaire, assistance de conduite automobile, système d'air-bag) ou tout simplement permettre l'apparition de nouveaux loisirs (ex : console portable, appareil photo numérique, tablette, module d'accès Internet Set-Top box), les SE sont aujourd'hui partie intégrante de notre quotidien [Figure 1.1].

Une étude estimait, en 2008, l'utilisation en moyenne par personne d'environ 230 systèmes embarqués en l'espace d'une journée, en considérant la maison, le lieu de travail, les voitures et les appareils portables [Sifakis, 2011]. Aujourd'hui, soit 10 ans plus tard, ce nombre a explosé, notamment avec l'arrivée des objets connectés (*Internet of Things*, IoT).

Du robot-aspirateur à la liseuse, d'un ordinateur portable à une pompe à insuline, il peut être difficile d'établir une frontière claire définissant la notion de système embarqué. Tous, cependant, sont caractérisés par une limitation physique de leurs composants induite par leur intégration dans l'environnement des utilisateurs. Cette contrainte apporte des défis d'optimisation à résoudre, tels que la réduction de la taille des composants pour augmenter leur nombre dans un même espace, ou encore l'augmentation de la rentabilité des composants déjà présents par la gestion des ressources.

Par souci de fiabilité, les fabricants font souvent le choix de fixer, lors de la phase de conception, les tâches auxquelles les ressources seront dédiées. Par exemple, un processeur sera dédié à un encodage logiciel et un autre à l'affichage d'une vidéo. Cette solution prend en compte une marge de capacité afin d'éviter un blocage du système en cas de forte demande.

Le système embarqué ne peut alors pas faire appel à une autre ressource vacante dédiée à une tâche différente. L'appareil n'est pas programmé pour tenir compte des variations pouvant être induites par l'environnement ou des besoins spécifiques des utilisateurs et n'exploite donc pas tout son potentiel. Pourtant, certaines tâches peuvent être traitées par diverses unités. Par exemple dans les Set-Top Box, une vidéo peut aussi bien être décodée par programmation logicielle que par une unité matérielle spécialisée.

Ces systèmes possèdent le plus souvent une programmation matérielle prédéfinie a priori. Or, certains SE peuvent intégrer des unités reprogrammables permettant de modifier des fonctionnalités après la finalisation du produit. Cette possibilité s'accompagne néanmoins d'un retour en phase d'ingénierie. En terme d'adaptation directe à l'environnement utilisateur, ou à un contexte spécifique à un instant donné, les systèmes embarqués n'ont actuellement pas vraiment de solutions.

Nous soulevons donc un besoin d'adaptabilité des systèmes embarqués, à la fois au niveau de l'optimisation de la gestion des tâches attribuées aux ressources limitées et au niveau de leur réactivité pour la prise en compte du contexte lors de la réalisation des tâches.

Prenons le temps de revenir à la personne que vous aviez sollicitée au démarrage de ce mémoire et évoquons avec elle à présent la notion d'Intelligence Artificielle Distribuée (IAD). Elle sera sûrement plus apte à nous apporter son avis sur le vaste sujet qu'est l'Intelligence Artificielle (IA), inspirée par les nombreuses œuvres de fictions populaires. Ici, nous nous intéressons tout particulièrement à la notion de distribution, adaptable à notre contexte de multiples ressources matérielles. Une notion peut-être plus obscure pour la personne tout juste interrogée. Née d'une approche de résolution des activités simples ou complexes par les interactions et coopérations d'entités autonomes, cette discipline explore deux démarches principales :

- l'approche de l'IA à une architecture distribuée ;
- la répartition conceptuelle d'une IA sur différentes entités.

Dans le cadre des besoins des systèmes embarqués précédemment relevés, nous distinguons une nécessité d'interactions autonomes entre différentes ressources matérielles. La première approche permettrait de développer une entité intelligente répartie au niveau de ces ressources en se focalisant sur un contrôle centralisé. Nous ne hiérarchisons pas nos différentes ressources matérielles et ne souhaitons donc pas travailler ce type d'approche. Notre intérêt se porte donc sur la deuxième approche répondant plus à ce besoin de multiples entités autonomes, et notamment sur l'une de ses branches principales : les Systèmes Multi-Agents (SMA). Les entités agents composant ces derniers peuvent aussi bien correspondre à un humain qu'à une fourmi, une entité simulée ou encore un composant électronique. Grâce à leur nature distribuée et à travers leurs multiples interactions, les systèmes multi-agents évoluent en s'adaptant à leur environnement [Sabouret, 2009].

Les systèmes multi-agents couvrent de nombreuses thématiques de recherche comme la simulation de phénomènes complexes (sociologiques par exemple) ou sont appliqués directement dans l'industrie (jeux vidéos, animation, etc.) ou encore dans la finance (agents experts en trading, etc.). La Figure 1.2 illustre différents exemples de systèmes multi-agents existants. Ils peuvent ainsi simuler des modèles sociaux, permettre d'étudier la robotique avec l'expérimentation de comportements sur une partie ou l'ensemble de la machine, ou être adaptés à des problématiques de la vie courante, comme par exemple la gestion de l'énergie, pour déterminer le paramétrage le plus pertinent du

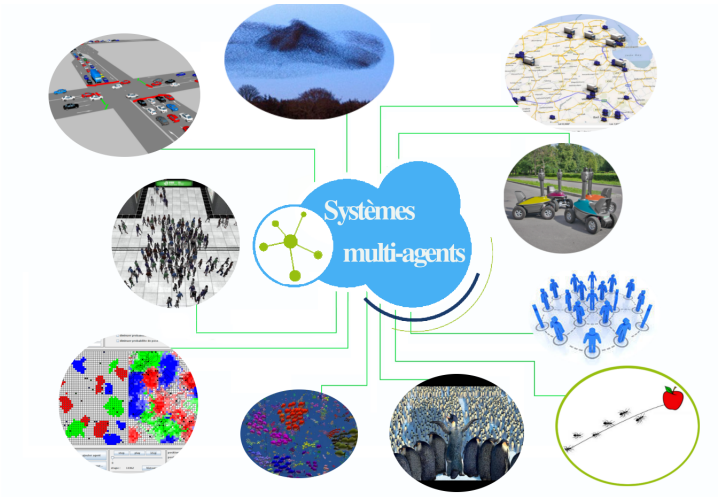


FIGURE 1.2 – *Différents exemples d'application des systèmes multi-agents.*

matériel.

Il est cependant encore rare de les voir directement intégrés au cœur des unités matérielles de façon autonome. Il reste donc de nombreuses études à mener sur les langages ou supports à appliquer dans le domaine des systèmes embarqués concernant la mise en place des interactions entre agents. Nous constatons notamment qu'il n'existe pas de proposition de plateforme agent embarquée pouvant s'adapter à un SE donné pour expérimenter l'intégration de modèles multi-agents au cœur du matériel. Cette constatation représente un objectif à remplir pour expérimenter les SMA en réponse aux besoins d'optimisation et d'adaptabilité des systèmes embarqués.

Avec la démocratisation des objets connectés et l'expansion de l'Internet des objets, ces thématiques représentent des enjeux actuels tout à fait pertinents.

1.2 | Positionnement des travaux

L'intégration des SMA à des environnements matériels embarqués est abordée dans plusieurs travaux couvrant de nombreux domaines dont nous souhaitons ici donner un aperçu. Nous présentons de manière la plus exhaustive possible des cas d'expérimentation de SMA en embarqué afin de positionner au mieux notre travail.

Gestion de l'énergie

Certains travaux de recherche appliquant les SMA dans le cadre de la régulation intelligente de l'énergie dans les bâtiments proposent leur implémentation au sein de systèmes embarqués. Ces applications se focalisent alors sur la fonctionnalité des SE concernés. Les recherches analysent les résultats de différents algorithmes agents dans le but d'établir une gestion de l'énergie plus intelligente. Les résultats sont majoritairement obtenus par le biais de simulations pour déterminer le paramétrage le plus pertinent du matériel concernant l'énergie consommée sur l'ensemble de la zone considérée. Ce paramétrage est ensuite intégré au niveau du matériel. Les agents n'interagissent pas directement avec ce dernier pour le modifier, ils sont également rarement placés au cœur même du matériel.

C'est par exemple le cas du système multi-agents SAVES (*Sustainable multi-Agents building application for optimizing Various objectives including Energy and Satisfaction*) où les agents restent au niveau d'applications relevant des données et simulant des possibilités pour baisser le niveau de consommation de l'énergie. Après l'établissement d'une solution, l'agent propose à l'utilisateur de l'appliquer s'il le souhaite [Kwak et al., 2012].

Dans le même contexte, le système multi-agents BLEMS (Building-Level Energy Management Systems) utilise des capteurs pour estimer la présence de personnes dans une pièce et proposer une adaptation de la consommation d'énergie en fonction de cette information [Mamidi et al., 2012].

La prédiction de l'activité et la présence humaine est également le cas d'étude de SMACH, une plateforme multi-agents [Gil-Quijano and Sabouret, 2010], permettant de simuler les déplacements d'une famille dans sa maison au quotidien. Ce projet est repris dans le cadre de la simulation des activités mêmes de la famille, simulant complètement le comportement d'un être humain [Amouroux et al., 2013]). Les résultats des simulations ont ici encore pour but d'être analysés afin de proposer des solutions pour réduire la consommation d'énergie en adéquation avec les habitudes de la famille.

Le projet MAHAS (*Multi-Agents Home Automation System*), destiné à réguler l'énergie dans le bâtiment, propose une intégration des SMA dans les systèmes embarqués à travers les appareils consommateurs et producteurs d'énergie [Abras et al., 2009]. Le projet utilise une méthode de conception du système multi-agents permettant d'allier SMA et SE sans aller jusqu'au bout de la mise en pratique : le projet est resté théorique [Abras, 2009]. Depuis ces travaux ont été repris avec un développement se concentrant sur l'analyse de simulations [Abras et al., 2013].

Nous pouvons ainsi observer que le domaine de recherche de gestion de l'énergie se concentre sur un résultat applicatif global et la simulation reste le principal outil. Nous souhaitons adapter les agents au cœur de l'embarqué et leur permettre d'interagir directement avec ce dernier. Nous pourrions ainsi exploiter l'autonomie leur permettant de réagir en temps-réel et de façon appropriée face à des situations imprévues ou nouvelles pour apporter de l'adaptabilité et de la réactivité à notre SE.

Maison intelligente

Les objets connectés représentent un marché en forte croissance : la notion d'objets intelligents correspond à une large demande. Ces objets incarnent des systèmes embarqués à l'origine d'un nouveau contexte commercial et industriel. En effet, une étude menée par la société Gartner estime un nombre d'objets connectés dans notre quotidien montant à 20 milliards pour l'horizon 2020 [Hung, 2017]. Les marchés couverts par les projets basés sur les IoT sont les environnements intelligents, les entreprises intelligentes, les vêtements intelligents, les maisons intelligentes et les cités intelligentes [Perera et al., 2015].

Ici le terme "objet intelligent" est utilisé pour désigner un objet capable de s'identifier, d'agir sur son environnement et de communiquer avec d'autres objets intelligents via des réseaux. Dans le cadre de l'Internet des objets, nous retrouvons essentiellement la capacité à communiquer des informations relevées en temps-réel pour un traitement distant. L'implémentation d'agents au cœur de ces objets pourrait permettre d'apporter des capacités d'interaction avec l'environnement en plus d'une possibilité d'adaptation ou d'apprentissage.

Le domaine de la "maison intelligente" correspond à l'intégration d'IoT dans l'environnement utilisateur personnel. Des recherches parlent de la possibilité d'intégrer des SMA au cœur d'unités matérielles dans ce but. Il s'agit actuellement d'approches théoriques. Leur objectif n'est actuellement pas d'être confrontées aux problématiques abordées par une réalisation concrète telles que les notions de bande passante et de contraintes mémoire ou temps réel pour les interactions des agents.

Par exemple, certains travaux proposent la modélisation d'un réseau de capteurs et d'acteurs multi-agents pour équiper une maison intelligente. Ces travaux relèvent ainsi l'utilité d'appliquer des SMA pour communiquer dans un environnement composé de machines très hétérogènes. Ils se concentrent sur la présentation d'un modèle agent BDI (*Belief Desire Intention*) et l'évaluation des performances de ce modèle dans le cadre de simulations [Sun et al., 2013]. La mise en pratique est brièvement présentée en perspective.

De la même manière, le projet TORRENT devait expérimenter un système multi-agents en tant que réseau intelligent de l'habitat, mais n'a a priori pas été poursuivi [Scharf et al., 2002].

La proposition d'une architecture de SMA adaptée à la domotique tout en prenant en compte le contexte matériel a également été proposée. Elle se focalise sur une formalisation de cette architecture et ne propose pas d'expérimentations post-implémentation [Miraoui et al., 2016].

Dans un contexte plus axé sur le multimédia de l'habitat, des travaux ont été effectués pour mettre en place un SMA pour réaliser un service de jeu multi-joueurs [Poggi, 2008]. L'idée est de pouvoir intégrer des agents aussi bien sur un ordinateur que sur un téléphone mobile ou une télévision interactive. Le projet s'en tient cependant à des agents n'ayant pas été intégrés au cœur même des différents appareils cités.

Sans spécifiquement nous restreindre au domaine de la domotique ou des IoT, nous souhaitons pour notre part une intégration concrète sur SE pour en étudier l'ensemble des contraintes. Nos travaux se positionnent ainsi en parallèle des travaux réalisés dans ce domaine.

Agents mobiles

Certains projets travaillent sur une approche d'agents mobiles, c'est à dire pouvant migrer d'un contexte d'exécution à un autre. Cette possibilité est utilisée en parallèle de plateformes de communications multi-agents : en effet cette approche autorise le déplacement de l'agent pour lui permettre d'obtenir les informations dont il a besoin ou pour transmettre des informations directement aux agents évoluant dans un autre contexte [Mayowa et al., 2016].

Un modèle agent spécifiquement adapté à ce type d'agent mobile a été proposé [Gherbi et al., 2013]. Cette approche, bien qu'utilisée dans le domaine de l'embarqué, se concentre sur l'évolution des agents et leur capacité de mobilité. Le matériel est considéré comme une ressource. De la même façon que pour les travaux liés à la gestion de l'énergie, les travaux ne se focalisent pas sur l'optimisation du système embarqué.

Exécution sur un processeur graphique

Certains travaux dans le domaine des systèmes multi-agents proposent d'adapter leur système à une application sur unité matérielle : l'unité processeur graphique (*Graphical Processing Unit*, GPU). Les recherches nécessitent d'adapter le langage de programmation pour être en adéquation avec l'architecture parallèle de ce composant. L'intégration du SMA sur GPU permet d'augmenter considérablement la puissance des simulations expérimentées en traitant un plus grand nombre d'agents simultanément ou d'augmenter la vitesse de calcul du résultat.

Par exemple, dans le domaine de la sécurité des personnes, l'expérimentation sur GPU de certaines recherches permet d'augmenter la complexité du cas étudié en simulation [Kleiner et al., 2013]. Des travaux ont également proposé une amélioration de la plateforme de simulation multi-agents TurtleKit3 afin qu'elle puisse intégrer le langage de programmation spécifique à cette unité : le GPGPU (*General-Purpose Computing on Graphics Processing Units*), pour faire des simulations à très grande échelle [Michel, 2013]. Un autre projet propose une modification de l'algorithme de programmation dynamique (*Improved Dynamic Programming*, IDP), utilisé dans le cadre de génération de coalition en simulation, pour le faire fonctionner sur GPU afin d'en améliorer les performances de calcul [Pawlowski et al., 2014].

Ici le but est donc d'appliquer ce composant embarqué comme ressource pour améliorer des performances de SMA dans le cadre de simulations. Dans notre cas, nous nous intéressons à la possibilité d'expérimenter des SMA au cœur de systèmes embarqués pour optimiser l'utilisation des ressources de ces derniers.

Robotique

En robotique, une partie ou l'ensemble de la machine peut être considérée comme un agent auquel sont appliqués des comportements sociaux. Cet agent dispose ainsi des acteurs et capteurs du robot pour interagir avec son environnement et les autres agents-robots. Par exemple, une distribution des tâches entre les différents robots peut ainsi s'organiser [Liekna et al., 2012].

Dans ce contexte, des recherches ont proposé une approche de développement générique pour des agent-robots [Cossentino et al., 2003]. Ces travaux ont évolué vers la présentation d'une structure logicielle dédiée au développement de SMA appliqués à la robotique [Iñigo-Blasco et al., 2012]. Un modèle de formalisation de système multi agent-robots fut soumis [García et al., 2012]. Plus récemment, certains travaux se sont focalisés sur la proposition d'une plateforme dédiée aux agent-robots pour permettre de faire le lien entre des SMA logiciels et leur support embarqué dans le cadre de la robotique [Lazarin and Pantoja, 2015]. Ces travaux sont poursuivis avec la présentation d'une architecture agent, ARGO, pour la programmation d'agent-robots embarqués. Elle relie les capteurs et acteurs de l'agent aux informations de son environnement matériel tout en conservant une base de raisonnement de type BDI [Pantoja et al., 2016].

A travers tous ces travaux, nous observons la prise en compte des contraintes matérielles sur les systèmes multi-agents. Cependant nous constatons la focalisation de ces projets sur l'ensemble du système, pour lui donner des comportements sociaux ou coordonner plusieurs pièces entre elles par le biais des protocoles d'interaction multi-agents. A la différence de ces travaux, nous souhaitons expérimenter nos agents en tant que processus au cœur même de nos unités matérielles. Notre perspective de recherche repose sur l'application de ces protocoles d'interaction entre les ressources à l'échelle de nos composants, amenant des contraintes embarquées strictes aux systèmes multi-agents.

1.3 | Définition de la problématique

"Les systèmes embarqués recherchent une utilisation optimale des ressources en assurant des niveaux acceptables de qualité de service" [Sifakis, 2011]. Nous avons relevé, d'une part, un problème de manque d'adaptabilité des systèmes embarqués ainsi qu'un défi d'optimisation de la gestion des tâches attribuées à leurs ressources et, d'autre part, le manque de travaux apportant des propositions de déploiement des systèmes multi-agents au cœur d'unités matérielles. Nous considérons la possibilité de joindre ces deux domaines d'études en les liant au travers de la problématique globale suivante :

Problématique

Des modèles multi-agents intégrés au cœur d'un système embarqué peuvent-ils apporter des capacités d'adaptation dynamique au contexte d'utilisation et à son environnement ?

Dans le cadre des SE, nous ciblons à travers cette problématique la possibilité d'employer des algorithmes d'interactions agents pour corriger les besoins suivants :
- optimiser la gestion des tâches assignées aux ressources matérielles de façon réactive ;
- apporter de l'adaptabilité au système embarqué lors de la réalisation d'une tâche pour prendre en compte le contexte d'exécution.

Dans le cadre des SMA, nous proposons ainsi de travailler sur la formalisation d'agents embarqués pour répondre aux problématiques des SE. Nous avons observé lors du positionnement de nos travaux que notre approche de déploiement de systèmes multi-agents au sein d'unités matérielles existantes n'avait pas encore été abordée. Nous nous concentrons donc sur les points spécifiques de notre problématique suivants :
- formaliser des agents embarqués ;
- formaliser l'architecture d'un SMA embarqué ;
- élaborer une plateforme multi-agents embarquée pouvant s'intégrer dans un SE.

Pour répondre à cette problématique globale et à ses besoins spécifiques à chaque domaine, nous établissons des choix de réalisation afin d'affiner notre approche.

1.4 | Contexte de la recherche

Ce travail s'inscrit dans l'action de recherche (AR) Tifaifai développée au sein du Centre de Recherche en Éducation de Nantes (CREN), site du Mans, depuis septembre 2012. La problématique générale de cette AR est la conception de nouveaux espaces d'interactions adaptatifs, mobiles et connectés entre eux dans des contextes éducatifs. Les méthodes et architectures informatiques sont indépendantes des contextes d'application. Nos contributions doivent donc assurer une transposabilité des travaux pour renforcer le caractère générique de la démarche informatique. A partir de nos propositions, les recherches pourront donc se poursuivre en ciblant des contextes applicatifs plus précis, tels que la sécurité ou l'éducation, qui apporteront de nouveaux défis et de nouvelles problématiques associées.

L'entreprise STMicroelectronics fait partie des leaders mondiaux dans le domaine du semi-conducteur et produit notamment des systèmes embarqués de type Set-Top Box. Traditionnellement, ces Box n'étaient utilisées que pour afficher des retransmissions télévisuelles (type décodeur satellite, décodeurs des chaînes payantes, etc.). Aujourd'hui, elles sont en train de devenir des objets multimédia plus complexes se connectant aux smartphones, à Internet, servant de serveur multimédia ou de stockage de données. Face à cette augmentation du nombre de fonctionnalités à remplir, ce SE doit faire face à la problématique d'optimisation de ses ressources pour continuer de satisfaire le client aussi bien en qualité de résultat qu'en termes de réduction de sa taille pour une meilleure intégration à l'environnement de l'utilisateur.

Nous souhaitons assurer la généricité de nos recherches en proposant des solutions les plus indépendantes possibles du système embarqué considéré. Nous travaillons donc sur des systèmes embarqués existants. Par ailleurs, ces

Set-Top Box sont actuellement présentes dans bon nombre de foyer. Elles représentent donc un système idéal pour expérimenter nos propositions.

Ces travaux de recherche sont réalisés dans le cadre d'une thèse CIFRE en partenariat avec l'entreprise ST-Microelectronics et le CREN. Cette collaboration nous permet de nous appuyer sur une dynamique "Recherche et Innovation" en restant en contact direct avec les ingénieurs développant ce matériel. Nous avons ainsi pu échanger sur les besoins du projet et leur faire des retours d'usages sur les cartes. Les Set-Top Box sont des systèmes embarqués principalement dédiés à des traitements multimédia. Nos expérimentations seront donc également liées à ce domaine, spécifiquement au traitement d'images. Exigeant en terme de temps de réponse et gourmand en terme d'utilisation de ressources, ce domaine d'étude nous permet d'évaluer nos propositions sur le SE.

1.5 | Hypothèses et contributions visées

Dans le cadre de ce travail de thèse, nous proposons d'établir les bases d'une approche multi-agents au sein des systèmes embarqués. À ce titre, nous souhaitons apporter des contributions théoriques et techniques de formalisation du déploiement d'agents embarqués.

Pour assurer la généricité de nos travaux et leur adaptation à divers systèmes embarqués voire à des ensembles de cartes embarquées, nous souhaitons pouvoir ajouter ou retirer des agents embarqués au SMA à tout instant en nous focalisant sur les apports liés à leur comportement. Nous proposons donc de centrer notre formalisation autour des interactions du SMA pour ne pas fixer les entités agents le composant. Dans ce but, nous nous inspirons de modèles agents eux-mêmes centrés sur les interactions. Concernant ces dernières, nous regardons en particulier les notions de collaboration et de coopération pouvant émerger. Nous concentrons notre travail sur l'apport des comportements des agents sur le système embarqué indépendamment de leur réalisation.

Hypothèse #1

L'approche centrée interaction de la formalisation de nos agents embarqués nous permettra de conserver une forme de généricité dans notre proposition.

Cette première hypothèse guide notre proposition dans le cadre du déploiement des agents. Nous relevons également un besoin de gestion de ces agents. La proposition d'une plateforme multi-agents adaptée à un contexte embarqué permet d'apporter un contexte de communication reliant les agents entre eux et aux autres processus du SE. Elle permet également de proposer des bibliothèques de fonctions intégrées facilitant le déploiement des agents embarqués. Dans le cadre du partenariat avec STMicroelectronics, la société a émis le souhait de pouvoir déployer nos agents dans un contexte matériel hétérogène, voire de pouvoir relier ce projet à d'autres travaux de recherche d'intelligence artificielle distribuée.
Nous proposons donc de normaliser les échanges et la gestion administrative de cette plateforme multi-agents suivant les standards proposés dans le domaine.

Hypothèse #2

Le suivi des standards d'administration et d'architecture multi-agents nous permettra de proposer de nouveaux modèles de communication pour une plateforme embarquée.

Dans le cadre de l'optimisation de la gestion des ressources embarquées, l'ordonnancement effectue une première allocation selon les ressources fixées lors de la conception. Nous souhaitons apporter une solution indépendante des composants constituant le système embarqué. Nous devons donc prendre en considération des unités matérielles potentiellement non reprogrammables. Par conséquent, nous proposons de faire intervenir nos agents embarqués en correction des choix effectués par l'ordonnanceur en considérant les ressources de calculs disponibles et le contexte

d'exécution du processus. Nous travaillons donc à la modification les allocations pré-définies en déléguant les tâches à d'autres ressources éligibles par des algorithmes dynamiques multi-agents de type négociation.

Pour ce travail de thèse, s'appuyant sur des bases de formalisation des agents embarqués, nous nous concentrons sur la délégation des tâches entre différents processus. Pour apporter une optimisation de la gestion des ressources d'un SE sans lier directement nos agents à des unités matérielles, nous rendons possible cette délégation des tâches entre différents systèmes embarqués.

Hypothèse #3

Les processus pouvant négocier une délégation des tâches allouées, par le biais d'interactions multi-agents entre différents systèmes embarqués, optimiseront la gestion des tâches de chaque SE de façon réactive.

Par ailleurs, cette proposition nous vérifierons la généricité de nos contributions en les expérimentant sur un ensemble de systèmes embarqués hétérogènes.

En association avec cette optimisation, nous proposons de lier les agents embarqués à la réalisation de certaines tâches pour apporter une prise en compte du contexte d'exécution dans le traitement de cette tâche. Les interactions entre agents et avec leur environnement augmentent le nombre de choix de traitement possible. De plus, la distribution du travail entre des entités agents permet d'ajouter dynamiquement diverses compétences.
Cette éventualité augmente la flexibilité du système embarqué, c'est-à-dire une plus grande capacité d'adaptation en fonction des différents paramètres de décision.

Hypothèse #4

La réalisation d'une tâche partagée entre plusieurs processus agents nous apportera de la flexibilité via différents choix d'exécution et l'adaptation au contexte d'exécution.

A travers ces deux dernières hypothèses spécifiques aux besoins de l'embarqué, basées sur les deux premières hypothèses liées aux SMA embarqués, nous ajoutons des capacités d'adaptation dynamique au contexte d'utilisation du SE, répondant ainsi à notre problématique.

1.6 | Structure du mémoire

Ce manuscrit de thèse se compose de trois parties principales, telles que résumées en Figure 1.3. Lors de cette introduction, nous avons présenté les problématiques liées aux deux domaines principaux de cette thèse ainsi que la problématique de recherche globale.

Dans la **Partie I**, nous détaillons les spécificités propres à chaque domaine. Nous présentons tout d'abord les systèmes embarqués en précisant les principaux défis de recherche régissant ce domaine. Nous évoquons ainsi les notions de fiabilité, de sécurité, de réactivité et d'optimisation des SE. Nous précisons en particulier ce dernier point impacté directement par notre problématique. Nous détaillons ensuite les notions de ressources et de tâches au cœur de nos hypothèses. Les moyens de communication seront également abordés pour expliciter nos choix relatifs à notre plateforme multi-agents embarquée. Enfin, nous abordons les méthodes d'évaluation des travaux usuelles du domaine qui seront ré-employées lors de nos expérimentations.
Nous développons ensuite les spécificités des systèmes multi-agents en revenant sur les différentes formalisations existantes du point de vue de l'agent, de son organisation en système selon des modèles donnés et du point de vue des plateformes multi-agents offrant un environnement de déploiement, de gestion et d'interaction. Nous exposons alors les standards de gestion de la communauté multi-agents pour l'administration et le transport des messages afin de détailler nos choix relatifs à notre plateforme multi-agents embarquée. Nous présentons ensuite les différents algorithmes d'interactions multi-agents dont les protocoles de négociation que nous appliquerons par la suite pour la

délégation de tâches dans le système embarqué. Enfin, nous évoquons également les méthodes usuelles d'évaluation du domaine.

Dans la **Partie II**, nous précisons nos hypothèses. Dans un premier temps, nous détaillons notre formalisation du déploiement des agents embarqués en précisant leur cadre d'évolution et leurs spécificités. Les étapes de réalisation d'un SMA embarqué sont inspirées de modèles agents présentés dans la partie précédente. Dans un deuxième chapitre, nous développons notre proposition d'une plateforme : MERMAID (*Managed Embedded Resources by Multi-Agents applied to Integrated and Distributed systems*). Afin de réaliser cette dernière, nous détaillons nos choix concernant son architecture, notamment la gestion de la communication, son administration, avec l'application des normes sélectionnées et son organisation.

La **Partie III** reprend ces contributions et propose des expérimentations dans le cadre applicatif de traitement d'images. Le SMA est réalisé en suivant les étapes formalisées en partie II. L'expérimentation est conduite grâce à une mise en application au sein du SE sur la plateforme MERMAID. Nous effectuons des évaluations de performance et des analyses visuelles pour mettre à l'épreuve notre proposition. Ensuite, le principe de délégation de tâches est expérimenté avec un SMA intégrant des protocoles de négociation. Nous expérimentons d'abord le principe de délégation seul. Nous reprenons ensuite notre évaluation liée à la flexibilité pour la réaliser avec cet apport de la délégation des tâches. Enfin nous éprouvons cet ensemble en permettant une délégation entre différentes cartes embarquées dans un contexte hétérogène.

A travers la **Conclusion**, nous récapitulons nos contributions et analysons nos résultats vis-à-vis de nos hypothèses et de notre problématique. Nous ouvrons ensuite les perspectives de ce travail tant au niveau de l'approfondissement de nos travaux qu'au niveau de leur extension à d'autres domaines d'application.

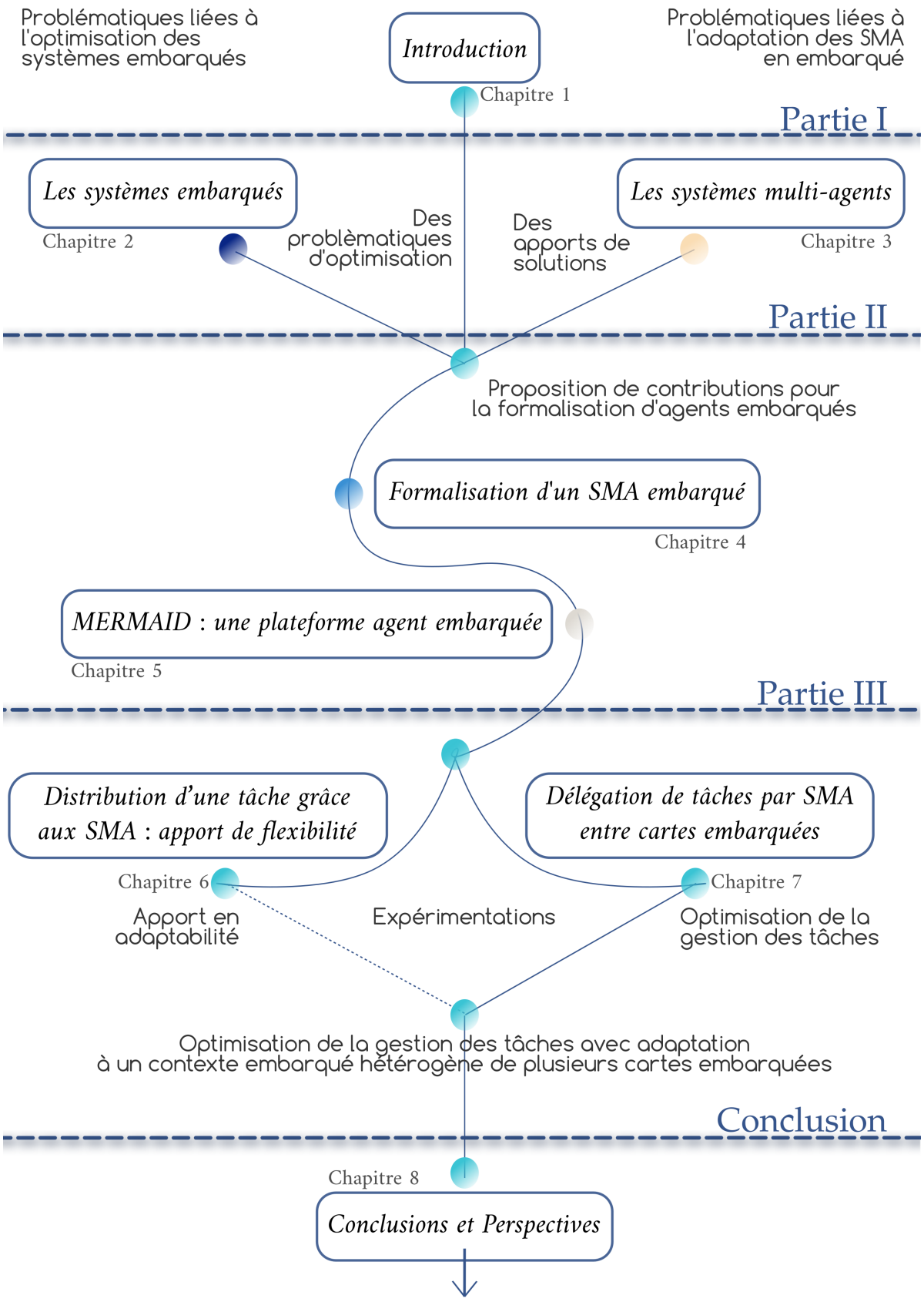


Figure 1.3 – *Organisation du manuscrit de thèse.*

Bibliographie

[Abras, 2009] Abras, S. (2009). *Système domotique multi-agents pour la gestion de l'énergie dans l'habitat*. PhD thesis, Institut Polytechnique de Grenoble, Laboratoire d'Informatique de Grenoble, IMAG.

[Abras et al., 2013] Abras, S., Kieny, C., Ploix, S., and Wurtz, F. (2013). Mas architecture for energy management : Developing smart networks with jade platform. In *International Conference on Smart Instrumentation, Measurement and Applications (ICSIMA)*, Kuala Lumpur, Malaysia. IEEE Computer Society Press.

[Abras et al., 2009] Abras, S., Ploix, S., Pesty, S., and Jacomino, M. (2009). Apport d'une approche multi-agents pour la résolution d'un probleme de gestion de l'energie dans l'habitat. In *17èmes Journées francophones sur les systèmes multi-agents (JFSMA)*, pages 110–116, Lyon, France. Cépaduès Édition.

[Amouroux et al., 2013] Amouroux, E., Huraux, T., Sempé, F., Sabouret, N., and Haradji, Y. (2013). Simulating human activities to investigate household energy consumption. In *5th International Conference on Agents and Artificial Intelligence (ICAART)*, pages 71–80. Springer.

[Cossentino et al., 2003] Cossentino, M., Sabatucci, L., and Chella, A. (2003). A possible approach to the development of robotic multi-agent systems. In *IEEE/WIC International Conference on Intelligent Agent Technology (IAT)*, pages 539–548. IEEE Computer Society Press.

[García et al., 2012] García, C., Cárdenas, P. F., Puglisi, L. J., and Saltaren, R. (2012). Design and modeling of the multi-agent robotic system : Smart. *Robotics and Autonomous Systems*, 60(2) :143–153.

[Gherbi et al., 2013] Gherbi, T., Borne, I., and Meslati, D. (2013). An mde approach to develop mobile-agents applications. *Communications in Computer and Information Science*, 417 :64–80.

[Gil-Quijano and Sabouret, 2010] Gil-Quijano, J. and Sabouret, N. (2010). Prediction of humans' activity for learning the behaviors of electrical appliances in an intelligent ambiant environment. In *IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT)*, page 283–286. IEEE Computer Society Press.

[Hung, 2017] Hung, M. (2017). Leading the iot, gartner insights on how to lead in a connected world. Technical report, Gartner.

[Iñigo-Blasco et al., 2012] Iñigo-Blasco, P., del Rio, F., Romero-Ternero, M., DanielCagigas-Muñiz, and SaturninoVicente-Diaz (2012). Robotics software frameworks for multi-agent robotic systems development. *Robotics and Autonomous Systems*, 60(6) :803–821.

[Kleiner et al., 2013] Kleiner, A., Farinelli, A., Ramchurn, S., Shi, B., Mafioletti, F., and Refatto, R. (2013). Rmasbench : benchmarking dynamic multi-agent coordination in urban search and rescue (extended abstract). In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1195–1196. International Foundation for Autonomous Agents and Multiagent Systems.

[Kwak et al., 2012] Kwak, J., Varakantham, P., Maheswaran, R., Tambe, M., Jazizadeh, F., Kavulya, G., Klein, L., Becerik-Gerber, B., Hayes, T., and Wood, W. (2012). Saves : A sustainable multiagent application to conserve building energy considering occupants. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, volume 1, pages 21–28. International Foundation for Autonomous Agents and Multiagent Systems.

[Lazarin and Pantoja, 2015] Lazarin, N. M. and Pantoja, C. E. (2015). A robotic-agent platform for embedding software agents using raspberry pi and arduino boards. In *Software Agents, Environments and Applications School (WESAAC)*, pages 13–20. UFAL, UFF, IME.

[Liekna et al., 2012] Liekna, A., Lavendelis, E., and Grabovskis, A. (2012). Experimental analysis of contract net protocol in multi-robot task allocation. *Applied Computer Systems*, 13(1) :6–13.

[Mamidi et al., 2012] Mamidi, S., Chang, Y.-H., and Maheswaran, R. (2012). Improving building energy efficiency with a network of sensing, learning and prediction agents. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, volume 1, pages 45–52, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

[Mayowa et al., 2016] Mayowa, O., M, F. T., Olabiyisi, S. O., Omidiora, E., and Fawole, A. (2016). A survey on migration process of mobile agent. *World Congress on Engineering and Computer Science (WCECS)*, 1 :415–419.

[Michel, 2013] Michel, F. (2013). Intégration du calcul sur gpu dans la plate-forme de simulation multi-agent générique turtlekit 3. In *Journées francophones sur les systèmes multi-agents (JFSMA)*, pages 189–198. Cépaduès Édition.

[Miraoui et al., 2016] Miraoui, M., El-etriby, S., Abid, A. Z., and Tadj, C. (2016). Agent-based context-aware architecture for a smart living room. *International Journal of Smart Home*, 10(5) :39–54.

[Pantoja et al., 2016] Pantoja, C. E., Junior, M. F. S., Lazarin, N. M., and Sichman, J. S. (2016). Argo : A customized jason architecture for programming embedded robotic agents. In *4th International Workshop on Engineering Multi Agent Systems (EMAS 2016)*, pages 133–148. Springer.

[Pawlowski et al., 2014] Pawlowski, K., Kurach, K., Svensson, K., Ramchurn, S., Michalak, T. P., and Rahwan, T. (2014). Coalition structure generation with the graphics processing unit. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 293–300, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

[Perera et al., 2015] Perera, C., Liu, C. H., and Jayawardena, S. (2015). The emerging internet of things marketplace from an industrial perspective : A survey. *IEEE Transactions on Emerging Topics in Computing*, 3(4) :585–598.

[Poggi, 2008] Poggi, A. (2008). Developing multi-user online games with agents. *Journal WSEAS Transactions on Computers*, 7(8) :1240–1249.

[Sabouret, 2009] Sabouret, N. (2009). *Interactions about Actions in Open and Heterogenous Multi-Agent Systems*. Habilitation à diriger des recherches, Université Pierre et Marie Curie - Paris VI.

[Scharf et al., 2002] Scharf, E., Hamer, P., Smparounis, K., Payer, W., Ronan, J., and Crotty, M. (2002). An intelligent integrated approach to multi-service residential access networks. In *41st FITCE European Telecommunications Congress*, pages 1–6. FITCE.

[Sifakis, 2011] Sifakis, J. (2011). A vision for computer science — the system perspective. *Central European Journal of Computer Science*, 1 :108–116.

[Sun et al., 2013] Sun, Q., Yu, W., Kochurov, N., Hao, Q., and Hu, F. (2013). A multi-agent-based intelligent sensor and actuator network design for smart house and home automation. *Journal of Sensor and Actuator Networks*, 2(3) :557–588.

Partie I
État de l'art

En introduction, nous avons détaillé notre problématique de travail ainsi que nos hypothèses et propositions. Nous revenons dans cette partie sur les concepts et définitions déterminant nos deux domaines d'étude. Nous abordons donc cette partie en deux chapitres.

Le premier expose le domaine des systèmes embarqués. Nous présentons les contraintes qui caractérisent ces systèmes et les problématiques qu'elles apportent. Nous considérons chacune de ces contraintes tout en nous focalisant sur la thématique d'optimisation des SE. Comme nous travaillons sur des systèmes embarqués avec une composition matérielle fixée, nous n'abordons pas les modèles de développement et nous nous concentrons plutôt sur les méthodologies du domaine concernant la gestion des tâches et les algorithmes de distribution utilisés. La communication étant une notion importante concernant les SMA, les méthodes d'interaction employées dans l'embarqué sont présentées. Nous exposons enfin les procédures d'évaluation dans ce domaine.

Notre deuxième partie est dédiée aux systèmes multi-agents. Nous décrivons les systèmes multi-agents eux-mêmes, en détaillant la notion de comportement agent, et offrons un aperçu des modèles et des plateformes existantes. Nous présentons ensuite les standards du domaine en nous focalisant particulièrement sur les apports de la fondation FIPA. Nous expliquons enfin les notions de comportements sociaux et détaillons spécifiquement les algorithmes de négociation multi-agents que nous souhaitons utiliser pour optimiser les systèmes embarqués. Comme pour les SE, nous présentons enfin les méthodes d'évaluation appliquées dans ce domaine.

Chapitre 2

Les systèmes embarqués

"Dans la vie, rien n'est à craindre, tout est à comprendre."

Marie Curie
Physicienne, Scientifique (1867 – 1934)

2.1 | Défis de l'embarqué

Les systèmes embarqués sont des ensembles complexes, électroniques et informatiques, confinés et mobiles, aux applications très variées [Etiemble, 2016]. Ils sont décrits par le terme "systèmes enfouis" (*ubiquitous systems*) car ils représentent la majorité des processeurs utilisés dans le monde, tout en restant invisibles pour l'utilisateur [Want et al., 2002]. Là où un ordinateur standard est qualifié de "généraliste" capable d'exécuter tout type d'application, un système embarqué est dédié à une application précise.

Les définitions du terme système embarqué évoluent en fonction des travaux considérés. Tout en restant relativement vagues, nous pouvons constater qu'elles s'appuient sur certaines caractéristiques jugées pertinentes pour la suite de nos travaux. Ainsi, Laurgeau, Steux et Metman définissent les systèmes embarqués comme "*tout système informatique qui est appelé à fonctionner sur une plate-forme mobile (véhicule terrestre, voiture, camion, engin agricole, militaire etc.)*" [Laurgeau et al., 2002]. A partir de cette affirmation, ils précisent que leurs spécificités reposent sur "*des contraintes d'autonomie énergétique, la résistance aux vibrations et le cas échéant à d'autres problèmes liés à la masse ou au volume transporté, à la dissipation thermique, aux contraintes électromagnétiques*". Dans cette définition, nous pouvons relever la diversité des SE alliée à de nombreuses contraintes de réalisation. Dans les travaux de Sifakis et Henzinger un système embarqué est défini comme "*un artefact d'ingénierie nécessitant du calcul et étant sujet à des contraintes physiques*" [Sifakis and Henzinger, 2007]. Ils spécifient par ailleurs : "*les contraintes physiques interviennent au travers de deux types d'interaction entre le processus informatique et le monde physique : réaction à un environnement physique et exécution sur une plateforme physique*". Dans cette vision, nous relevons l'importance de l'interaction entre le système embarqué et son environnement. Vient s'y ajouter la notion de contraintes physiques déjà soulevée dans les défis de réalisation énoncés dans la précédente définition. Cette notion de contrainte est présente et plus détaillée dans les travaux de Sangiovanni-Vincentelli déterminant un système embarqué comme "*un système dédié dans lequel l'unité de calcul est complètement contenue dans un système qu'elle contrôle*" [Sangiovanni-Vincentelli, 2007]. Il précise que le système embarqué "*interagit avec l'environnement qui l'entoure d'une manière contrôlée satisfaisant un ensemble de contraintes de réactivité en termes de qualité et de rapidité*". Nous relevons ici que les SE apportent également des contraintes de résultat et de fonctionnement.

Dans ces trois définitions, nous retrouvons l'idée d'intégration à l'environnement, d'autonomie et de réalisation de tâches spécifiques. Nous relevons également la caractérisation de ces systèmes par des contraintes physiques importantes. Cependant, vis-à-vis de notre travail, nous souhaitons souligner l'impact de ces limites physiques sur les ressources disponibles pour la réalisation des tâches du SE. Aussi, nous proposons notre définition d'un système embarqué regroupant l'ensemble de ces spécificités [Définition 1].

Définition #1 : *Système embarqué*

Un **système embarqué** est un système intégré dans l'environnement utilisateur et équipé d'une carte électronique dédiée à la réalisation de tâches spécifiques. Autonome, il est soumis à des contraintes fortes de restriction de ses ressources directement induites par ses limitations physiques.

La notion d'intégration à l'environnement utilisateur souligne qu'un SE n'est souvent qu'une part d'un ensemble plus important. Le système physique dans lequel le système embarqué est intégré va apporter plusieurs contraintes à la réalisation de ce dernier. Il est également primordial de les considérer lors de la phase de programmation du SE [Habiba et al., 2014]. Nous regroupons ces contraintes selon quatre catégories représentant les défis majeurs du domaine : la fiabilité, la sécurité, la réactivité et l'optimisation du système.

2.1.1 Fiabilité du système

L'intégration du SE à un ensemble physique le rend difficilement accessible pour une réparation ou une mise à jour logicielle. Par ailleurs, un même SE peut être confronté à différents environnements d'utilisation impliquant une variation de la température externe, des vibrations, des chocs, différents niveaux d'humidité, etc. Enfin, la connaissance des utilisateurs, parfois inexistante, peut entraîner des erreurs d'utilisation. Ces constatations impliquent un besoin de résistance de l'appareil, aussi bien au niveau des composants qu'à travers l'anticipation des comportements pour éviter un dysfonctionnement. Ce besoin est accentué par la fonction de certains SE, directement liés à la sécurité de leurs utilisateurs (matériel médical, déclenchement d'un airbag, etc.).

Nous résumons ces contraintes par un besoin de fiabilité du système embarqué dans le cadre du stockage des données [Chao and Fang, 2012], des communications générées [Aza-Vallina et al., 2011] ou de l'ensemble matériel [Narayanan and Xie, 2006]. Il existe deux manières d'augmenter cette fiabilité : en travaillant sur la robustesse ou la redondance du système. Les systèmes les plus fiables exploitent les deux possibilités.

La **robustesse** est un indicateur du niveau de stabilité de la performance d'un système vis-à-vis de son contexte d'utilisation. Elle prend en compte des notions de résistance des composants, par exemple à la pression atmosphérique, à la température, ou encore à une durée prolongée d'utilisation [Midonnet et al., 2010]. La robustesse mesure également la capacité du SE à éviter les dysfonctionnements induits par une utilisation erronée de l'appareil. Un système robuste fonctionne pour de larges contextes d'utilisation et comporte des programmes de relance en cas d'erreur.

La **redondance** est un principe de répétition permettant de limiter les risques d'arrêt du système ou de vérifier des résultats critiques. Le principe de redondance est respecté uniquement si plusieurs objets, identiques ou différents, exercent les mêmes fonctions indépendamment les uns des autres. Il existe trois formes de redondance : matérielle, logicielle et des données.

- La redondance matérielle :
  Elle consiste en une duplication d'une portion ou de la totalité d'un composant. Il est possible d'utiliser une réplique parfaitement identique. Ce type de redondance permet d'assurer le fonctionnement du système dans le cadre d'une défaillance du premier composant. Les éléments dupliqués sont alors interchangeables. Il est également possible de dupliquer l'élément avec un composant différent pour augmenter la robustesse du système. Dans ce cadre, l'élément matériel effectue toujours le même travail, mais n'est pas constitué de la même façon. En effet, nous diminuons ainsi le risque d'une défaillance simultanée due à un même phénomène. Par exemple, si le matériel d'origine rencontrait un dysfonctionnement du fait d'une hausse de température inhabituelle, le composant dupliqué identique ferait face au même problème. Une réplique différente ne serait, elle, pas sujette à la même défaillance et pourrait assurer le fonctionnement du système.

- La redondance logicielle :
  Elle correspond à la réalisation simultanée de la même opération par plusieurs unités du système. Une unité de contrôle va ensuite comparer les résultats obtenus pour déterminer, selon sa programmation, le plus valable. Par exemple, elle peut sélectionner le résultat majoritaire. Il est également possible de dupliquer l'unité de contrôle pour assurer l'interprétation des résultats.

- La redondance des données : Elle implique l'enregistrement de données identiques dans plusieurs blocs mémoires différents. Cette pratique permet de diminuer le risque de perte de données sensibles en cas de défaillance d'un bloc. Une unité de contrôle peut également être assignée à ces données et les comparer pour détecter d'éventuelles corruptions des informations. Selon la gravité de l'erreur, elle pourra être en capacité de corriger les données grâce aux duplicatas.

Dans tous les cas, le principe de redondance peut théoriquement être appliqué à l'infini. Dans les faits, la duplication est effectuée jusqu'à l'obtention d'un équilibre entre un niveau de fiabilité satisfaisant et les coûts supplémentaires générés par cette méthode [Meedeniya et al., 2010]. Nous prenons garde à cet aspect lors de nos propositions d'agents embarqués. Nous expérimentons également leur apport aux SE vis-à-vis de ce besoin de fiabilité.

2.1.2 Sécurité du système

Beaucoup de systèmes embarqués peuvent être amenés à traiter des données sensibles (données personnelles bancaires, contenus multimédias propriétaires, mots de passe, etc.) ou à offrir des fonctionnalités de contrôle sur des appareils sensibles (porte de garage d'un particulier, système de surveillance d'une prison, alarme intrusion d'un magasin, pilote automatique d'une voiture, etc.). Pour des raisons d'intégrité, il est donc primordial d'assurer qu'une personne mal-intentionnée ne puisse pas s'introduire dans le système pour récupérer des données ou corrompre son utilisation.

Nous résumons ces contraintes par un besoin de sécurité du système embarqué. Les solutions employées peuvent utiliser le cryptage des données à sécuriser, le blocage de l'accès à ses données par authentification de l'utilisateur, l'adaptation de l'architecture du SE [Kocher et al., 2004], la fermeture en partie ou totale du composant pour créer une zone fiable qualifiée de *TrustZone* [Benhani et al., 2017] ou des programmes pour détecter un comportement inhabituel du système [Balaji et al., 2017]. Ces solutions reposent sur une évaluation du type d'attaque possible pour en bloquer les failles.

Ces aspects sécuritaires ne sont pas traités dans cette thèse. Nous prenons cependant garde à ce besoin pour nos choix d'architectures afin de pouvoir intégrer des solutions dans de futurs travaux.

2.1.3 Réactivité du système

Certains SE doivent réagir immédiatement aux événements extérieurs. Ils sont qualifiés de systèmes embarqués temps-réel. L'exécution de chaque tâche donnée dans ces systèmes doit se terminer avant une date limite, qualifiée d'échéance, au-delà de laquelle les résultats ne seront plus considérés comme valides. Le temps d'exécution du programme est donc pris en compte pour valider le résultat obtenu dans une fenêtre de temps donnée. Par exemple, ce besoin est relevé dans le cadre de l'automatique industrielle [Rutagangibwa and Krishnamurthy, 2014] ou de la détection visuelle [Wang et al., 2014]. La qualification "temps-réel" se spécifie en deux catégories :

- Temps-réel dur/strict :
  Cette catégorie s'applique pour les cas d'utilisation où le non-respect de l'échéance entraîne une conséquence critique pour l'environnement d'utilisation (destruction) ou l'utilisateur (blessure, mort). La fenêtre de temps considérée est alors de l'ordre de la nanoseconde.
  Exemples : appareil médical, système de freinage, système de détection de problèmes lors du lancement d'une fusée, etc.

- Temps-réel souple :
  Cette appellation est employée si le dépassement de l'échéance n'entraîne pas un désagrément critique mais simplement une baisse de la fiabilité de l'appareil. Le respect de l'échéance est toujours primordial pour le bon fonctionnement du système, mais son non-respect reste sans conséquences graves. Ici, la fenêtre de temps est de l'ordre de la milliseconde.
  Exemples : appareil de lecture d'une vidéo, billetterie automatique, etc.

Par exemple, ce besoin de réactivité du SE peut être résolu par un réseau de commutation temps-réel dans le cadre d'une architecture hétérogène [Stilkerich et al., 2014]. La réalisation de ce type de système embarqué devra prendre en compte une phase d'expérimentation rigoureuse comportant une analyse précise des pires temps d'exécution du système.
Pour cette thèse, nous appliquons nos expérimentations dans le contexte du traitement d'images soumis à un besoin temps-réel souple.

2.1.4 Optimisation du système

L'intégration du système embarqué implique une limite physique en taille et en poids. Cette limitation impacte directement les composants associés au SE. Ces composants représentent les ressources nécessaires à l'accomplissement des tâches visées. Elles représentent donc un lien direct avec la performance du SE. Nous relions cette contrainte à un besoin d'optimisation du système embarqué.

Dans le cadre de cette thèse, notre problématique est directement liée à ce défi. Pour bien comprendre le fonctionnement d'un SE, il nous semble essentiel de détailler les ressources le composant. L'optimisation concerne tout type de ressources comme par exemple l'énergie [Mittal, 2014]. Pour nos travaux, nous nous focalisons sur les ressources de calcul. Nous présentons donc plus techniquement les principaux composants de type processeur pouvant constituer un SE. Nous détaillons ensuite leur organisation et leurs limites au sein d'un système embarqué. Nous listons enfin les méthodes d'optimisation possibles.

2.1.4.1 Composants matériels

Les composants matériels sont principalement des circuits logiques répartis selon trois grandes catégories : les processeurs, les circuits programmables et les circuits intégrés dédiés à une application (*Application-Specific Integrated Circuit*, ASIC) [Waghmare and Mohite-Patil, 2012]. Les composants de la famille des processeurs permettent l'exécution des instructions machines issues des programmes informatiques. Parmi ces composants, les plus utilisés dans le domaine des systèmes embarqués sont les microprocesseurs, les microcontrôleurs et deux cas plus spécifiques : les unités de processeur de signal numérique (*Digital Signal Processor*, DSP) et les unités de processeur graphique (*Graphics Processing Unit*, GPU). Les circuits programmables sont d'abord principalement représentés par les circuits logiques programmables complexes (*Complex Programmable Logic Device*, CPLD). Ces dernières évoluent ensuite pour proposer une nouvelle branche : les unités de circuit logique programmable (*Field-Programmable Gate Array*, FPGA). Parmi ces deux catégories nous détaillerons particulièrement le cas des FPGA. Enfin, les ASIC impliquent un champ très vaste d'applications possibles.

Les processeurs

Les **microprocesseurs** sont suffisamment miniaturisés pour comporter tous les éléments nécessaires à leur fonctionnement en un seul circuit intégré, comme par exemple le microprocesseur *Haswell* [Hammarlund et al., 2014] réalisé par Intel. Ils comportent principalement une unité de traitement pour l'exécution des calculs et opérations logiques, des registres pour stocker temporairement les données et une unité de contrôle pour commander l'ensemble des actions du microprocesseur en fonction des instructions du programme informatique.

L'unité de traitement regroupe trois unités d'exécution principale. La première est une unité arithmétique et logique (Arithmetic–Logic Unit, ALU). Par la suite, les deux autres furent ajoutées pour optimiser les performances des microprocesseurs : l'unité de calcul en virgule flottante capable de réaliser des opérations de calcul pour les réels ainsi que les calculs mathématiques et scientifiques complexes et l'unité multimédia chargée d'accélérer l'exécution des instructions liées à des traitements de vidéo ou de son.

Les registres correspondent à de petits blocs de mémoires internes auxquels il est possible d'accéder très rapidement. Leur taille et leur type dépend de l'architecture du microprocesseur mais nous retrouvons globalement les mêmes fonctions pour ces registres : les registres généraux et les registres d'adressage. La première catégorie stocke les données ou informations nécessaires à l'exécution du programme. La deuxième catégorie permet d'enregistrer les adresses des lignes d'instructions du programme. Les deux principaux registres d'adressages sont le pointeur de programme (*Program Counter*, PC) contenant l'adresse de l'instruction à exécuter et le pointeur de pile (*Stack Pointer*, SP) contenant l'adresse de la pile.

Enfin, l'unité de commande initie toutes les actions relatives au déroulement du programme. Elle séquence le déroulement des instructions au rythme de l'horloge, effectue la recherche en mémoire de l'instruction à exécuter (en se référant au PC), active les circuits de traitement selon l'instruction en cours et une fois l'instruction réalisée prépare l'instruction suivante pour recommencer son travail jusqu'à la dernière ligne du programme. Les microprocesseurs nécessitent d'être associés à d'autres composants pour réaliser des tâches dédiées. Selon les unités de traitement qu'ils intègrent et la capacité de leurs registres de mémoires, ils peuvent être adaptés à des tâches complexes, c'est à dire pouvant se décomposer en plusieurs sous-tâches assez différentes.

Les **microcontrôleurs** sont des circuits intégrés caractérisés par leur haut degré d'intégration des fonctions logiques, comme par exemple les microcontrôleurs PIC [Kumar and Indira, 2017]. Leur taille, leur coût et leur consommation énergétique sont réduits. Il en va de même pour leur capacité et vitesse de fonctionnement. Ils ont été typiquement conçus pour des systèmes embarqués dédiés à un type de tâche très spécifique comme par exemple une télécommande, un appareil électroménager ou des jouets. Ils rassemblent les éléments essentiels d'un ordinateur : processeur, mémoire vive et morte, unités périphériques et interfaces d'entrées-sorties. Le tout est relié par le biais de bus de données, d'adresse et de contrôle. Leur architecture peut être de type Harvard ou Von Neumann.

Une architecture de type Harvard sépare le bloc de mémoire vive (*random access memory*, RAM) dédiée au stockage des variables et le bloc de mémoire morte (*read-only memory*, ROM) utilisée pour stocker le programme principal. Établir le jeu d'instruction en mémoire morte permet de lancer directement le programme au démarrage du système embarqué contrairement à un ordinateur qui charge d'abord le système d'exploitation. Le processeur peut avoir accès simultanément aux instructions et aux données associées car il est relié aux deux mémoires par des bus séparés. Cette architecture se base sur un jeu d'instructions de type réduit (*reduced instruction set computer*, RISC), c'est à dire des instructions exécutées en un seul cycle d'horloge.

Une architecture de type Von Neumann, à l'inverse, regroupe la mémoire morte et la mémoire vive en un même composant auquel elle accède par le même bus. Elle nécessite une unité de contrôle chargée du séquençage des opérations. En effet, comme l'accès aux instructions et aux données se font par le même bus, il est important de limiter le nombre d'accès en lecture du programme. Pour cette raison, l'architecture Von Neumann se base sur un jeu d'instructions complexes (*complex instruction set computer*, CISC) réalisées en plusieurs cycles.

La capacité des processeurs est souvent représentée par leur cadencement en fréquence. Signifiée en Hertz, elle représente le nombre de cycles effectués par le composant en une seconde. Si le composant est capable d'effectuer une instruction de son programme en un cycle d'horloge alors la cadence représente sa rapidité d'exécution. Cependant il est rare qu'une instruction soit résolue en un seul cycle, cette capacité est donc représentée par une autre mesure : le nombre d'instructions résolues par seconde, généralement exprimées en termes de million (*Million Instructions Per Second*, MIPS). Comme il existe plusieurs types de jeu d'instruction (RISC et CISC, évoqués précédemment) dont les performances ne sont pas équivalentes, une référence qualifiée "Dhrystone" a été mise en place pour évaluer cette performance pour tout type de composant. La capacité du processeur est alors désignée en DMIPS (*Dhrystone Million Instructions Per Second*).

Les processeurs dédiés

Les **DSP** sont des microprocesseurs optimisés pour des applications liées au traitement numérique du signal, comme les DSP développés par Texas Instrument [Ranganadh et al., 2011]. Ils permettent ainsi de réaliser des opérations par exemple de filtrage ou d'extraction de signaux le plus rapidement possible. Leur architecture a été adaptée à ce type de traitements par l'intégration de spécificités comme la mise en place de certains registres très larges appelés accumulateurs permettant l'obtention d'un résultat plus précis. Par exemple, la famille DSP56k de Motorola propose une précision de 56 bits. Les DSP supportent également des accès simultanés à la mémoire permettant des actions de lecture et d'écriture en parallèle en un seul cycle d'horloge grâce à l'intégration de plusieurs blocs mémoires ou de mémoires multiports. En contre-partie, les DSP ont une gestion rudimentaire des interruptions. Par ailleurs, ils n'intègrent pas de mémoire cache pour limiter la latence des accès mémoire. Les DSP sont naturellement sélectionnés dans le cadre d'exécution de tâches relatives au traitement du signal.

Les **GPU** sont des circuits intégrés spécialisés dans les fonctions de calcul d'affichage [Bridges et al., 2016]. Leur structure est généralement hautement parallèle dans un objectif d'optimisation du traitement d'images, matriciel et vectoriel. Ils sont très utilisés par exemple pour des rendus 3D, de la gestion de vidéo ou la décompression d'encodage d'images. Principalement intégrés à des cartes graphiques, ces composants sont de plus en plus fréquemment employés pour des tâches beaucoup plus variées.

Les FPGA

Les **FPGA** sont des circuits en silicium reprogrammables constitués d'une matrice de blocs logiques entourés de blocs d'entrées-sorties. L'ensemble est relié par un réseau d'interconnexions. Toutes ces parties du composant sont programmables et reconfigurables. Elle offrent la possibilité pour les FPGA de remplir des fonctionnalités très personnalisées non figées dans le temps. A l'origine les FPGA sont utilisés comme interfaces d'appoint pour le pré-calcul ou le formatage de données. Depuis, leur architecture parallèle indépendante assurant rapidité et fiabilité et leur possibilité de reconfiguration totale les ont rendus populaires dans des applications de contrôle et même de traitement de signaux où ils concurrencent les DSP.

L'architecture d'un FPGA peut être vue en trois parties : la matrice de cellules, les interconnexions et la programmation. Dans la matrice, chaque cellule représente une unité configurable. Il peut s'agir d'un élément de logique séquentielle permettant de réaliser une fonction, un bloc de mémoire RAM pour stocker des données ou une entrée/sortie. L'ensemble de ces cellules sont reliées par un réseau de routage complété à chaque croisement par des blocs d'interconnexions programmables. Enfin, la programmation de ces cellules et de l'ensemble du circuit d'interconnexion permet de personnaliser le FPGA pour une application donnée. Par exemple, le FPGA Atmel AT40K et la famille de FPGA Xilinx Virtex, permettent de réaliser des modifications post-conception [Hauck and DeHon, 2007].

Les ASIC

En parallèle des processeurs permettant l'exécution de jeux d'instructions vastes et des FPGA rendant possible une configuration personnalisée à partir d'un circuit intégré générique, les **ASIC** représentent la possibilité d'obtenir un circuit très spécifique, optimisé pour la réalisation d'une tâche donnée, comme par exemple des adaptateurs de courant dédiés réalisés par STMicroelectronics [Pappalardo et al., 2017]. La réalisation d'une architecture dédiée selon un cas d'utilisation précis permet d'obtenir un temps de réponse bien supérieur pour l'application visée. En effet, les composants intégrés, l'architecture choisie, la puissance des unités de calcul, tout est réalisé sur mesure. Ce type d'intégration permet d'augmenter la fiabilité du matériel. En revanche, les coûts de développement seront eux très élevés car la spécificité du matériel empêche une production à large échelle.

2.1.4.2 Organisation des composants

La majorité des SE, en dehors des microcontrôleurs, comporte un système intégré sur une puce (*System-on-Chip*, SoC). Dans ce cadre, une partie des composants matériels nécessaires à la réalisation de nos tâches spécifiques est réunie dans un unique circuit intégré. Ces éléments sont reliés par un réseau également intégré sur la puce. Cette distinction permet un gain en performance pour les échanges d'informations des unités concernées. Si un SE comporte toujours au moins une unité processeur centrale (*Central Processing Unit*, CPU), des unités de calcul secondaires spécifiques à certaines applications peuvent lui être associées via ce SoC.
Par exemple, selon le type de tâche à traiter il est possible d'intégrer :

- un GPU pour du traitement d'images (multimédia, médical, etc.) ;
- un DSP pour du traitement de signal (télécommunication, filtres, etc.) ;
- un FPGA pouvant être reprogrammé pour apporter de la souplesse à notre SE ;
- etc.

Il est également fréquent de retrouver au moins une unité de mémoire flash (NAND) pour assurer dynamiquement la gestion des registres sur la puce. Le SoC peut ensuite être accompagné par d'autres composants dépendant de l'environnement d'utilisation. En Figure 2.1, nous présentons le noyau de l'architecture d'un système embarqué.

Le processus de décision d'intégration du nombre et du type de composants se fait en fonction de l'espace disponible sur le SE. Sachant que ces composants génèrent de la chaleur, il est nécessaire d'ajouter des unités de refroidissement supplémentaires. Parfois l'alimentation est également intégrée au SE et la capacité de la batterie en fonction de sa taille doit rentrer en ligne de compte. Par ailleurs, le logiciel associé à notre SE peut aller d'un système d'exploitation à un programme d'exécution unitaire, auquel cas l'utilisation d'un microcontrôleur est favorisée. Dans le premier cas le stockage du programme nécessite plus d'espace mémoire, or la limite physique impacte également ce type d'unité.

Le SE comporte toujours au moins un capteur et un acteur afin d'interagir avec son environnement direct. Les Convertisseurs Analogique-Numérique (CAN) et Numérique-Analogique (CNA) assurent la traduction de l'information brute en données binaires compréhensibles par le système. Optionnellement dans le cadre d'un SE interagissant directement avec un utilisateur, une Interface Homme-Machine (IHM) est intégrée. Elle peut aller de la simple diode électroluminescente (Light-Emitting Diode, LED) à un écran à cristaux liquides plus sophistiqué (*Liquid Crystal Display*, LCD). Des ports d'entrées-sorties (E/S) peuvent également être ajoutés pour assurer des transferts d'informations plus importants (USB, réseau ETHernet, série, etc.).

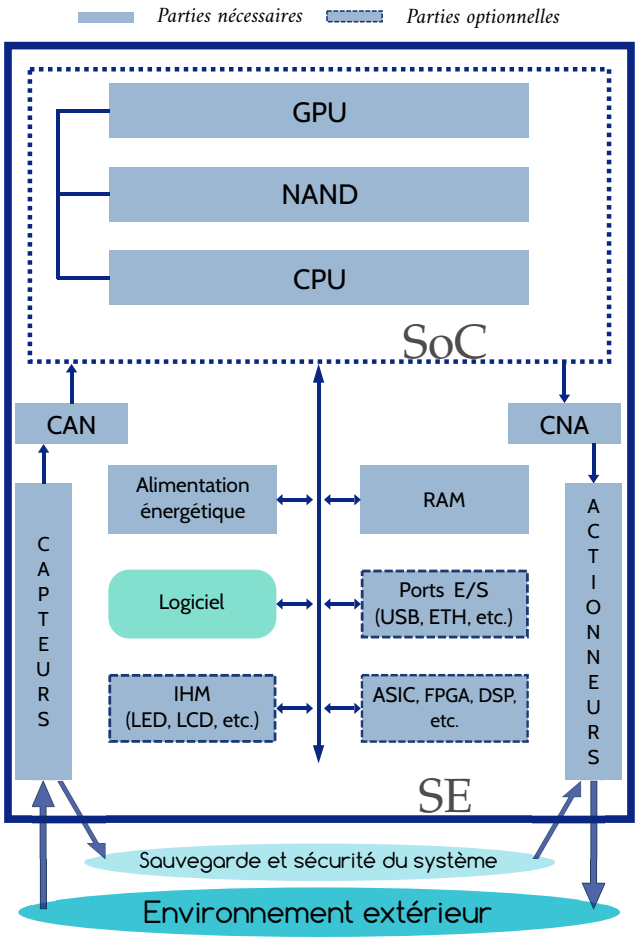


Figure 2.1 – *Noyau de l'architecture d'un système embarqué.*

Définition #2 : *Optimisation*

L'**optimisation** d'un système embarqué correspond à l'obtention du meilleur rapport entre l'encombrement et la performance de ce système tout en restant dans des coûts de production compétitifs.

De plus, avec le contexte de l'embarqué, diminuer les ressources nécessaires tout en gardant les mêmes performances grâce à une gestion plus intelligente doit permettre un gain en terme d'espace pour l'architecture matérielle.

Pour optimiser un SE, les deux principales solutions employées sont :
- la miniaturisation des composants pour maximiser leur nombre et augmenter les capacités du SE en respectant la limite d'encombrement ;
- le perfectionnement de l'allocation des tâches aux ressources déjà en place pour maximiser la rentabilité des composants matériels sans modifier leur nombre ou leur type.

C'est dans ce deuxième axe que nous positionnons nos travaux.

2.2 | Ressources et tâches

Les ressources [Définition 3], comme la mémoire, les processeurs, l'alimentation ou encore la bande passante du bus, sont essentielles au fonctionnement des SE. Elles sont cependant limitées [Vulgarakis and Seceleanu, 2008].

Définition #3 : *Ressource*

Une **ressource** correspond à un composant matériel procurant un apport nécessaire au fonctionnement d'un système embarqué et à la réalisation des tâches associées.

La gestion de ces ressources est directement liée aux tâches [Définition 4] dont elles permettent la réalisation. Une tâche peut être reliée à un capteur (ex : récupération de données), à un actionneur (ex : envoi d'informations sur un écran), ou simplement servir au fonctionnement interne du système (ex : traitement des données récupérées par le capteur pour les afficher au niveau de l'actionneur).

Définition #4 : *Tâche*

Une **tâche** est un ensemble d'instructions informatiques en attente d'exécution et permettant d'effectuer un service. Les tâches représentent ainsi une catégorie de processus, purement applicatifs.

Ainsi, les processus doivent partager l'accès aux ressources du SE. Il est donc nécessaire d'implémenter des algorithmes particuliers pour gérer les conflits pouvant découler de ces partages. L'importance d'allier la programmation au niveau logiciel à ces contraintes matérielles est donc de taille pour assurer la qualité et la performance du système embarqué.

Pour limiter les conflits au maximum et renforcer ainsi la fiabilité du SE, les fabricants déterminent lors de la phase de conception le type de tâches auxquelles les ressources sont dédiées. Par exemple, un sous-processeur sera dédié au décodage d'images, un autre à la gestion du son, etc. A chaque fois, les ressources concernées sont considérées en surplus pour éviter un blocage dans le cadre de très nombreuses requêtes pour une même tâche. Lorsque la ressource n'est pas utilisée au maximum par le processus, elle ne pourra pas être mise à disposition de processus différents. Le potentiel de l'appareil n'est donc généralement pas pleinement exploité. Par ailleurs, cette pratique ne tient pas compte des variations pouvant être induites par l'environnement ou des besoins spécifiques des utilisateurs.

Outre l'amoindrissement des conflits de partage des ressources, ce choix découle également de la contrainte matérielle. Cette dernière ne permet pas une utilisation des ressources parfaitement libre. En effet, un composant matériel (*hardware*) ne peut généralement réaliser aucune tâche sans programmes logiciels (*software*). Trois types de programmes sont distingués : les microprogrammes (*firmware*), les pilotes (*driver*) et les applications logicielles.

Les **microprogrammes** sont des logiciels spécialement conçus pour décrire les configurations et fonctionnalités d'un composant donné. Ils sont intégrés dans ce composant. Ils permettent son exploitation par les applications. Les *firmware* servent d'interfaces entre les applications logicielles et un composant. Plus une unité matérielle peut exécuter de tâches différentes, plus ses microprogrammes impliqueront une forte empreinte mémoire.

Les **drivers** offrent la même possibilité que les *firmwares*. Ils divergent de par leur positionnement car ils ne sont pas intégrés dans le composant. Ils sont gérés par un système d'exploitation (*operating system*, OS) et servent d'interface pour les composants. Les systèmes embarqués ne possèdent cependant pas tous un OS et ne peuvent alors pas compter sur les apports des drivers.

La Figure 2.2 représente ces trois types de programme et leur positionnement vis-à-vis du *hardware*.

Pour exécuter une tâche, il faut donc que le composant possède un microprogramme adapté en complément de la puissance de calcul. Afin qu'une unité puisse réaliser plusieurs types de tâches très différentes, le *firmware* doit soit :

- décrire plus de fonctionnalités. Cette solution implique une empreinte mémoire plus importante. Dans un cadre de ressources limitées, ce choix est délaissé, et un composant ne peut donc pas assurer n'importe quel travail.

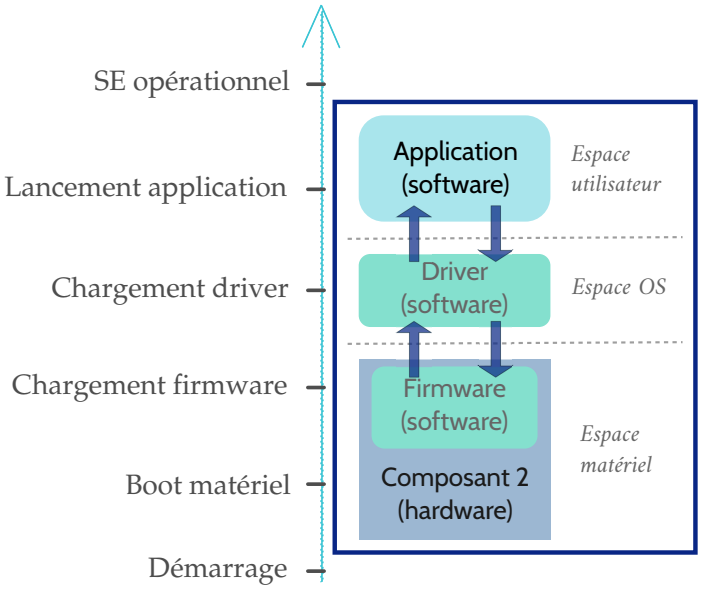


FIGURE 2.2 – *Représentation des différents types de programme (firmware, driver et application) et de leur ordre de chargement au démarrage du SE.*

- être réécrit à chaque fois pour proposer une description adaptée à la prochaine tâche à traiter. Cette option représente également une solution lourde en termes de lignes de code à implémenter. Elle reste cependant envisageable.

Sans aller jusqu'à reprogrammer chaque *firmware*, plusieurs unités de calcul sont capables d'effectuer des tâches communes. Par exemple dans le domaine du multimédia, une image peut aussi bien être décodée par le CPU que par le GPU. Chaque composant possède ses avantages et inconvénients. Le GPU propose une gestion matricielle adaptée à des travaux sur image. Il est par conséquent beaucoup plus rapide mais ne décode qu'une image à la fois. Le CPU permet un décodage de plusieurs images en parallèle (multi-tâches), mais sera moins rapide à traiter chaque image. Leurs spécificités les rendent plus adaptés à des situations différentes. Par exemple pour le parcours d'un dossier de photographies où l'utilisateur souhaite simplement retrouver une image précise et obtenir une visualisation rapide, le CPU tirera partie de son traitement de nombreuses images en parallèle. Si nous souhaitons plutôt afficher les photographies une à une en ultra-haute définition (3 840 par 2 160 pixels), le GPU tirera avantage de son traitement rapide. Face à ce choix, les fabricants font ici encore une allocation fixe des ressources en fonction du type de codec utilisé pour encoder l'image, mais pas du contexte de décodage, correspondant dans notre exemple au nombre d'images à traiter ou de leur qualité. La décision appliquée n'exploite donc pas toujours les avantages des composants.

Pour optimiser notre système embarqué par le perfectionnement de la gestion de nos ressources, nous avons deux approches possibles :

- **Gestion des ressources** : améliorer l'allocation de nos tâches à certaines ressources.

Nous travaillons alors sur l'aspect de répartition de nos différents types de tâches entre les ressources. Cette partie intervient lors de la mise en place du SE, comme par exemple dans le cadre d'un système embarqué reconfigurable nécessitant un gestionnaire de ressources [Mariani et al., 2013]. Ce type de gestion peut se concentrer sur les tâches tolérantes aux pannes pour associer la fiabilité à l'optimisation du SE [Huang et al., 2011], ou se focaliser sur la minimisation du coût d'allocation [Xie et al., 2017].

- **Gestion des tâches** : proposer une ré-affectation des tâches précédemment allouées.

Cette partie intervient en correction du travail de répartition effectué par la gestion des ressources. Elle opère en temps-réel durant le fonctionnement du SE. Nous qualifions ce principe de délégation des tâches [Définition 5].

Définition #5 : *Délégation de tâches*

La **délégation d'une tâche** à une ressource est définie par la ré-affectation de cette tâche à une cible différente.

Par exemple si la tâche de décodage de l'image est affectée au CPU déjà occupé, le principe de délégation de tâche est appliqué pour l'affecter au GPU. Nous distinguons ce principe de la capacité de reconfiguration des FPGA. En effet, pour effectuer des modifications de configuration post-conception, ces derniers nécessitent de passer par une phase d'ingénierie plus axée sur la gestion des ressources.
Une solution selon l'angle de la gestion des tâches permet de s'adapter dynamiquement au contexte, suivant ainsi la problématique de notre thèse.

La démarche de délégation des tâches correspond à notre hypothèse. Il s'agit d'une gestion apportée par l'expérimentation des SMA au sein de l'embarqué. Nous détaillerons donc cette forme de gestion lors de nos expériences. Nous détaillons ici les objectifs recherchés lors de l'allocation et les limites de la gestion des ressources pour positionner notre travail concernant cette délégation des tâches.

2.3 | Allocation des tâches

La gestion des ressources est assurée par l'ordonnancement [Définition 6] des tâches en attente. Par exemple pour une ressource de calcul, il s'agit d'allouer du temps d'exécution d'un processeur à un processus.

Définition #6 : *Ordonnancement*

L'**ordonnancement** est le mécanisme correspondant à l'allocation d'une tâche à une ressource.
Une unité dédiée à ce travail est appelée un ordonnanceur.

L'efficacité d'un algorithme d'ordonnancement des tâches peut être mesurée en fonction de critères tels que le Temps de Réponse (TR), le Temps d'Attente Moyen des tâches (TAM) et le Temps Réel d'Exécution (TRE).
Le temps de réponse correspond à l'instant t de fin d'exécution dont est soustraite la date de déclenchement de la tâche comme stipulé en Formule 2.1.
Ce temps de réponse correspond en réalité au temps réel d'exécution de la tâche auquel s'ajoute le temps d'attente subi par la tâche (TA). Ces deux données interdépendantes sont présentées en Formule 2.2.
Le temps d'attente moyen, détaillé en Formule 2.3, correspond à la somme des temps d'attente des tâches par rapport au nombre N de tâches gérées par l'algorithme d'ordonnancement.

$$TR_{tâche} = t_{fin\ exécution} - t_{déclenchement} \tag{2.1}$$

$$\begin{aligned} TR_{tâche} &= TRE_{tâche} + TA_{tâche} \\ TA_{tâche} &= TR_{tâche} - TRE_{tâche} \end{aligned} \tag{2.2}$$

$$TAM = \frac{\sum_{i=1}^{N} TA_{tâche(i)}}{N} \tag{2.3}$$

Selon les objectifs de l'ordonnanceur, les algorithmes choisis maximiseront les performances selon un critère donné. Par exemple, si l'objectif de l'ordonnanceur est de présenter un temps de réponse acceptable global, les choix algorithmiques seront orientés vers un TAM le plus faible possible. Dans le cadre d'un système temps-réel, l'objectif est de terminer chaque tâche en respectant sa date d'échéance. Les tâches à faible priorité peuvent alors accumuler des temps d'attente importants, générant un TAM plus élevé. Cela n'est pas impactant pour la performance d'ordonnancement par rapport à l'objectif fixé. La diversité des objectifs possibles a induit un grand nombre d'algorithmes d'ordonnancement existants [Goel and Garg, 2012].

À l'origine, l'algorithme d'ordonnancement standard appliqué était de type FIFO (*First In First Out*). Cet algorithme correspond à un système de file d'attente suivant un ordre d'exécution linéaire [Figure 2.3].
La capacité de la file notée N représente le nombre de tâches pouvant être placées en attente. Sur des ordinateurs ou des serveurs, cette capacité tend vers l'infini : la limite proposée lors de l'établissement de cette file est rarement atteinte. Dans le domaine de l'embarqué, cette capacité est limitée selon le nombre de ressources mémoire disponibles.

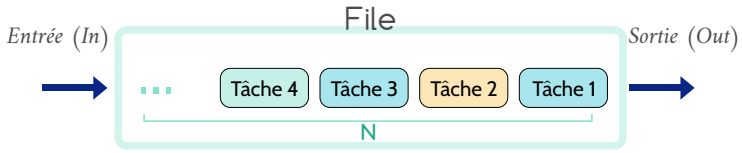


FIGURE 2.3 – *Principe de file de l'algorithme d'ordonnancement FIFO.*

L'algorithme FIFO a l'avantage d'être aisé à implémenter, de représenter une faible empreinte mémoire et d'avoir un temps de décision minime. Il représente également une solution fiable car il assure le traitement des tâches dans un ordre linéaire et donc reproductible. Cependant, il pénalise les processus à bref temps d'exécution dont le TA risque d'être largement supérieur à leur TRE. Par ailleurs, l'algorithme FIFO ne gère pas la notion de priorité. Dans le cadre de systèmes temps-réel, il n'est pas pertinent car il ne permet pas d'assurer le respect de l'échéance d'une tâche donnée.

Ce manque d'adaptation fut un point de départ pour réaliser de nouveaux algorithmes d'ordonnancement. Aujourd'hui, les algorithmes les plus utilisés pour l'allocation de tâches peuvent être répartis selon deux grandes catégories d'ordonnancement : l'ordonnancement en temps partagé et l'ordonnancement en temps-réel.

L'**ordonnancement en temps partagé** correspond aux algorithmes traitant les tâches en attente dans l'optique d'optimiser le TAM. Ces algorithmes se soucient principalement de répartir l'allocation des ressources selon des critères internes aux tâches (type de ressource nécessaire, durée de la tâche, etc.). Ils peuvent être préemptifs, c'est à dire qu'une tâche tout juste arrivée peut prendre le dessus sur la tâche en cours, mais toujours en fonction de ces critères internes. Les algorithmes d'ordonnancement en temps partagé sont les plus communément employés sur les systèmes non contraints par des impératifs temporels.

L'**ordonnancement en temps-réel** prend en considération les notions de priorité et d'échéance. Ils permettent la prise en compte de contraintes externes à la tâche en attente. Ils nécessitent de la part de l'ordonnanceur une estimation de la longueur en nombre de cycles processeurs d'une tâche et de sa planification par rapport à une estimation du nombre de cycles restant avant son échéance. Cette partie implique une empreinte mémoire plus importante pour ce type d'ordonnancement et ramène son utilisation aux systèmes éponymes presque exclusivement.
De nombreuses variantes existent pour ces deux types d'ordonnancement tout en restant marquées par ces caractéristiques principales.
Nous listons trois algorithmes représentatifs pour chaque catégorie en Table 2.1. Nous y résumons leurs différences en fonction des critères suivants :

- **Algorithme préemptif (Pr)** : L'exécution d'une tâche peut être interrompue en faveur d'une autre tâche plus prioritaire.
- **Arrivée (Ar)** : L'ordre d'arrivée détermine la sélection entre plusieurs tâches éligibles pour une allocation.
- **Durée d'exécution (Du)** : La priorité est donnée à la tâche la plus courte en termes de nombre de cycles nécessaires à sa réalisation.
- **Échéance (Ec)** : La date limite à laquelle la tâche doit avoir été exécutée est considérée. La tâche avec le moins de temps restant est priorisée.
- **Répartition égale (Eg)** : Si plusieurs tâches sont éligibles, l'ordonnanceur leur alloue tour à tour des ressources pour un temps donné. Ce temps est le même pour toutes. Plus le nombre de tâches concernées est important, plus le temps accordé est court.

Nous pouvons constater que la prise en compte d'une priorité liée à l'échéance est représentative d'un ordonnancement temps-réel. La notion d'égalité de répartition est elle assez rare car elle implique des changements fréquents de contexte d'exécution.

Le mécanisme d'ordonnancement intervient à deux reprises : entre différentes unités matérielles et de façon interne au sein d'un même composant. Pour la première étape, les tâches sont réparties selon le type de ressources

Algorithme	Description	Catégorie	Pr	Ar	Du	Ec	Eg
FIFO [Beaumont et al., 2006]	(*First In First Out*) Allocation des processus selon leur ordre d'arrivée. Pénalise les processus à bref temps d'exécution ou prioritaires.	Temps partagé		X			
Round-Robin [Rasmussen and Trick, 2008]	Répartition du temps de calcul d'un processeur de façon équitable entre les processus en attente.	Temps partagé		X			X
SJF [Lupetti and Zagorodnov, 2006]	(*Shortest Job First*) Peut être implémenté en préemptif (possibilité d'interrompre la tâche en cours). Favorise les tâches courtes.	Temps partagé	X		X		
EDF [Liu and Layland, 1973]	(*Earliest Deadline First*) Priorité plus importante aux tâches ayant la plus proche échéance.	Temps réel	X			X	
RMS [AlEnawy and Aydin, 2005]	(*Rate-Monotonic Scheduling*) Priorité aux tâches de plus faibles périodes. Ciblé pour des tâches périodiques synchrones et indépendantes.	Temps réel			X	X	
CFS [Beisel et al., 2011]	(*Completely Fair Scheduler*) Arbre bicolore représentant une chronologie des futures exécutions des tâches : l'arbre trie les processus selon le manque de temps d'exécution par rapport au temps alloué par un processeur multi-tâches idéal.	Temps réel	X		X	X	

Table 2.1 – *Algorithmes d'ordonnancement utilisés dans les systèmes embarqués. Selon les caractéristiques suivantes : Préemptif (Pr), ordre d'Arrivée (Ar), Durée du processus (Du), Échéance proche (Ec), Répartition égale (Eg).*

dont elles ont besoin. Dans la seconde étape, l'algorithme d'ordonnancement permet de déterminer à quelle tâche la ressource est allouée parmi celles en attente. La Figure 2.4 illustre la première étape par une représentation d'un ordonnancement sur l'ensemble du système. Les tâches sont allouées selon leur type à l'unité matérielle correspondante. La numérotation des tâches correspond à leur ordre d'arrivée et leur couleur détermine leur type. Après le travail de l'ordonnanceur, les tâches ont été allouées à une unité mais ne sont pas encore forcément exécutées par la ressource.

Nous proposons deux exemples pour représenter la deuxième étape : un algorithme d'ordonnancement en temps partagé avec SJF et en temps-réel avec EDF [Figure 2.5]. Pour chaque figure illustrant ces exemples, la numérotation des tâches correspond toujours à leur ordre de déclenchement.

Le premier exemple, illustré à gauche, présente une unité pilotée par un algorithme d'ordonnancement SJF préemptif. Au départ seule la tâche n°2 est en attente, l'ordonnanceur lui alloue donc la ressource pendant plusieurs cycles pour son exécution. Au cours du deuxième cycle, la tâche n°7 est également rattachée à cette unité. L'ordon-

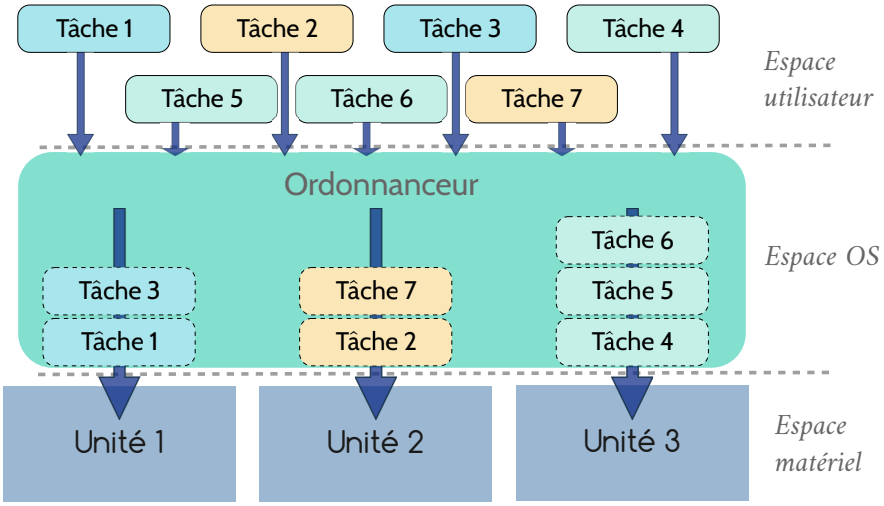


FIGURE 2.4 – *Représentation d'un ordonnancement entre différentes unités matérielles.*

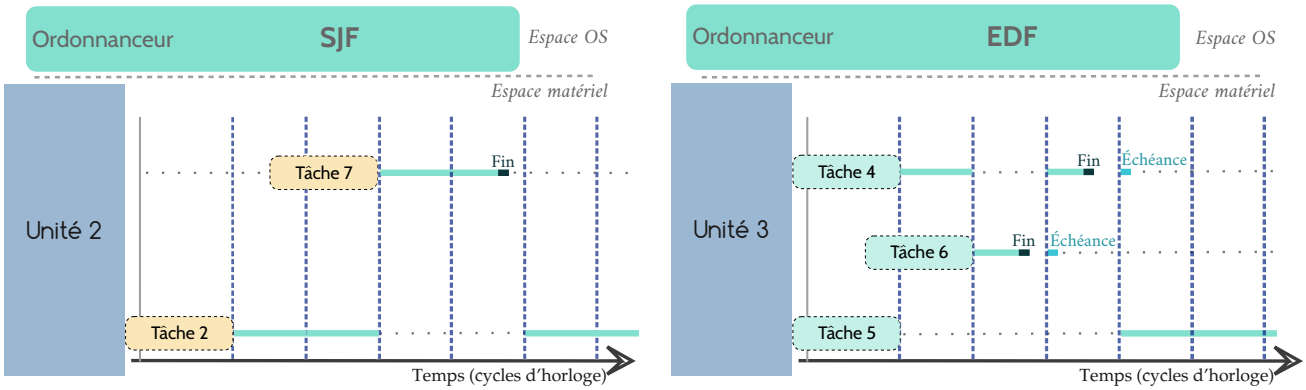


FIGURE 2.5 – *Exemples d'ordonnancement au sein d'unités selon deux mécanismes différents tout deux préemptifs.*

nanceur estime sa durée d'exécution et analyse qu'elle est plus courte que celle de la tâche en cours d'exécution. Elle va donc être priorisée. Comme l'algorithme est préemptif, le contexte d'exécution est directement permuté pour permettre à la tâche n°7 de s'exécuter. Une fois achevée, l'ordonnanceur redonne la main à la tâche n°2 pour qu'elle poursuive son exécution.

Le second exemple, illustré à droite, présente une unité régulée par un algorithme EDF préemptif. Cette unité se voit assigner presque simultanément les tâches n°4 et 5. Comme la tâche n°4 est celle possédant l'échéance la plus proche, elle est priorisée. Au deuxième cycle, la tâche n°6 est assignée à l'unité 3. Son échéance étant plus courte que celle de la tâche en cours d'exécution, l'ordonnanceur lui passe la main. Une fois la tâche achevée, il repasse à la tâche n°4 pour qu'elle puisse être achevée avant sa date d'échéance. Enfin, l'exécution de la tâche n°5 est reprise.

Les algorithmes d'ordonnancement visent des objectifs différents selon les critères d'évaluation retenus [Formules 2.1, 2.2 et 2.3]. Il est cependant notable qu'ils se basent sur des paramètres établis sans prendre en compte une éventuelle évolution de leur environnement ou des besoins utilisateurs. Le système ne peut donc pas s'adapter à un contexte changeant en prenant en compte plusieurs indicateurs de façon modulable. Nous souhaitons pouvoir prendre dynamiquement en compte ce contexte grâce aux systèmes multi-agents appliqués à une délégation des tâches. Pour faire intervenir ce principe de délégation des tâches sans modifier les composants du SE, nous nous concentrons sur les allocations des tâches entre les différentes unités matérielles (première étape d'ordonnancement).

Notre proposition implique des échanges d'informations entre les différents processus concernés et leur environnement. Nous nous intéressons donc aux moyens de communication utilisés en embarqué.

2.4 | Communication entre les tâches

2.4.1 Notion de processus

A travers la Définition 4, nous avons caractérisé une tâche comme un ensemble de processus logiciels permettant d'effectuer un service.

Définition #7 : *Processus*

Un **processus** est un programme informatique dédié à une ou plusieurs fonctionnalités. Il évolue avec des ressources propres et non partagées.

Un processus peut se découper en plusieurs fils d'exécution appelés *threads* (ou processus légers). Un thread représente également l'exécution d'instructions d'un programme. Cependant, si deux processus ne partagent pas de ressources mémoire, deux threads issus d'un même processus partageront eux les mêmes zones de mémoires : celles allouées au processus parent. Les threads sont utilisés pour gérer plusieurs tâches en "parallèle". Nous mettons ce terme entre guillemets car en réalité le temps processeur est tout de même réparti entre les différents threads.

Définition #8 : *Thread*

Un **thread** est issu d'un processus. Il regroupe une sous-partie du programme de ce dernier. Il permet ainsi son exécution en parallèle de la suite du programme du processus parent. Il partage les ressources mémoire avec le processus parent et les autres threads issus de ce même processus.

Tout échange d'information ou synchronisation entre deux processus se fait à travers des protocoles formalisés.

2.4.2 Échange d'informations et synchronisation

Comme deux processus n'ont pas accès aux mêmes zones mémoires, ils doivent enregistrer les informations qu'ils souhaitent échanger dans des zones accessibles pour tous. Deux méthodes sont possibles : la mémoire centralisée (par exemple avec l'utilisation de fichiers) ou la mémoire partagée. Pour ces deux méthodes le principe est similaire : enregistrer les données dans des zones de mémoires communes à tous les processus. Elles restent limitées à un même SE. Dans les deux cas, il faut faire attention aux accès concurrentiels. Si deux processus tentent d'écrire au même moment dans la même zone mémoire, deux types de problème peuvent survenir :

- les données sont corrompues ou ne sont plus cohérentes : cas le plus fréquent avec les fichiers ;
- les processus concernés génèrent des erreurs mémoires de type *segmentation fault* provoquant une interruption involontaire du processus : cas le plus fréquent avec la mémoire partagée.

Dans le cadre de partage d'informations, il y a donc une notion de section mémoire critique. Ces sections nécessitent d'être protégées afin d'assurer que les processus y accèdent à tour de rôle et non pas simultanément. Les mécanismes de synchronisation permettent de bloquer ou débloquer l'accès à une section critique. Un processus demandera le blocage de l'accès avant d'accéder à la section. Si cette dernière était déjà bloquée car utilisée par un autre processus, le programme est mis en attente jusqu'à ce que la section soit de nouveau disponible.

Nous listons ici les principaux mécanismes de synchronisation :

- Les **signaux**.

Ils fonctionnent sous forme d'interruption asynchrone d'un processus en cours et sont toujours gérés de façon parallèle. Ils sont généralement utilisés pour terminer un processus en cas d'erreur. Le programme est placé en attente active de la réception d'un signal avant de continuer son exécution. Le signal sera renvoyé par un autre processus quand celui-ci sera prêt. La notion d'attente active implique que le processus continue de consommer la ressource processeur.

- Les **verrous**.

Principalement utilisés pour protéger des fichiers, ils peuvent bloquer un accès en écriture tout en permettant à tout processus le souhaitant de lire le contenu du fichier (verrou partagé), ou simplement bloquer tout accès en lecture et

écriture par tout autre processus que celui ayant posé le verrou (verrou exclusif). Les verrous impliquent également une attente active.

- Les **sémaphores**.

Mécanisme plus général, un sémaphore peut être utilisé pour n'importe quel type de ressource représentant une section critique. Contrairement aux signaux et aux verrous, il provoque une attente passive, c'est à dire que le processus est placé en état de sommeil et permet au processeur de reprendre la main. Un sémaphore est initialisé avec le nombre de ressources qu'il va protéger. Les sémaphores peuvent être appliqués pour synchroniser deux processus entre eux en dehors d'un accès à une section critique.

- Les **mutex**.

Un mutex est un sémaphore n'autorisant l'accès qu'à un seul processus. Il est qualifié de sémaphore d'exclusion mutuelle (*MUTual EXclusion*). Un mutex fonctionne de façon booléenne pour protéger une ressource. Cette dernière sera soit disponible, soit bloquée. Quand un processus veut se servir d'une ressource critique, il demande au mutex de bloquer son accès. Quand il a fini son utilisation, il indique au mutex que la ressource est de nouveau disponible. Un processus souhaitant accéder à une ressource bloquée par un mutex sera mis en attente passive par ce dernier. Lors de la libération de la ressource, le processus mis en attente pourra en disposer à son tour.

Les mécanismes de synchronisation utilisés pour sécuriser l'échange d'informations peuvent cependant générer des erreurs d'interblocage (tous les processus sont en attente les uns des autres), et le moyen le plus sûr de communiquer entre processus reste d'employer des méthodes dédiées regroupant cet échange d'informations et la synchronisation.

2.4.3 Protocoles de communication

Les mécanismes regroupant les deux spécificités précédemment décrites utilisent le principe des files de messages. Un processus souhaitant envoyer des informations à un autre se connecte à une file, écrit les données dans un buffer mémoire et envoie ensuite ce contenu sous forme de message à l'entrée de la file où un autre processus pourra les récupérer et les enregistrer en mémoire. Les opérations d'envoi et de récupération des messages sont bloquantes afin d'assurer la synchronisation. Ces mécanismes sont qualifiés de communication inter-processus (*Inter-Process Communication*, IPC) et suivent des protocoles établis [Définition 9].

Définition #9 : *Protocole*

Un **protocole** correspond à un ensemble de règles et de procédures à respecter pour réaliser une opération complexe.
Dans ce mémoire, nous parlons principalement de protocoles de communication (SE), d'interaction (SMA) et d'évaluations (SE et SMA).

Nous listons ci-après les principaux IPC pour les systèmes embarqués.

Les tubes et tubes nommés

La communication par tube (*pipe*) se produit sous forme d'un flot continu d'octets passant par le noyau du SE. Pour créer un tube, il faut initier un descripteur pour l'entrée (écriture) et un descripteur pour la sortie (lecture). Les descripteurs étant déclarés en interne, seul le processus les ayant créés et ses processus parents (par exemple ses threads) peuvent utiliser le tube. Il s'agit donc d'un IPC interne. Un tube est unidirectionnel, un processus aura donc accès à l'écriture, et un autre processus aura accès à la lecture. Le flot d'octets est organisé selon l'ordre d'arrivée des données. Une fois lues, ces dernières quittent définitivement le tube. Ce dernier possède une capacité limitée et peut donc mettre un processus écrivain en attente d'un processus lecteur si le buffer est complet, ou inversement s'il est vide. Les tubes sont détruits lorsque le processus les ayant créés disparaît.

La Figure 2.6 présente un schéma du fonctionnement d'un tube. Les processus P1 et P2 y sont connectés en écriture, le processus P3 y est connecté en lecture. P1 envoie tout d'abord le message "Bonjour.", P2 envoie ensuite le message "Comment ça va ?". Les données sont traitées dans leur ordre d'arrivée par le tube et le processus P3

recevra donc à la réception l'ensemble des informations à la suite.

Reprenant les principes de fonctionnement de base des tubes, les tubes nommés (*named pipe*) apportent l'avantage d'être déclarés au niveau du système de fichiers. Par conséquent, n'importe quel processus peut les utiliser à partir de leur référence.

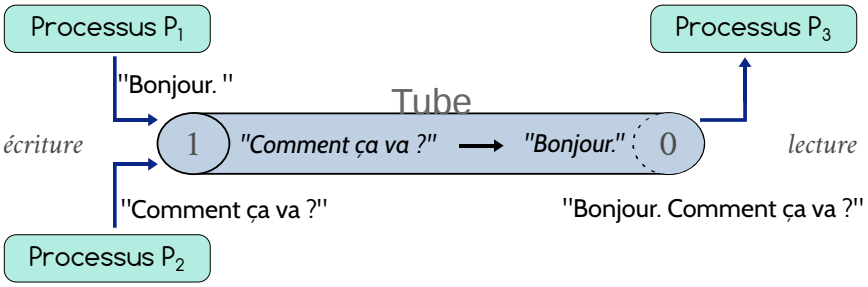


Figure 2.6 – *Représentation d'un tube, l'un des principaux IPC.*

Les files de messages

Cet IPC propose un canal de communication permettant des liaisons asynchrones (le rédacteur et le lecteur ne sont pas obligés d'être connectés en même temps). Un processus souhaitant envoyer des informations se connecte à la file, écrit les données dans un buffer mémoire et envoie ensuite ce contenu sous forme de message à l'entrée de la file. Le processus souhaitant les récupérer doit se connecter également sur cette file et récupérer le message en sortie. Il en tirera les données qu'il pourra lui-même enregistrer en mémoire.

Les opérations d'envoi et de récupération des messages sont bloquantes afin d'assurer la synchronisation. L'ordre de lecture par défaut est FIFO, mais comme les messages sont typés par un identifiant, il est possible de les lire dans un ordre différent pour accéder exclusivement aux données qui nous intéressent.

Cet IPC propose plusieurs options pour les données :
- la persistance, c'est-à-dire l'enregistrement des messages dans des fichiers pour assurer leur conservation ;
- la sécurité avec la protection des messages ;
- l'obsolescence, c'est-à-dire l'indication d'une limite de validité ;
- le filtrage avec la proposition d'options pour ne recevoir que certains types de messages ;
- et l'accusé de réception (*acknowledge*) pour indiquer à l'envoyeur quand son message a été reçu.

Les sockets

Les sockets proposent des canaux de communication passant par une interface réseau (locale ou entre plusieurs systèmes) à laquelle elles ajoutent un ensemble de primitives pour assurer le service. Elles peuvent fonctionner en mode connecté (synchrone) ou non (asynchrone) selon les besoins. Les sockets transfèrent des suites d'octets et nécessitent donc la spécification d'un protocole de communication [Définition 9] pour assurer la préservation du message.

Les différents protocoles de communication servent principalement à encapsuler les données avec des informations de description de ces données et des informations concernant l'expéditeur et le destinataire. Il est nécessaire que ces deux derniers utilisent le même protocole de communication pour s'échanger des informations.

Pour l'envoi d'un message d'un système à un autre, c'est souvent une suite de protocole (*protocol stack*) qui assure la transmission. Il est alors question d'un modèle de transport. Deux modèles représentent les standard actuellement : le modèle d'interconnexion des systèmes ouverts (*Open Systems Interconnection*, OSI) et le modèle TCP/IP, d'après le nom des protocoles qu'il emploie le plus fréquemment, aussi appelé modèle Internet. Les deux modèles furent développés sensiblement au même moment ; ils sont similaires. Le modèle OSI, plus détaillé, se compose de 7 couches présentées en Table 2.2.

Couche	Description
Application	Cette première couche propose une interface logicielle avec les applications. Elle représente le niveau le plus proche de l'utilisateur.
Présentation	Elle expose le format des données à transférer, leur représentation et éventuellement leur compression ou chiffrement. Ces indications sont indépendantes du système.
Session	Cette couche détermine l'ouverture ou la fermeture des sessions de communication entre les systèmes du réseau.
Transport	Elle propose le transfert des données découpées en paquets à transmettre et prend en charge les erreurs de transfert.
Réseau	Elle gère l'acheminement des données encapsulées à travers le réseau en proposant un routage et adressage de ces dernières.
Liaison données	Cette couche établit l'interface avec la carte réseau et définit le partage du média de transmission.
Physique	Enfin, la dernière couche établit la conversion des données en signaux inter-systèmes (numérique) ou intra-système (analogiques).

Table 2.2 – *Couches du modèle de communication OSI.*

Le modèle TCP/IP se compose de 4 couches détaillées dans la Table 2.3. Nous y indiquons également l'équivalence avec les couches du modèle OSI.

Couche	Correspondance OSI	Description
Application	(3 premières couches OSI)	Propose une interface avec les autres applications de réseau.
Transport	(4 ème couche OSI)	Permet le transfert des données découpées en paquets à transmettre et prend en charge les erreurs de transfert.
Internet	(5 ème couche OSI)	Gère l'acheminement des paquets de données à travers le réseau en permettant leur routage et leur adressage.
Accès réseau	(6 ème et dernière couches OSI)	Spécifie la forme d'acheminement des données quel que soit le type du réseau utilisé.

Table 2.3 – *Couches du modèle de communication TCP/IP.*

La partie protocolaire de l'application des services des sockets est gérée par deux protocoles de la couche Application du modèle TCP/IP : le protocole de datagramme utilisateur (*User Datagram Protocol*, UDP) en mode non connecté, et le protocole de contrôle de transmissions (*Transmission Control Protocol*, TCP) en mode connecté. Tous deux sont des protocoles de la couche transport. La communication entre machines distinctes passe par le réseau et est gérée par le protocole Internet (*Internet Protocol*, IP) de la couche du même nom.

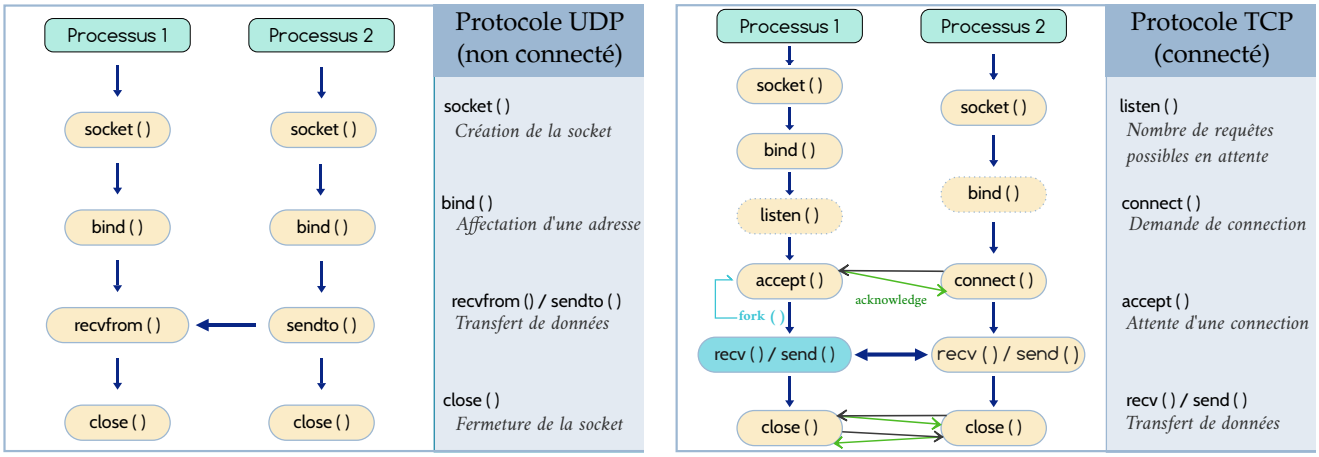


Figure 2.7 – *Diagramme de présentation des fonctions sockets en mode connecté et non connecté.*

La Figure 2.7 présente les diagrammes des fonctions des sockets des deux modes. Dans le cadre du protocole UDP, les deux processus suivent globalement le même diagramme : ils créent une socket et lui affectent la même adresse. A partir de cette affectation, la socket est ouverte entre ces deux processus. Sur la figure, le processus 2 contacte l'autre en utilisant la fonction "sendto". Il ferme ensuite sa connexion à la socket. Comme le protocole est non connecté, les deux processus n'ont pas besoin d'être synchronisés pour ce transfert d'information. Le processus 1 peut tout à fait venir récupérer les informations stockées dans la socket avec la fonction "recvfrom" après que le processus 2 ait fermé sa connexion.

Le mode connecté du protocole TCP implique lui une synchronisation au moment des échanges d'informations. Elle se traduit par la fonction "accept" qui va permettre au processus l'appliquant d'établir des connexions synchronisées en restant à l'écoute du réseau. Le processus 2 utilise la fonction "connect" pour établir une synchronisation avec le processus 1 sur une socket déterminée. Pour que la connexion soit effective, le processus 1 va accepter la connexion. Une fois celle-ci établie, les deux processus peuvent échanger autant d'informations qu'ils le souhaitent avec les fonctions "recv" et "send". Lorsque la communication est coupée d'un côté, la synchronisation est rompue mais la communication reste établie jusqu'à ce que les données en transit soient récupérées. La socket est effectivement libérée uniquement une fois la connexion arrêtée par les deux processus.

Avec ce mode, la fonction "bind" est optionnelle pour le processus qui demande une connexion. En effet, la fonction "connect" intègre déjà l'information d'adresse associée à la fonction "bind". De plus, en mode TCP, un processus peut gérer plusieurs connexions grâce à l'emploi de la fonction "listen". Cette dernière permet de mettre en attente des demandes de connexion détectées si une connexion est déjà en cours. En utilisant la fonction "fork", permettant la création d'un thread fils, la connexion peut être exécutée en parallèle. Il est alors possible d'accepter immédiatement une nouvelle connexion avec le processus principal. Les threads se terminent naturellement à la fin de la connexion à laquelle ils ont été reliés. L'utilisation de ces fonctions optionnelles permet une écoute presque discontinue du réseau. Par ailleurs, en mode connecté, les connexions et déconnexions sont fiabilisées par un accusé de réception, et l'état de la connexion est régulièrement vérifié pendant le transfert des données.

D-Bus

Les mécanismes d'IPC sont des primitives bas-niveau et ne constituent pas à eux seuls un système de communication complet.
Le protocole inter-processus D-Bus utilise ces primitives pour proposer un service de transfert de messages entre applications de façon standardisée à travers un bus de communication. Ce protocole apporte aussi de nombreux paramètres et avantages comme par exemple la sécurisation des communications, l'activation de services à la demande ou de modèles normalisés d'appel de fonctions, la possibilité d'établir des communications synchrones ou asynchrones ou encore sa capacité à permettre un travail modulaire. Il s'agit d'un IPC interne au SE.

D-Bus met à disposition deux types de bus : système et session.
Le premier correspond au bus de la machine elle-même. Il permet de contacter des "daemons", c'est à dire des processus s'exécutant en arrière plan au lieu d'être sous le contrôle direct de l'utilisateur, de très bas niveau et directement liés aux spécificités matérielles de la machine. Pour chaque application, il est possible d'ouvrir un bus session distinct. Nous y retrouvons donc des services plutôt logiciels et notamment les services liés à l'utilisateur. Lors du développement d'une application employant D-Bus, il est possible de se raccorder à un bus session existant ou d'en créer un nouveau dédié aux processus de notre application.

D-Bus va permettre de relier deux processus par le biais de rôles "client" et "service". Un processus peut mettre un service à disposition des autres sur le bus en créant une interface de contact enregistrée par D-Bus. Les clients peuvent alors contacter ce service. Bien souvent, les processus prennent les deux rôles en étant client tout en proposant un ou plusieurs services.

Une interface permet aussi de définir comment le service pourra être contacté en spécifiant des méthodes et des signaux. Les méthodes correspondent à des messages ciblés pour un service donné, elles permettent de définir le contenu du message ainsi que les paramètres à utiliser pour contacter le service. Les signaux sont des notifications d'événements sans accusé de réception. Ils sont diffusés sur le bus sans spécification d'un destinataire par l'expéditeur. Plusieurs processus peuvent s'abonner au signal de l'interface d'un service pour pouvoir détecter les événements ainsi diffusés.

Pour le routage des messages de type méthode, l'encapsulation du message contient une adresse source et une adresse de destination. Ces adresses sont représentées sur D-Bus par des interfaces spécifiques appelées "objets" et leur chemin d'accès. Un service peut proposer plusieurs objets pour le contacter. Dans la majorité des cas le service exporte un seul objet portant son nom.

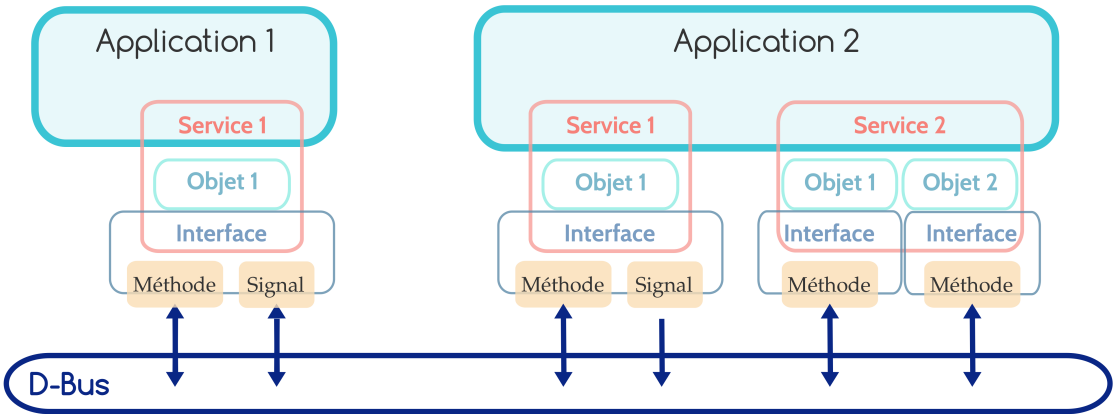


FIGURE 2.8 – *Exemple de deux applications connectées au protocole D-Bus.*

La Figure 2.8 illustre deux applications ayant enregistrées des services sur D-Bus. La première ne propose qu'un seul service à travers une interface spécifiant une méthode et un signal. L'interface est rattachée au service par l'adresse objet. La deuxième spécifie deux services, l'un avec une interface reliant une méthode et un signal à une adresse objet, l'autre proposant deux interfaces spécifiant chacune leur propre méthode avec une adresse objet distincte.

Par son fonctionnement, D-Bus permet donc de découper des applications en plusieurs modules communicants. Si un module rencontre un problème il n'impactera pas le fonctionnement des autres modules de la même application. Il est cependant important de noter que D-Bus a été pensé pour un fonctionnement en local entre processus tournant sur la même machine. Sa capacité à faire interagir les processus entre eux et avec des applications systèmes très bas niveau d'une façon standardisée et donc simplifiée en fait une solution adaptée pour les systèmes embarqués. En effet, ces derniers nécessitent souvent des interactions avec des périphériques très spécifiques, par exemple pour réagir à l'ajout d'une clef USB sans intervention utilisateur.

Dans le cadre de nos travaux, les deux derniers IPC considérés correspondent le plus à nos besoins liés à des communications au sein de SMA. Nous précisons notre proposition de leur application pour des interactions multi-agents dans la suite de ce mémoire.

2.5 | Évaluation des travaux dans les systèmes embarqués

Pour évaluer des travaux, il faut d'une part pouvoir les tester, et d'autre part pouvoir analyser les résultats des expérimentations par rapport à des objectifs posés.

Deux formes d'évaluation sont distinguées : statique et dynamique. La première consiste à vérifier le système sans l'exécuter par exemple en effectuant la vérification du code, des connectiques ou encore la vérification du respect des spécifications. Par abus de langage, nous utilisons couramment le terme "test" pour désigner les évaluations dynamiques. Elles impliquent d'activer le système éprouvé pour le tester en conditions réelles. Ces expérimentations peuvent se découper en trois catégories :

- les tests de robustesse. L'objectif est de vérifier la façon dont le système se comporte face à une utilisation erronée ou non prévue.

- les tests de performance. Les critères de performances du système tels que le temps de réponse ou la consommation des ressources sont évalués pour un contexte donné.

- les tests de sécurité. Le but est de détecter d'éventuelles vulnérabilités du système pour pouvoir les corriger a posteriori.

L'évaluation selon des objectifs peut également impliquer l'estimation de variables qualitatives, c'est à dire non quantifiables mais dont la qualité peut être déterminées suivant des codes bien définis. Par exemple dans le domaine multimédia après avoir appliqué un filtre sur une image, il n'est pas possible de "mesurer" au sens propre du terme la validité du résultat. Il est alors nécessaire de définir des critères pour pouvoir évaluer qualitativement le résultat, comme par exemple la netteté des contours ou le rendu des couleurs.

Dans le cadre de notre travail, nous éprouvons nos résultats en partie par des évaluations de performance et à travers des considérations qualitatives. Au niveau des performances, notre thèse s'appuie notamment sur une adaptation d'algorithmes à un contexte de ressources limitées. Nous nous concentrons donc sur des critères liés à cette problématique :

- le temps de réponse du système,
- le pourcentage de consommation du processeur,
- le nombre d'unités mémoires utilisées.

Bibliographie

[AlEnawy and Aydin, 2005] AlEnawy, T. A. and Aydin, H. (2005). Energy-aware task allocation for rate monotonic scheduling. In *11th IEEE Real Time on Embedded Technology and Applications Symposium (RTAS)*, pages 213–223. IEEE Computer Society Press.

[Aza-Vallina et al., 2011] Aza-Vallina, D., Denis, B., and Faure, J. (2011). *Communications reliability analysis in networked embedded systems*, pages 2594–2602. CRC Press.

[Balaji et al., 2017] Balaji, M., Aarthi, E., Kalpana, K., Nivetha, B., and Suganya, D. (2017). Adaptable and reliable industrial security system using pic controller. *International Journal for Innovative Research in Science & Technology (IJIRST)*, 3(12) :56–60.

[Beaumont et al., 2006] Beaumont, O., Marchal, L., Rehn, V., and Robert, Y. (2006). Fifo scheduling of divisible loads with return messages under the one-port model. In *International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1–25. IEEE Computer Society Press.

[Beisel et al., 2011] Beisel, T., Wiersema, T., Plessl, C., and Brinkmann, A. (2011). Cooperative multitasking for heterogeneous accelerators in the linux completely fair scheduler. *Application-Specific Systems, Architectures, and Processors (ASAP)*, pages 223–226.

[Benhani et al., 2017] Benhani, E. M., Marchand, C., Aubert, A., and Bossuet, L. (2017). On the security evaluation of the arm trustzone extension in a heterogeneous soc. In *30th IEEE International System-on-chip Conference (SOCC)*, pages 1–6. IEEE Computer Society Press.

[Bridges et al., 2016] Bridges, R. A., Imam, N., and Mintz, T. M. (2016). Understanding gpu power : A survey of profiling, modeling, and simulation methods. *ACM Computing Surveys*, 49(3) :1–27.

[Chao and Fang, 2012] Chao, L. and Fang, H. (2012). Research on reliability design of data storage for embedded system. *Physics Procedia, International Conference on Solid State Devices and Materials Science*, 25 :1405 – 1408.

[Etiemble, 2016] Etiemble, D. (2016). *Introduction aux systèmes embarqués, enfouis et mobiles*. Editions Techniques de l'Ingénieur.

[Goel and Garg, 2012] Goel, N. and Garg, R. (2012). A comparative study of cpu scheduling algorithms. *International Journal of Graphics & Image Processing (IJGIP)*, 2(4) :245–251.

[Habiba et al., 2014] Habiba, U., Khan, S. A., and Javed, Y. (2014). Gap analysis in software engineering process adoption in implementing high end embedded system design. pages 495–503, Department of Computer Engineering, College of EME, National University of Sciences and Technology (NUST), Islamabad, Pakistan. TextRoad Publication.

[Hammarlund et al., 2014] Hammarlund, P., Martinez, A. J., Bajwa, A. A., Hill, D. L., Hallnor, E., Jiang, H., Dixon, M., Derr, M., Hunsaker, M., Kumar, R., Osborne, R. B., Rajwar, R., Singhal, R., D'Sa, R., Chappell, R., Kaushik, S., Chennupaty, S., Jourdan, S., Gunther, S., Piazza, T., and Burton, T. (2014). Haswell : The fourth-generation intel core processor. *IEEE Micro*, 34(2) :6–20.

[Hauck and DeHon, 2007] Hauck, S. and DeHon, A. (2007). *Reconfigurable Computing : The Theory and Practice of FPGA-Based Computation*. Systems on Silicon. Morgan Kaufmann.

[Huang et al., 2011] Huang, J., Blech, J. O., Raabe, A., Buckl, C., and Knoll, A. (2011). Analysis and optimization of fault-tolerant task scheduling on multiprocessor embedded systems. In *7th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, pages 247–256. ACM.

[Kocher et al., 2004] Kocher, P., Lee, R., McGraw, G., Raghunathan, A., and Ravi, S. (2004). Security as a new dimension in embedded system design. In *41st Annual Design Automation Conference (DAC)*, pages 753–760. ACM.

[Kumar and Indira, 2017] Kumar, G. S. and Indira, S. (2017). Implementation of pic microcontroller based boost converter for solar panel without battery backup. *International Journal of Scientific Research in Science, Engineering and Technology (IJSRSET)*, 3(6) :52–55.

[Laurgeau et al., 2002] Laurgeau, C., Steux, B., and Metman, G. (2002). Outils pour le prototypage des systèmes embarqués temps réel. *Revue Jautomatise*, 21(21) :58–62.

[Liu and Layland, 1973] Liu, C. L. and Layland, J. W. (1973). Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1) :46–61.

[Lupetti and Zagorodnov, 2006] Lupetti, S. and Zagorodnov, D. (2006). Data popularity and shortest-job-first scheduling of network transfers. In *International Conference on Digital Telecommunications (ICDT)*, pages 26–32. IEEE Computer Society Press.

[Mariani et al., 2013] Mariani, G., Sima, V.-M., Palermo, G., Zaccaria, V., Marchiori, G., Silvano, C., and Bertels, K. (2013). Run-time optimization of a dynamically reconfigurable embedded system through performance prediction. *International Journal for Innovative Research in Science & Technology (IJIRST)*, pages 1–8.

[Meedeniya et al., 2010] Meedeniya, I., Buhnova, B., Aleti, A., and Grunske, L. (2010). Architecture-driven reliability and energy optimization for complex embedded systems. In *6th International Conference on Quality of Software Architectures : Research into Practice - Reality and Gaps (QoSA)*, pages 52–67. Springer.

[Midonnet et al., 2010] Midonnet, S., Masson, D., and Lassalle, R. (2010). Slack-time computation for temporal robustness in embedded systems. *IEEE Embedded Systems Letters*, 2(4) :119–122.

[Mittal, 2014] Mittal, S. (2014). A survey of techniques for improving energy efficiency in embedded computing systems. *International Journal of Computer Aided Engineering and Technology (IJCAET)*, 6(4) :440–459.

[Narayanan and Xie, 2006] Narayanan, V. and Xie, Y. (2006). Reliability concerns in embedded system designs. *Computer*, 39(1) :118–120.

[Pappalardo et al., 2017] Pappalardo, S., Alessandro, A., Santagati, C., Ardizzone, C., Ribellino, C., Mirabella, I., Principato, O., and Tonicello, F. (2017). Miniaturised integrated circuit core element for point of load (pol) conversion. *European Space Power Conference (ESPC)*, 16 :1–8.

[Ranganadh et al., 2011] Ranganadh, N., Patel, P., and Grigoryan, A. (2011). Performances of texas instruments dsp and xilinx fpgas for cooley-tukey and grigoryan fft algorithms. *Journal of Engineering and Technology*, 1(2) :83–87.

[Rasmussen and Trick, 2008] Rasmussen, R. V. and Trick, M. A. (2008). Round robin scheduling - a survey. *European Journal of Operational Research*, 188(3) :617–636.

[Rutagangibwa and Krishnamurthy, 2014] Rutagangibwa, V. and Krishnamurthy, B. (2014). A survey on the implementation of real time systems for industrial automation applications. *International Journal for Innovative Research in Science & Technology (IJIRST)*, 1(7) :174–177.

[Sangiovanni-Vincentelli, 2007] Sangiovanni-Vincentelli, A. (2007). Quo vadis, sld ? reasoning about the trends and challenges of system level design. *IEEE*, 95(3) :467–506.

[Sifakis and Henzinger, 2007] Sifakis, J. and Henzinger, T. A. (2007). The discipline of embedded systems design. *Computer*, 40 :32–40.

[Stilkerich et al., 2014] Stilkerich, S., Siemers, C., and Ristig, C. (2014). Appropriate multi-core architecture for safety-critical aerospace applications - certifiable real-time switching network. In *International Conference on Pervasive and Embedded Computing and Communication Systems (PECCS)*, pages 180–185. SciTePress.

[Vulgarakis and Seceleanu, 2008] Vulgarakis, A. and Seceleanu, C. (2008). Embedded systems resources : Views on modeling and analysis. In *38th Annual Computer Software and Applications Conference (COMPSAC)*, pages 1321–1328, Los Alamitos, CA, USA. IEEE Computer Society Press.

[Waghmare and Mohite-Patil, 2012] Waghmare, N. and Mohite-Patil, P. T. (2012). Comparison of embedded system and fpga application for universal multi-resource bus. *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)*, 1 :23–29.

[Wang et al., 2014] Wang, J., Zhong, S., Yan, L., and Cao, Z. (2014). An embedded system-on-chip architecture for real-time visual detection and matching. *IEEE Trans. Cir. and Sys. for Video Technol.*, 24(3) :525–538.

[Want et al., 2002] Want, R., Pering, T., Borriello, G., and Farkas, K. I. (2002). Disappearing hardware. *IEEE Pervasive Computing*, 1(1) :36–47.

[Xie et al., 2017] Xie, G., Chen, Y., Liu, Y., Wei, Y., Renfa, L., and Li, K. (2017). Resource consumption cost minimization of reliable parallel applications on heterogeneous embedded systems. *IEEE Transactions on Industrial Informatics*, 13(4) :1629–1640.

Chapitre 3

Les systèmes multi-agents

> "L'individu est éphémère, les races et les nations viennent et passent, mais l'Humain reste. C'est là que réside la différence profonde entre l'individu et l'ensemble."
>
> Nikola Tesla
> *Inventeur, ingénieur (1856 - 1943)*

3.1 | Définitions et spécifications

Les systèmes multi-agents sont issus de l'intelligence artificielle distribuée. Ferber indique qu'ils émergent d'une remise en question de l'informatique séquentielle en considérant que les activités sont résolues par interaction et coopération d'entités autonomes appelées agents [Ferber, 1997].

Différentes définitions de l'entité agent existent mais trois principales se démarquent et font encore aujourd'hui office de références.
Selon Ferber, un agent est défini comme "*une entité physique ou virtuelle évoluant dans un environnement dont il n'a qu'une représentation partielle et sur lequel il peut agir*" [Ferber, 1995]. Cet agent peut "*communiquer avec d'autres agents et est doté d'un comportement autonome*". A travers cette définition nous relevons quelques points clefs : l'agent se caractérise par sa capacité à communiquer avec l'environnement et les autres entités agents. Par ailleurs, l'accent est mis sur la notion d'autonomie, c'est à dire la capacité des agents à évoluer sans intervention directe de l'utilisateur.

Nous pouvons retrouver ces éléments dans la définition de Demazeau et Costa établissant un agent comme "*une entité réelle ou virtuelle dont le comportement est autonome, évoluant dans un environnement qu'il est capable de percevoir et sur lequel il est capable d'agir, et d'interagir avec les autres agents*" [Demazeau and Costa, 1996]. Ici la notion d'interaction est soulevée. Elle regroupe l'idée de communication, d'action et de perception des éléments entourant l'agent. Nous retrouvons également l'idée d'autonomie.

Enfin, selon Wooldridge un agent est "*un système informatique capable d'agir de manière autonome et flexible dans un environnement*" [Wooldridge 1999]. Il précise le terme de flexibilité avec trois éléments : la réactivité, la pro-activité et les capacités sociales. Il définit ainsi "*un système réactif maintient un lien constant avec son environnement et répond aux changements qui y surviennent*", "*un système pro-actif génère et satisfait des buts. Son comportement n'est donc pas uniquement dirigé par des événements*" et "*un système social est capable d'interagir ou coopérer avec d'autres systèmes*". Nous retrouvons toujours cette notion d'autonomie à laquelle Wooldridge présente ainsi trois caractéristiques supplémentaires aujourd'hui essentielles pour définir un agent.

L'architecture générale d'un agent [Russell and Norvig, 2009] [Figure 3.1] illustre l'interaction avec l'environnement et les autres entités agents le composant par le biais d'acteurs et de capteurs. Les choix d'action sont dictés par une notion de réaction et/ou décision de l'entité agent. Cette notion est induite par les connaissances et compétences de l'entité agent.

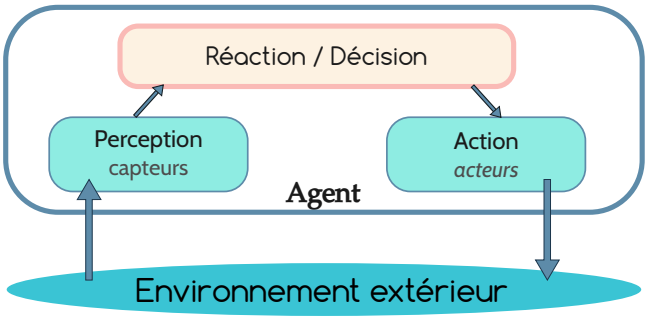


FIGURE 3.1 – *Architecture générale d'un agent.*

Nous souhaitons expérimenter les algorithmes agents au cœur de systèmes embarqués pour permettre leur optimisation. Nous sommes particulièrement intéressés par la capacité d'interaction des agents. Leurs compétences et connaissances sont mises en avant pour faire ressortir leur capacité à rendre des services. Cette caractéristique nous conduit à la définition d'un agent [Définition 10].

Définition #10 : *Agent*

Un **agent** est une entité physique ou virtuelle autonome capable d'interagir avec son environnement dont il possède une représentation partielle. Il peut également interagir avec les autres agents partageant cet environnement.
Un agent possède des compétences et connaissances propres. Il peut les combiner pour offrir des services.

Par exemple dans le cadre d'un vol d'oiseau, l'agent oiseau peut interagir avec les autres oiseaux pour décider des directions de vol. Il prend également connaissance de son environnement pour repérer les obstacles. En regroupant ces agents en systèmes multi-agents, nous obtenons des ensembles de nature distribuée capables de générer de multiples interactions. Cette capacité apporte aux systèmes multi-agents la possibilité d'évoluer en s'adaptant à leur environnement. Leur faculté à prendre des décisions et à agir sans intervention d'un tiers participe à leur autonomie [Sabouret, 2009].

Les SMA se concentrent sur une vision décentralisée de la formalisation et résolution des problèmes [Définition 11]. Par exemple, dans le cadre de simulations, des voitures sur une portion de route peuvent représenter un SMA où chaque voiture est une entité agent.

Définition #11 : *Systèmes multi-agents*

Un **système multi-agents** est un système constitué de plusieurs entités agents situées dans un même environnement et interagissant pour répondre à des requêtes ou objectifs précis.

3.1.1 Différentes catégories d'agents

Les agents peuvent être classifiés dans trois grandes catégories : les agents dit réactifs, cognitifs ou hybrides [Ferber, 1995].

- Agent **réactif**.

Comme son nom l'indique, il réagit aux stimuli perçus dans l'environnement. Ses actions sont déterminées par des règles pré-établies et les conditions perçues. Un agent réactif peut intégrer un état interne qui sera partie intégrante des conditions à analyser dans le choix de l'action à effectuer. Les règles régissant les actions sont établies selon le service que doit accomplir l'agent. Il peut s'agir d'un besoin interne (se nourrir pour un animal, maintenir son niveau énergétique pour une machine), ou d'une tâche spécifique à accomplir. Les agents réactifs ne possèdent pas de vision à long terme de leur environnement. Ils ne prévoient donc pas leurs actions futures selon une planification des changements de l'environnement. En contrepartie, ils sont robustes et fiables car ils proposent des comportements redondants basés sur des réactions à des effets de seuils. Leur empreinte mémoire est moindre et ils interagissent

rapidement avec leur environnement. Leur force vient des comportements issus de leurs interactions et permettant leur adaptation avec le SMA et leur environnement.

- Agent **cognitif**.

Il possède des raisonnements internes basés sur des buts à réaliser et une représentation de son environnement et des autres agents qui l'entourent. L'agent cognitif détient une certaine mémoire des événements passés. Il est ainsi capable de raisonner sur ce qu'il perçoit de son environnement, sur les impacts qu'auraient ses actions et sur la satisfaction qu'il tirerait de ces impacts. Il décide en fonction de ces raisonnements quelle action appliquer. Il peut ainsi mémoriser des situations et des réactions de l'environnement à ses actions. Cette faculté lui permet de planifier son comportement en prévoyant sur le long terme les fluctuations de l'environnement et les actions les plus appropriées. Les agents cognitifs sont plus gourmands en termes d'empreinte mémoire ou de temps de calcul. En effet, chaque décision passe d'abord par une étape d'analyse augmentant leur temps de réponse en comparaison des agents réactifs. Par ailleurs, il est difficile de déterminer comment l'agent évoluera à partir de sa base de règles puisqu'il apprend de son environnement. Cette spécificité peut créer des biais de répétabilité. En contrepartie, leurs capacités de planification, d'analyse et d'apprentissage permettent à certains agents d'optimiser leur comportement par rapport à leur environnement.

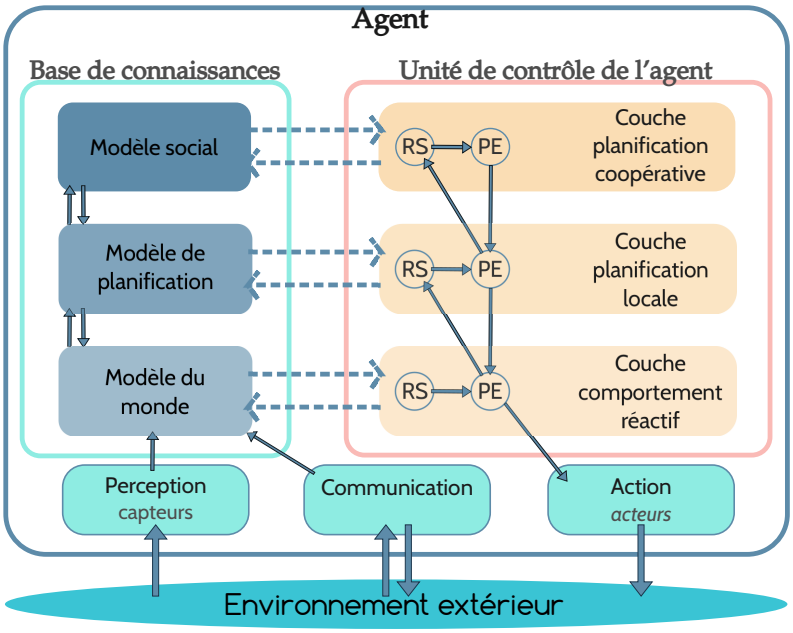


FIGURE 3.2 – *Architecture d'un agent hybride ilustrée par l'exemple InterRaP.*

- Agent **hybride**.

Un exemple d'architecture d'agent hybride est illustré en Figure 3.2 [Chin et al., 2014]. Dans cet exemple, les agents hybrides sont construits sur un modèle de couches pour assurer les deux aspects vus précédemment : réactif et cognitif. La couche la plus bas niveau correspond à une approche réactive. Elle effectue sa Reconnaissance de Situation (RS) directement à partir des données brutes de la base de connaissances fournies par les capteurs et la communication avec les autres agents. Elle est capable de prendre des décisions et d'initier des actions (Planification et Exécution, PE). Cette étape et les actions en découlant pourront être mises à jour par les couches supérieures. La couche intermédiaire travaille avec une vision plus globale de l'environnement. Sa reconnaissance de situation reste à partir de données locales. La dernière couche représente la partie cognitive. Elle ajoute les aspects sociaux aux connaissances locales. L'agent peut alors se représenter les autres agents partageant son environnement (leurs buts et connaissances, etc.). La reconnaissance de situation permet de prendre en compte le long terme dans le choix d'exécution.

Ces différentes catégories d'agents ont longtemps défini la nature de ces derniers. L'évolution des différents modèles proposant des représentations plus variées des entités agents ont amené une convergence des caractéristiques vers une notion de comportement. Cette notion permet de concentrer la description de l'entité sur ses actes sans l'astreindre à une catégorie fixe.

3.1.2 Modèles agents

Un modèle correspond à la formalisation d'un SMA. Il détermine les agents qui le composeront, leurs actions, leurs interactions, leur forme de réflexion, leur capacité de décision, etc. Nous citons ici une liste non exhaustive de modèles tout en souhaitant rester représentatifs de la diversité des approches.

Certains modèles s'appuient sur l'approche objet, comme la méthode MaSE (*Multiagent Systems Engineering*) [Wood and Deloach, 2000]. Ce modèle délimite son fonctionnement à des SMA fermés, dont l'environnement n'évolue pas. Ces environnements doivent également être statiques, à savoir composés d'agents dont l'état ne change pas durant l'exécution. Par ailleurs, le modèle considère uniquement les interactions d'un agent vers un autre agent en opposition aux interactions à multiples destinataires. MaSE se découpe en deux parties : d'abord une phase d'analyse des spécifications du SMA, puis une phase de conception guidée. La phase d'analyse comprend la spécification des objectifs du système à partir d'un cahier des charges, la formalisation des cas d'utilisation à travers des conversations inter-agents et enfin la transformation de ces cas d'utilisation en rôles en considérant les objectifs définis. La phase de conception comporte la réalisation de classes d'agents à partir des rôles définis, la réalisation des conversations formalisées, l'assemblage des différentes classes en un SMA et enfin le déploiement de ce dernier. Cette forme de modèle développe peu le concept social de l'agent du fait de la limitation des conversations à des cas d'utilisation pré-établis.

D'autres modèles choisissent plutôt de s'appuyer sur une architecture agent BDI, basée sur la psychologie du raisonnement et largement validée par la communauté de recherche [Georgeff et al., 1999]. Ce modèle repose sur trois ensembles inter-connectés : les croyances, les désirs et les engagements. La notion de croyance traduit la base de connaissances de l'agent et lui donne une certaine vision du monde auquel il appartient. Cette base de connaissances est mise à jour par l'agent grâce à la perception de son environnement et des interactions avec les autres entités agents. Ces croyances ne sont pas forcément vraies, elles peuvent être faussées par les perceptions de l'agent, ou simplement être tronquées dès le départ. Les désirs d'un agent sont générés à partir de ses croyances. Ils reflètent à la fois les opportunités de l'agent à un instant donné et ses objectifs à long terme. Ils nécessitent une ou plusieurs actions pour être remplis. L'agent va parfois décider de passer à l'acte. Il est alors question d'engagement pour signifier que le désir a été retenu. Avec ce modèle, le comportement de l'agent va évoluer en fonction de ses croyances mais aussi avec les expériences accumulées suite aux différents engagements.

Parmi les méthodes utilisant cette architecture d'agents, nous retrouvons par exemple la méthode AAII (Australian Artificial Intelligence Institute Methodology) développée pour la gestion du trafic aérien [Kinny et al., 1996]. Nous pouvons également citer la méthode DESIRE (*DEsign and Specification of Interacting REasoning framework*) issue de l'ingénierie des connaissances [Brazier et al., 1997] [Brazier et al., 2002].

Une autre méthode reprenant l'héritage des approches d'ingénieries des deux modèles précédemment cités est Gaia [Wooldridge et al., 2000]. Ce modèle, se voulant plus générique, considère les agents comme des systèmes informatiques approximatifs reliés à l'utilisation d'une ressource. L'objectif du SMA développé par Gaia est d'optimiser la qualité de certaines mesures générales même si la solution émergente n'est pas optimale d'un point de vue matériel. Par conséquent, cette méthode se décrit elle-même comme non adaptée dans le cadre de conflits de ressources. Tout comme la méthode MaSE, Gaia considère uniquement des SMA statiques. Les services et la base de connaissances sont eux spécifiés à partir des rôles et des interactions déterminés durant la phase d'analyse. Cependant elle ne prend pas en compte les systèmes devant gérer des conflits.

L'architecture agent AGR (Agent Groupe Rôle) définit les rôles joués par l'agent au sein de groupes, avec une vision centrée organisation. La méthode Aalaadin propose un cadre solide de développement de systèmes multi-agents à partir de cette forme d'architecture [Ferber and Gutknecht, 1998].

Les modèles pour les systèmes multi-agents peuvent se baser sur l'organisation des agents selon leur rôle (AAII, Aalaadin), leurs buts (MaSE) ou leurs tâches (DESIRE, Gaia). Pour nos travaux, les agents pourront être répartis sur plusieurs niveaux de l'architecture matérielle avec différentes contraintes. Il est donc important de rester indépendants des spécifications de l'agent lors de la formalisation du SMA pour garantir une meilleure adaptation à notre contexte d'application. Pour cela, nous nous intéressons aux algorithmes agents liés à leurs interactions et partons donc d'un modèle agent centré sur ce point. Deux modèles correspondent à ces attentes : DIAMOND (*Decentralized Iterative Multiagent Open Networks Design*) [Jamont and Occello, 2007] et IODA (*Interaction-Oriented Design of Agent simulations*) [Kubera et al., 2010].

3.1.2.1 DIAMOND

DIAMOND utilise le modèle AEIO (Agent Environnement Interaction Organisation) privilégiant une description explicite des interactions et de l'environnement de l'agent [Demazeau, 1995]. DIAMOND est une méthode de co-design ; elle offre une formalisation de conception hybride traitant conjointement le logiciel et le matériel. Cette pratique implique que toutes les étapes de la méthode sont pensées pour une concordance optimale entre les deux parties. La particularité de DIAMOND est d'être dédiée à la spécification de logiciels multi-agents [Jamont, 2005].

Le cycle de vie de DIAMOND est un modèle en spirale composée de plusieurs cycles, chacun découpé en quatre phases : la détermination des besoins du cycle à venir, l'analyse des risques, le développement et la validation de la solution retenue. Cette dernière phase implique la revue des résultats obtenus et la planification du cycle suivant. Ce modèle offre donc la possibilité de remettre en cause les besoins et la finalité du produit à chaque cycle. Au sein de ce cycle, une section de la méthode est dédiée à la réalisation du modèle multi-agents considéré comme ensemble logiciel. Cette partie comprend l'analyse des besoins logiciels, la conception du logiciel et l'expérimentation du logiciel. Elle reste cependant très liée au développement du reste du système pour assurer une certaine symbiose. Ainsi, les choix de développement du système embarqué permettent de combler des besoins du SMA et réciproquement.

La phase de conception générique développe la formalisation du SMA. DIAMOND y spécifie d'abord son contexte d'utilisation. Ensuite, la méthode propose la construction des tâches applicatives agent et la construction des modules de communication et des structures d'organisation. Enfin, DIAMOND détaille la construction du contrôle de l'agent. L'ensemble de cette partie s'appuie sur la documentation générée lors des précédentes phases de définition des besoins et d'analyse de DIAMOND. Notamment l'étape d'analyse étudie l'environnement dans lequel les entités évoluent, puis à l'axe agent, aux interactions, et enfin l'organisation de l'ensemble du système, selon la décomposition AEIO.

Dans la phase de conception générique, l'itération multi-agents passe également par ces quatre axes d'étude à travers les parties suivantes :

- le contexte d'utilisation spécifie des situations pour définir les limites du système multi-agents et de son environnement ;
- la construction des tâches applicatives agent détaille l'agent de façon interne et individuelle ;
- la construction des modules de communication et des structures d'organisation représente une phase sociale spécifiant les interactions et l'organisation des agents ;
- enfin, la construction du contrôle de l'agent correspond à une phase d'intégration des influences sociales au cœur des agents.

DIAMOND se concentre ensuite sur la réalisation complète d'un système embarqué adapté aux besoins physiques du système multi-agents spécifié. Pour notre part, nous souhaitons partir d'un SE existant et y associer un SMA pour proposer une nouvelle application d'optimisation au sein de ce système embarqué sans en modifier les composants. Dans le cadre de travaux joignant les systèmes multi-agents aux systèmes embarqués, notre démarche représente donc plutôt une certaine complémentarité avec l'approche DIAMOND.

3.1.2.2 IODA

IODA est un modèle multi-agents basant sa réalisation sur les interactions entre agents et leur spécification [Kubera et al., 2011]. L'avantage est de proposer une formalisation générique des communications, indépendante du contexte d'utilisation. Le modèle IODA part des Interactions et de l'Environnement pour ensuite s'intéresser à l'Organisation et enfin à l'Agent. Les contraintes de l'environnement matériel peuvent donc être prises en compte très tôt dans la procédure. L'organisation du SMA en découlera directement, notamment pour la prise en compte des méthodes de communication. Cet état de fait nous permet d'inclure dans notre démarche les besoins d'une adaptation aux SE avant de finaliser nos agents, assurant ainsi un résultat plus adéquat.

IODA commence par établir les interactions nécessaires au système pour déduire les entités agent à créer. Une interaction est définie comme un bloc d'actions, impliquant un nombre fixe d'agents. Elle décrit quand et sous quelles conditions les agents vont interagir entre eux et avec leur environnement. Les attributs de ce bloc sont listés ci-dessous :

- **une condition de déclenchement** dont la description traduit implicitement un but ;
- **des pré-conditions** décrivant les dispositions logiques ou physiques nécessaires pour réaliser l'interaction ;
- **les actions** précisant les changements à appliquer sur l'environnement et les agents concernés ;

- **le nombre d'agents** impliqués. Ce nombre est représenté par une cardinalité (n,p) où n représente le nombre d'agents sources, et p correspond au nombre d'agents cibles ;
- **une liste de fonctions** exécutées par les différents agents impliqués dans l'interaction.

Cette méthode se focalise sur des cardinalités (1,1) ou (1,0). La première correspond à une interaction entre deux agents. La deuxième représente une interaction dite "dégénérée" réalisée par un agent source sur lui même. Dans ce dernier cas, la cible est représentée par un ensemble vide ∅. A partir de la liste des activités de notre contexte, une matrice des interactions est construite pour obtenir une vue d'ensemble. Ensuite, le modèle IODA demande de spécifier les pré-requis de déclenchement. Pour ce faire, les auteurs considèrent deux paramètres : la portée et la priorité. En effet, un agent source doit être à une distance suffisante pour initier l'interaction avec l'agent cible. De plus, si plusieurs interactions étaient simultanément éligibles, elles seraient alors sélectionnées selon leur priorité.

La Table 3.1 représente un exemple de matrice mis en avant dans les travaux de recherche liés au modèle IODA : le cas d'une fourmilière [LIFL, 2011]. Nous y retrouvons les différentes familles d'agents impliquées par les interactions nécessaires à l'évolution de la fourmilière : une fourmi, la fourmilière, la source de nourriture, un morceau de nourriture et les phéromones des fourmis.
Nous pouvons constater que la famille d'agents la plus active est celle de la fourmi. Cette dernière initie à elle seule la majorité des interactions du système multi-agents. Vis-à-vis des agents de type "Source nourriture", elle peut initier trois interactions : les pister, ramasser un morceau de nourriture ou vérifier son chargement. La première interaction nécessite une distance minimale de 5 unités et possède une priorité de 6 (faible). Les deux autres interactions requièrent une distance de 0. Elles seront donc éligibles au même instant. La priorité permet d'établir une sélection : la fourmi vérifiera d'abord son chargement avant d'éventuellement prendre un morceau depuis la source de nourriture. Nous retrouvons cette notion de priorité entre deux interactions éligibles chacune sur une cible différente. Par exemple, un morceau de nourriture initiera d'abord l'interaction "réveiller fourmi" avant d'envisager l'interaction dégénérescente "mourir".

Source \ Cible	∅	Fourmi	Fourmilière	Source nourriture	Morceau nourriture	Phéromone
Fourmi	+ (mourir, 8) + (vagabonder, 1) + (se-tourner, 0)		+ (suivre, 5, 6) + (stocker, 0, 5) + (se reposer, 0, 4) + (se décharger, 0, 3)	+ (pister, 5, 6) + (ramasser, 0, 5) + (déjàChargé, 0, 3)		+ (générer, 0, 7) + (suivre, 3, 2)
Fourmilière	+ (produireFourmi, 1) + (relacherEclaireur, 0)					
Source nourriture	+ (mourir, 0)					
Morceau nourriture	+ (mourir, 1)		+ (réveillerFourmi, 0, 0)			
Phéromone	+ (mourir, 1)					+ (disperser, 1, 0)

Table 3.1 – *Matrice des interactions d'un SMA simulant le fonctionnement d'une fourmilière selon le modèle IODA.*

Une fois les pré-requis établis, l'étape suivante consiste à lister les actions découlant de chaque interaction répertoriée, avant de les formaliser sous forme de fonctions à intégrer aux agents impliqués. Enfin, IODA formalise les architectures agents à partir de ces ensembles de fonctions en prenant en considération les points suivants :
- **une primitive d'initialisation** pour définir le démarrage de l'agent ;
- **un halo** déterminant comment l'agent perçoit les agents voisins ;
- **une matrice de mise à jour** pour décrire comment un agent évolue indépendamment des interactions auxquelles il participe ;
- **les primitives des actions** correspondant à la matrice des interactions pour cet agent en tant que source.

Ce modèle permet donc de rester indépendant du type d'agent et de son emplacement matériel lors de la spécification du SMA. C'est pourquoi nous avons développé nos architectures d'agents embarqués en nous inspirant de la matrice des interactions IODA.

3.1.3 Plateformes multi-agents

Une fois le système multi-agents formalisé, sa gestion amène plusieurs questions à régler :

- **du point de vue de l'agent :**
  - Comment interagir avec l'environnement ?
  - Selon quelles règles communiquer avec les autres agents ?
  - Quel support pour accomplir les buts assignés à l'agent ?

- **du point de vue de l'utilisateur :**
  - Comment déployer le SMA ?
  - Quelle administration pour les agents ?
  - Comment interagir avec le système ?

Une plateforme multi-agents permet de résoudre ces questions en apportant des outils de développement pour le fonctionnement du SMA et en lui servant d'environnement d'exécution. Elle assure une bibliothèque de fonctions pour les communications et interactions des agents, une structure organisationnelle pour le SMA, voire une interface de visualisation du fonctionnement des agents.

Il existe aujourd'hui une multitude de plateformes multi-agents développées pour correspondre à une famille de SMA spécifique. Les plateformes multi-agents proposeront le plus fréquemment une liste de critères eux-mêmes classés par types, tels que :

- **l'usage de la plateforme** : facilité d'utilisation, compatibilité avec les standards, choix pour la communication, etc.
- **la capacité de la plateforme** : son niveau de performance, sa stabilité, sa robustesse, le langage de programmation utilisé, la sécurité proposée, etc.

Par exemple, une comparaison des plateformes les plus couramment utilisées pour étudier les choix de développement en se focalisant sur l'avantage de la généricité est proposée [Bordini et al., 2006]. D'autres travaux présentent une étude de comparaison focalisée sur les spécificités des plateformes dédiées selon leur langage de programmation, le système d'exploitation nécessaire au fonctionnement de la plateforme, leur type de licence, le domaine d'application ciblé par la plateforme et le support utilisateur offert [Nikolai and Madey, 2009]. Plus récemment, plusieurs plateformes ont été comparées avec la même expérimentation [Kravari and Bassiliades, 2015]. Même dans le cadre de comparaisons selon un critère spécifique (comme par exemple le cas des agents mobiles [Gupta and Kansal, 2011], ou de plateformes parallèles [Herrmann et al., 2014]) permettant des comparaisons d'utilisation et de performances plus directes, le but reste d'observer les particularités de chaque plateforme plutôt que les placer en opposition.

Pour donner un aperçu du panel de plateformes multi-agents existantes, nous proposons ici d'en présenter quelques-unes parmi les plus utilisées.

Certaines plateformes sont pensées pour la simulation. C'est le cas notamment de JADE (*Java Agent DEvelopment Framework*), dédiée aux développements en langage de programmation Java [Bellifemine et al., 2005]. Respectueuse de la norme FIPA (*Foundation for Intelligent Physical Agents*), elle se veut très générique. Elle peut être distribuée entre différentes machines et sa configuration peut être modifiée durant l'exécution via l'interface graphique. JADE présente ainsi une solution compatible dans des ensembles hétérogènes.

Autre plateforme multi-agents orientée simulation et largement utilisée, NetLogo se base sur les concepts du langage de programmation Logo [Carbo et al., 2016]. Cette plateforme a été conçue pour fournir une base éducative servant à traiter des concepts complexes. Elle peut également servir à développer des applications sophistiquées grâce à son environnement graphique de création de SMA et de contrôle de ces derniers. NetLogo est ainsi adaptée pour les experts comme les débutants. La plateforme propose d'ailleurs beaucoup d'exemples d'applications. Contrairement à JADE, elle ne recherche pas forcément à être adaptée à divers contextes d'utilisation et se concentre sur la diversité de ses cas d'application.

Dans le cadre des plateformes orientées simulation, on trouvera également des modèles dédiés à un modèle agent. C'est le cas par exemple de la plateforme JEDI (*Java Environment for the Design of agent Interactions*) permettant d'implémenter et d'expérimenter le modèle IODA détaillé en section 3.1.2.2 [Picault, 2013]. Développée en langage Java, la plateforme permet notamment via une interface graphique intuitive de réaliser une matrice d'interactions

personnalisée. JEDI intègre un environnement de développement JEDI-Builder pour générer le SMA résultant de cette matrice et l'exécuter en simulation. Tout comme NetLogo, la plateforme permet une vision directe d'une modification de la matrice et est adaptée à des cas d'utilisation d'apprentissage comme d'expertise.

En plus d'une approche orientée simulation, certaines plateformes visent en particulier un cadre de simulation réaliste. Ainsi, la plateforme GAMA souhaite apporter aux experts un environnement de simulation spécialisé dans la réalisation spatialement explicite avec une interface graphique 3D, de la modélisation multi-niveaux et la gestion de données d'information géographique [Grignard et al., 2013]. Cette plateforme permet le développement de larges SMA en GAML, un langage orienté agent. Cette plateforme a notamment été utilisée pour simuler des évacuations de bâtiments ou de villes face à une catastrophe naturelle (incendie, tsunami, etc.) [Adam et al., 2017].

Pour une adaptation aux systèmes embarqués, il est fréquent de préférer les langages de programmation dits "bas niveau" car possédant une couche d'abstraction plus faible. L'emploi de ce type de langage permet d'optimiser son intégration en adéquation avec l'architecture de la machine, comme par exemple le cas de la gestion mémoire souvent inclus dans la couche d'abstraction pour les langages de haut niveau.

Certaines plateformes multi-agents proposent un développement en langage de programmation C, considéré bas-niveau. C'est le cas par exemple de Swarm, une plateforme multi-agents logicielle gérant des SMA adaptatifs en proposant leur spécification en langage C++ [Minar et al., 1996]. Swarm est également dédiée à la réalisation de simulations de systèmes multi-agents. La plateforme supporte des collections d'agents indépendants interagissant selon un planning d'événements. Ces collections sont appelées des swarms. La plateforme permet l'implémentation de n'importe quel modèle s'il est spécifié sous forme de swarms. Un swarm peut également être considéré lui-même comme un agent. Cette possibilité permet à la plateforme de gérer des SMA multi-niveaux et donc des modèles hiérarchiques.

La plateforme MaDKit (*multi-agents development kit*) [Gutknecht and Ferber, 2000] est pensée pour offrir un développement de systèmes multi-agents suivant l'organisation AGR et le modèle agent AALAADIN, mentionnés en section 3.1.2. D'abord utilisée spécifiquement en langage Java, elle propose également la possibilité de développer des SMA en langage C pour assurer un service applicatif hétérogène. La plateforme permet ainsi une utilisation adaptée aux nombreux domaines applicatifs des systèmes multi-agents de la simulation jusqu'à une utilisation dans les systèmes embarqués pour le domaine de la robotique. MaDKit propose une architecture modulaire comportant un agent "micro-kernel" regroupant les fonctions d'usage d'un agent évoluant sur la plateforme. Les applications développées s'appuient sur cet agent micro-kernel. La plateforme propose également une application graphique permettant une représentation visuelle de l'ensemble.

Dans le cadre de nos travaux, notre choix de plateforme est orienté par plusieurs critères. Nous souhaitons expérimenter un modèle inspiré de IODA, nous considérons donc les modèles agents supportés par la plateforme pour vérifier leur compatibilité avec ce modèle. Le langage de programmation représente un critère essentiel car nous nous plaçons dans un contexte bas-niveau et utilisons donc le langage C. Par ailleurs, nous considérons le domaine d'application de la plateforme : il doit permettre une application en embarqué et donc ne pas se focaliser sur la simulation. Enfin, nous nous situons dans un contexte hétérogène et souhaitons donc considérer une certaine normalisation pour la plateforme multi-agents utilisée.

A travers la Table 3.2, nous proposons un récapitulatif des plateformes multi-agents présentées dans cette section selon nos critères. Nous y indiquons par l'utilisation d'une couleur de fond les points forts des plateformes considérées par rapport à notre cadre d'étude. Nous tenons à rappeler que les plateformes présentées ici ne représentent qu'un échantillon de l'ensemble des plateformes multi-agents existantes. Nous avons cependant étudié selon ces même critères de nombreuses autres plateformes.

Plateforme	Modèle agent supporté	Langage de programmation	Domaine d'application	Normalisation
JADE	Indifférent	Java	Applications multiples	FIPA
NetLogo	Indifférent	NetLogo	Simulations multiples	/
JEDI	IODA	Java	Simulations multiples	/
GAMA	Indifférent	GAML	Spatialement réaliste	FIPA
Swarm	Indifférent	Java, C++	Simulations multiples	/
MaDKit	AALAADIN	Java, C, Python	Applications multiples	UML

Table 3.2 – *Plateformes multi-agents existantes selon les critères impactants dans le cadre de nos travaux.*

Nous constatons que les plateformes multi-agents existantes étudiées se concentrent sur d'autres cadres de travaux ne correspondant pas parfaitement à nos critères. La plateforme JADE est la plus adéquate d'après les critères considérés, cependant le langage de programmation est l'un de nos critères majeurs du fait de notre contexte d'étude. Dans le cadre de ce travail, nous proposons donc le développement de notre propre plateforme multi-agents adaptée à notre contexte d'étude.

3.2 | Normalisation des échanges

Les systèmes multi-agents et les systèmes embarqués possèdent chacun leurs propres ontologies. L'ontologie représente une manière de décrire les informations d'un domaine. Au sein même des systèmes multi-agents, toutes les plateformes ne possèdent pas la même ontologie. Or une représentation commune des données échangées est nécessaire pour établir une communication, a fortiori dans un contexte d'utilisation hautement hétérogène.

Pour définir une plateforme multi-agents embarquée, il est nécessaire d'établir une ontologie commune pour la mise en adéquation des messages des agents et des protocoles de communication embarqués tels que ceux détaillés en section 2.4.3. Ces derniers assurent la normalisation des échanges entre les processus et les différents SE. Il nous reste donc le choix d'une norme pour les communications multi-agents.

Deux propositions de standards existent :
- KQML (*Knowledge Query and Manipulation Language*) ;
- FIPA-ACL (*Foundation for Intelligent Physical Agents - Agent Communication Language*).

3.2.1 Norme KQML

KQML est un langage de haut niveau facilitant la communication entre agents logiciels. Développé dans les années 90, il fut récemment mis à jour pour répondre à des besoins d'abstraction de l'implémentation des systèmes multi-agents [Jagga et al., 2015]. Orienté message, ce protocole emploie la théorie des actes de langage, c'est à dire l'association de chaque message à un moyen d'agir sur l'environnement de l'initiateur. Dans KQML, nous retrouvons cette notion à travers les performatives. KQML propose diverses performatives regroupées en 6 catégories. Nous présentons ces catégories dans la Table 3.3.

KQML n'engage pas l'utilisation d'un mécanisme de transport ni d'un langage déterminé et d'une ontologie spécifiée pour le contenu d'un message. Il propose trois couches d'encapsulation pour un message. La première couche est celle du contenu, elle implique de préciser le langage du contenu ainsi que son ontologie. La couche suivante concerne le message dans sa globalité. Elle précise le type de message c'est à dire la performative, ainsi que les données du message. Enfin, la couche dite de communication détaille les informations concernant le transfert du message telles que le destinataire, l'expéditeur et l'adresse de retour en cas de réponse.

Compte tenu de nos besoins, l'application de ce standard nous pousserait à trouver une autre norme pour l'administration de notre plateforme. FIPA-ACL, pour sa part, est intégrée à une norme étendue à l'ensemble de la gestion d'un système multi-agents : FIPA.

Catégories de performatives	Description	Exemple
Les annonces (*Inform*)	Cette performative est utilisée pour agir sur la base de connaissance de l'interlocuteur en lui passant une information.	La performative "dire" (tell) permet à un agent initiateur d'indiquer quel service il possède à un agent destinataire.
Les requêtes (*Request*)	Lorsqu'un agent souhaite qu'un autre agent effectue un travail pour lui, il lui soumet une requête.	La performative "accomplir" (*achieve*) est utilisée pour demander à un autre agent d'effectuer une action donnée.
Les demandes (*Query*)	Semblables aux requêtes, les demandes impliquent un retour d'information de la part de l'agent destinataire.	La performative "demander tout" (*ask-all*) implique que l'agent destinataire renvoie toutes ses réponses à la question contenue dans le message.
Les réponses (*Reply*)	Après l'envoi d'une demande, une réponse est attendue et sera envoyée par le biais d'un message avec cette performative.	La performative "Envoyer tout" (*stream-all*) correspond à la performative de réponse à une demande de type "demander tout".
Les promesses (*Promise*)	A mi-chemin entre une information et une réponse, les promesses permettent d'indiquer à un instant donné la capacité d'un agent à effectuer une action.	La performative "avertir" (*advertise*) indique à un destinataire sa capacité à exécuter l'action du contenu.
Les flux (*Meta*)	Les flux sont des performatives d'indication et de mise à jour du SMA.	La performative "abonnement" (*subscribe*) indique à l'agent destinataire qu'il doit mettre à jour ses réponses à une performative donnée.

Table 3.3 – *Catégories de performatives supportées par le protocole de communication inter-agents KQML.*

3.2.2 Fondation FIPA et norme FIPA-ACL

La fondation FIPA propose l'établissement de standards dès 1996 pour favoriser l'utilisation d'agents logiciels dans des contextes industriels hétérogènes. Elle vise l'interopérabilité entre SMA développés par différentes compagnies et portés par différentes plateformes. En 2002, FIPA complète 25 spécifications [FIPA, 2002]. Le standard propose tout d'abord la spécification d'une architecture abstraite ainsi que les relations entre les éléments composant cette architecture [spécification FIPA00001]. Elle regroupe également des guides d'instanciation d'agents basés sur l'architecture FIPA ainsi que des guides d'interopérabilité pour l'implémentation de FIPA. Cette spécification permet de décrire trois éléments en particulier :

- **l'administration** : une architecture de gestion du SMA dont la spécification reste abstraite pour permettre une implémentation ouverte à tout langage de programmation ;
- **la communication** : la spécification des messages avec la norme FIPA-ACL et de nombreux protocoles d'interaction ;
- **le transport** : le routage des messages avec une couche d'abstraction appelée MTPS (*Message Transport Protocol Service*).

La norme FIPA propose une structure en plusieurs blocs. Nous présentons la hiérarchie de ces spécifications dans la Figure 3.3 [Smarsly et al., 2012]. Nous y retrouvons la notion d'une architecture générique avec des détails pour

l'administration de la plateforme, la communication entre les agents et le transport des messages. La couche d'applications fait référence aux agents logiciels développés par l'utilisateur de la plateforme multi-agents. Son positionnement en couche supérieure de la figure représente son appui sur l'architecture FIPA.

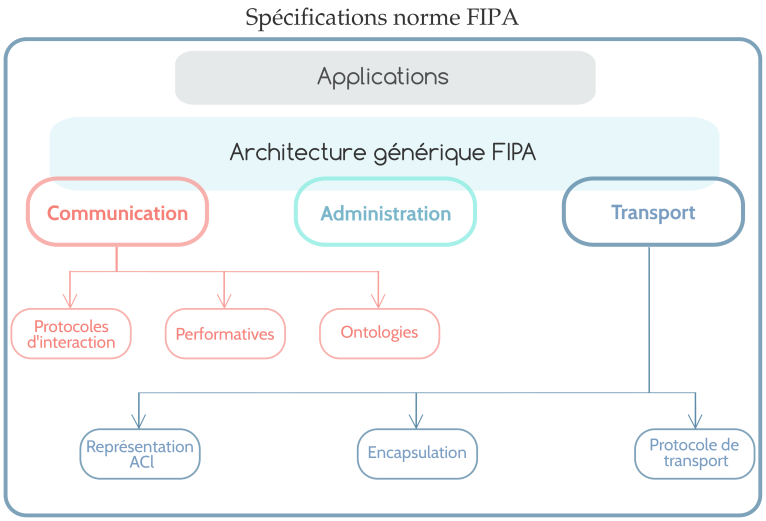


Figure 3.3 – *Structure hiérarchique des spécifications FIPA.*

Nous présentons tout d'abord les spécifications liées à la communication avec le standard FIPA-ACL. Ensuite nous détaillons la partie administrative d'une plateforme multi-agents suivant la norme FIPA. Enfin nous décrivons les spécifications relatives au transport des messages.

FIPA-ACL est un standard de communication entre agents proposant une syntaxe similaire à KQML pour standardiser un ensemble de protocoles d'interaction [Poslad, 2007]. Cette norme offre également une encapsulation des messages, bien que plus détaillée, et s'appuie sur le principe des performatives pour spécifier l'impact de ses messages. A l'instar de KQML, FIPA-ACL est une norme indépendante du mécanisme de routage du message ainsi que du langage et de l'ontologie de son contenu.

Le standard regroupe une spécification de la structure des messages, un ensemble de performatives détaillant l'impact de chaque message et des protocoles d'interactions. Ces derniers précisent une suite d'échanges de performatives, formant ainsi une conversation complète, pour achever une forme spécifique d'interaction. FIPA-ACL compte actuellement 22 performatives de base qu'on peut regrouper en cinq catégories selon leur objectif global. Nous les détaillons en Table 3.4 [spécification FIPA00037].

Actuellement FIPA propose ainsi neuf protocoles d'interaction. Par exemple, le protocole d'interaction de requête (*FIPA Request Interaction Protocol*) est présenté en Figure 3.4 [spécification FIPA00026]. Nous y retrouvons un agent initiateur de l'interaction et un agent participant. Plusieurs agents peuvent être destinataires d'une interaction d'où l'utilisation du terme "participant". L'agent initiateur envoie sa demande par le biais d'un message possédant la performative *request*. Nous avons donc ici affaire à une demande d'action. L'agent participant va traiter la requête et prendre la décision de l'accepter ou de la refuser. Il notifie cette décision à l'agent initiateur par le biais des performatives *refuse* ou *agree*. Si la proposition est refusée, l'interaction se termine. Si l'agent participant a accepté la demande, il effectue la ou les actions liées à cette requête. En cas d'erreur, l'agent participant notifie l'agent initiateur du "non aboutissement" de sa demande par la performative de gestion d'erreur *failure*. Si tout s'est bien déroulé, l'agent participant emploie les performatives de passage d'information pour indiquer à l'agent initiateur la complétion de la demande. Il utilisera *inform-done* si la demande ne nécessitait pas de retour d'information spécifique, *inform-result* sinon.

Enfin au niveau de la structure des messages, FIPA-ACL propose une encapsulation composée de 13 paramètres [spécification FIPA00061]. Parmi eux, 5 doivent être obligatoirement complétés. Nous précisons leur fonction en Table 3.5. Les autres paramètres représentent des champs de complétion optionnels et sont détaillés en Table 3.6. La notion d'option, dans ce cadre, indique que le message doit tout de même comporter ce paramètre mais qu'il est possible de ne pas lui attribuer de valeur.

Catégorie	Description	Performatives
Passage d'information	Ces performatives sont utilisées pour transférer des connaissances parfois sous certaines conditions.	*inform, inform-if, inform-ref, inform-done, inform-result, confirm, disconfirm*
Demande d'information	Ces performatives sont utilisées pour faire des requêtes impliquant un transfert de certaines connaissances en réponse.	*query-if, query-ref, subscribe*
Négociation	Les performatives de négociation sont employées dans le cadre de protocoles d'interaction dédiés au principe de négociation.	*accept-proposal, cfp, propose, reject-proposal*
Demande d'action	Un agent peut demander à un autre agent d'effectuer une action spécifique parfois sous certaines conditions.	*request, request-when, request-whenever, agree, cancel, refuse*
Gestion d'erreurs	Deux performatives sont dédiées à la gestion des erreurs intervenant lors d'une interaction.	*failure, not-understood*

Table 3.4 – *Performatives proposées par le standard de communication FIPA-ACL.*

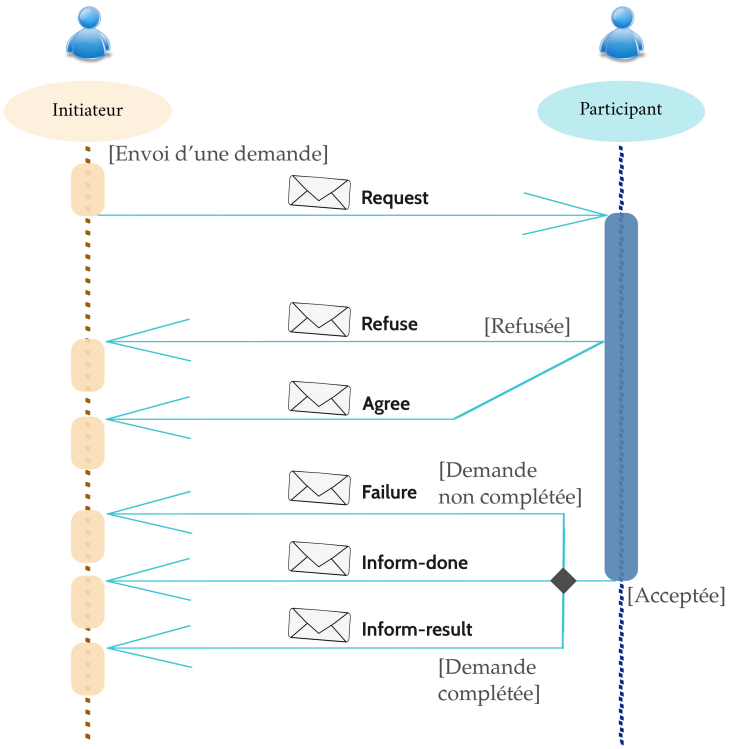


Figure 3.4 – *Diagramme du protocole d'interaction FIPA dans le cadre d'une demande (request).*

Paramètre	Description
Performative	L'usage des performatives permet d'indiquer l'impact souhaité du message sur l'environnement de l'agent. Elle permet également un suivi lors de l'utilisation d'un protocole d'interaction défini.
Expéditeur (*sender*)	L'identité de l'expéditeur du message, c'est à dire son identifiant. Ce paramètre permet une réponse directe si besoin.
Destinataire (*receiver*)	L'identité du ou des destinataire(s) du message. Normalement ce paramètre est obligatoire, mais il est possible de le laisser vide lors de messages adressés à tous.
Répondre-à (*reply-to*)	Ce paramètre permet d'indiquer si une éventuelle réponse doit être renvoyée vers un autre agent. Si ce n'est pas le cas ce champ comporte la même information que le champ "expéditeur".
Contenu (*content*)	Ce paramètre correspond au message d'origine avant son encapsulation selon la structure FIPA-ACL. Les paramètres suivants intègrent des informations sur ce contenu.

Table 3.5 – *Performatives obligatoires du standard de communication FIPA-ACL.*

3.2.3 Administration

La partie d'administration de la plateforme utilise les spécifications de FIPA-ACL. FIPA détaille un référencement des agents selon un identifiant AID (*Agent IDentifier*) permettant de distinguer chaque entité agent au sein du SMA [FIPA00023]. Cet identifiant se compose d'un nom et d'une "adresse", c'est à dire les informations nécessaires pour le contacter par message.

Pour la gestion d'une plateforme multi-agents, la norme FIPA propose la mise en place de trois agents administratifs : AMS, DF et ACC [consortium, 2003].

- **AMS**

Le système de gestion d'agents AMS (*Agent Management System*) exerce le contrôle de supervision sur la plateforme. Un SMA ne doit pas comporter plus d'un agent AMS. Il tient une liste des agents présents et actifs du système multi-agents de manière semblable à la tenue d'un annuaire. Cette liste se complète au fur et à mesure grâce aux agents : lors de leur initialisation, ils informent AMS de leur disponibilité via leur nom et leur adresse. Lorsqu'un agent n'est plus actif il en informe également AMS et ce dernier le retire de sa liste. Ce service est assuré par deux fonctions d'AMS : l'enregistrement (*register*) et le désenregistrement (*deregister*). Les agents peuvent également contacter AMS pour lui demander de mettre à jour les informations les concernant (statut ou adresse) par le biais de la fonction de modification (*modify*). Enfin, l'agent AMS met à disposition une fonction de recherche permettant d'obtenir l'adresse d'un agent donné (*search*).

Selon l'implémentation de la norme FIPA dans une plateforme multi-agents, AMS peut se voir octroyer des fonctions de contrôle supplémentaires. Par exemple, il est possible de lui permettre de demander à un agent de s'arrêter (*suspend agent*), voire de forcer son arrêt en cas de nécessité (*terminate agent*). Une autre option est de lui donner la possibilité d'ajouter des agents dans le système (*create agent*). Toutes ces possibilités doivent rester cohérentes avec la fonction principale de l'agent AMS : le contrôle du SMA.

Paramètre	Description
Langage (*language*)	Le langage est celui dans lequel le contenu du message a été exprimé. Si l'expéditeur est certain que le destinataire emploie le même langage que lui, ce paramètre n'a pas besoin d'être renseigné.
Encodage (*encoding*)	Si le contenu du message est encodé, ce paramètre sert à renseigner le destinataire sur l'encodage appliqué. Cette précision peut également être présente dans l'encapsulation de la couche de transport.
Ontologie (*ontology*)	La précision de l'ontologie du contenu est directement reliée au langage utilisé pour compléter sa compréhension. Si le langage n'a pas été spécifié ce paramètre n'est pas utilisé.
Protocole (*protocol*)	Si ce message s'inscrit dans une suite de messages déterminée par l'un des protocoles d'interaction proposé par FIPA-ACL, ce champ permet d'indiquer le protocole utilisé.
Identifiant de conversation (*conversation-id*)	Lors du suivi d'un protocole, tous les messages générés entre deux agents participants doivent posséder le même identifiant de conversation. Autrement, ce paramètre est optionnel.
Répondre-avec (*reply-with*)	Ce paramètre peut être employé lors d'échanges entre deux agents suivant plusieurs conversations en parallèle. Il permet de préciser le sujet de la conversation et fonctionne avec le paramètre suivant.
En-réponse-à (*in-reply-to*)	Ainsi dans le cadre de conversation parallèle, si un agent envoie un message avec une indication pour le champ "répondre-avec", l'autre répondra avec cette même indication pour le champ "en-réponse-à".
Répondre-avant (*reply-by*)	Ce paramètre indique un délai souhaité, en temps ou en date, à ne pas dépasser pour la réception d'une réponse de la part du destinataire.

Table 3.6 – *Performatives optionnelles du standard de communication FIPA-ACL.*

- **DF**

Le facilitateur d'annuaire DF (*Directory Facilitator*) tient une liste des services actifs sur la plateforme et des agents auxquels ils sont reliés. Le maintien de cette liste permet de savoir à tout instant quelles capacités sont à disposition sur la plateforme. Il est possible d'avoir plusieurs agents DF au sein d'un SMA. La répartition des capacités sur les différentes listes peut alors se faire selon leur type, ou bien selon leur localisation si le SMA s'étend à plusieurs machines. D'autres critères de répartition peuvent être choisis.

A l'instar d'AMS, DF propose des fonctions d'enregistrement, de désenregistrement et de modification des informations sauvegardées. Les agents l'informent de leurs capacités lors de leur activation et se désenregistrent à leur terminaison. L'agent DF apporte également une fonction de recherche pour obtenir l'identité des agents proposant un service donné. Lorsque plusieurs agents DF sont présents sur un même SMA et que la fonction recherche de l'agent DF contacté ne donne pas de résultat, il va étendre cette recherche aux autres agents DF du système.

Il est important de noter que l'agent DF ne fait qu'indiquer quel agent fournit quel service. Il ne peut garantir que l'agent ensuite contacté sera en état de réaliser ce service à un instant donné. Nous résumons ce point par le fait que l'agent DF offre une liste des capacités existantes, mais pas forcément une liste des capacités disponibles. Une option de délai de validité d'une capacité peut être mise en place pour s'approcher d'une liste à jour. Cette option implique cependant que les agents reviennent régulièrement informer DF des capacités qu'ils proposent, sans pour autant assurer qu'ils seront en état de la réaliser lorsqu'ils seront contactés. Elle reste pertinente dans le cadre d'une longue période d'exécution.

- **ACC**

Enfin, le canal de communication entre agents ACC (*Agent Communication Channel*) fournit un service fiable et précis pour le routage des messages à destination d'autres agents. Il peut y avoir plusieurs agents ACC au sein d'un même système multi-agents. La capacité de l'agent ACC permet aux agents logiciels composant la plateforme de ne pas devoir enregistrer les informations des uns ou des autres. Lorsqu'un agent veut en contacter un autre, il envoie son message à ACC avec une "enveloppe" contenant les informations nécessaires à l'identification du destinataire. Le message à transférer contient au minimum les champs de performative, d'expéditeur, d'agent à contacter en retour et du contenu avec les diverses options y étant rattachées. L'agent ACC utilise ensuite les fonctions de recherche des agents DF et AMS pour compléter les paramètres du message à transférer à partir des informations confiées dans l'enveloppe. Enfin il envoie le message complété au destinataire d'origine. Cette gestion relève de l'encapsulation, la diffusion de l'ensemble des messages du système à travers la couche de transport est gérée par la couche MTPS.

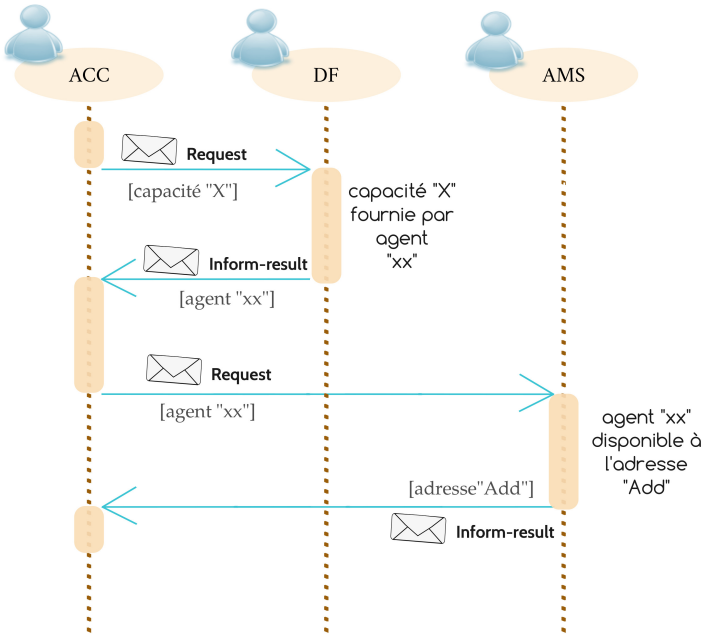


FIGURE 3.5 – *Diagramme d'échanges entre les agents ACC, DF et AMS pour compléter les paramètres d'un message confié à ACC pour être transféré.*

La récupération des informations auprès des agents AMS et DF implique des requêtes. Il est préconisé de suivre le protocole d'interaction de demande, présenté en section 3.2.2, mais ce n'est pas une obligation.
En Figure 3.5, nous présentons un exemple d'interaction entre les agents ACC, DF et AMS visant à compléter les paramètres d'un message. Dans le cadre de cet exemple, l'enveloppe envoyée à ACC contenait l'indication de la capacité "X" du destinataire. L'agent ACC utilise donc d'abord la fonction de recherche de l'agent DF pour obtenir l'identification de l'agent proposant cette capacité. L'agent DF traite la demande en parcourant sa liste, il renvoie ensuite l'identifiant de l'agent fournissant ce service : "xx". Avec cette information, l'agent ACC peut ensuite contacter l'agent AMS afin d'obtenir l'adresse "Add" qu'il pourra indiquer en paramètre "destinataire" du message à transférer. Dans cet exemple, la complétion des paramètres implique la génération de quatre messages. Si on choisit de suivre le protocole d'interaction d'une demande, le nombre de messages passe alors (dans le cas d'une demande complétée) à un total de six. L'envoi concret du message est ensuite passé à la couche de transport de la plateforme.

3.2.4 Transport des messages

FIPA est un standard indépendant du protocole de transport utilisé par la plateforme multi-agents l'appliquant. Elle propose tout de même un guide d'architecture pour favoriser l'interopérabilité à travers la spécification du protocole de transmission des messages MTPS [FIPA00067]. Ce protocole spécifie la nécessité d'appliquer une abstraction entre la notion de transport et le protocole effectuant ce transport. Au niveau de la gestion de l'administration, c'est l'agent ACC qui se charge de la gestion de la notion de transport en préparant l'encapsulation complète d'un message pour son transfert. L'application concrète du transport est gérée par la couche MTPS. Cette couche analyse les paramètres d'encapsulation et utilise les protocoles de transport à sa disposition pour effectuer le transfert concret. Par exemple, la couche MTPS peut considérer les champs d'expéditeur et de destinataire pour vérifier les adresses indiquées par ces champs. Si les adresses renvoient à la même machine, le protocole de transport utilisé pourra être différent d'un cas d'envoi entre deux machines différentes. Ce choix reste opaque pour les agents logiciels et administratifs. Par ailleurs, si l'encapsulation du message nécessite de nouveaux paramètres en fonction du protocole de transfert utilisé, c'est également le rôle de la couche MTPS d'assurer la correspondance de la structure du message.

Le standard FIPA nous permet de normaliser la communication de nos agents. Les algorithmes d'interactions employés par ces agents relèvent de leurs comportements sociaux.

3.3 | Algorithmes de comportements sociaux multi-agents

Les systèmes multi-agents ont de fortes capacités d'adaptation et d'auto-organisation leur permettant de former des systèmes hétérogènes évolutifs. Régulièrement, du fait de l'exécution simultanée des actions des différents agents, des relations synergiques ou conflictuelles peuvent émerger. La nature de ces relations dépend de la compatibilité des buts des différents agents, de leur capacité à accomplir ces buts et du nombre de ressources disponibles pour cette réalisation [Ferber, 1997].

Nous pouvons ainsi déterminer le type de comportements sociaux qu'adopteront les agents en suivant l'arbre de décision présenté en Figure 3.6.

La compatibilité des **buts** est éprouvée par les actions nécessaires pour achever ces derniers. Si les actions sont complémentaires, les buts seront déterminés comme compatibles. Si, à l'inverse, les actions entrent en conflit, ils sont déterminés comme incompatibles. Par exemple, si le but d'un agent est de sortir d'une pièce fermée, il devra en ouvrir la porte. Si le but d'un autre agent est d'empêcher quelqu'un d'entrer, il devra s'assurer que la porte reste fermée. Les buts de ces deux agents sont incompatibles car ils créent un conflit sur les actions relatives à la porte.
Nous évaluons ensuite les **ressources** considérées. Sont-elles suffisantes pour l'ensemble des agents ou impliquent-elles une gestion supplémentaire ? Par exemple, pour nos deux agents ayant des buts incompatibles, les ressources liées à leurs buts sont les portes. Si il n'y a qu'une seule porte les agents entreront obligatoirement en conflit, la ressource est alors "insuffisante".

Enfin, nous estimons les **compétences** des agents indépendamment de l'état de compatibilité des buts et des ressources. Nous devons pour ce faire vérifier si les agents ont ou non les capacités d'effectuer toutes les actions nécessaires à l'accomplissement de leurs buts. Par exemple, si nous reprenons l'exemple lié aux buts, nous vérifierons que le premier agent a la capacité d'ouvrir la porte pour sortir. Nous évaluerons également la capacité du deuxième agent à pouvoir détecter l'ouverture de la porte et savoir la refermer si besoin.
Suivant l'établissement de ces trois facteurs, nous pouvons déterminer la nature des relations sociales qui auront cours au sein du SMA. Si les buts sont compatibles, les ressources et les compétences suffisantes, alors le SMA sera composé d'agents totalement indépendants n'ayant pas besoin d'établir de relations sociales particulières.
Dans le cadre de buts compatibles mais de ressources insuffisantes ou de compétences insuffisantes, les agents seront dans une optique de coopération pour atteindre leurs objectifs. Les interactions peuvent alors établir cette coopération et sont qualifiées d'interactions de coordination [Bergenti and Ricci, 2002] [Subagdja and Tan, 2014]. Pour des buts incompatibles mais des ressources suffisantes, des relations de compétition émergeront, sensiblement différentes selon les compétences des agents. Si les ressources sont également insuffisantes les relations seront conflictuelles. Les agents doivent alors procéder à des interactions pour résoudre le problème posé tout en satisfaisant l'intérêt respectif de chaque agent, c'est à dire qu'un accord mutuellement acceptable est recherché. Ces interactions sont qualifiées de négociation [Calvaresi et al., 2018].

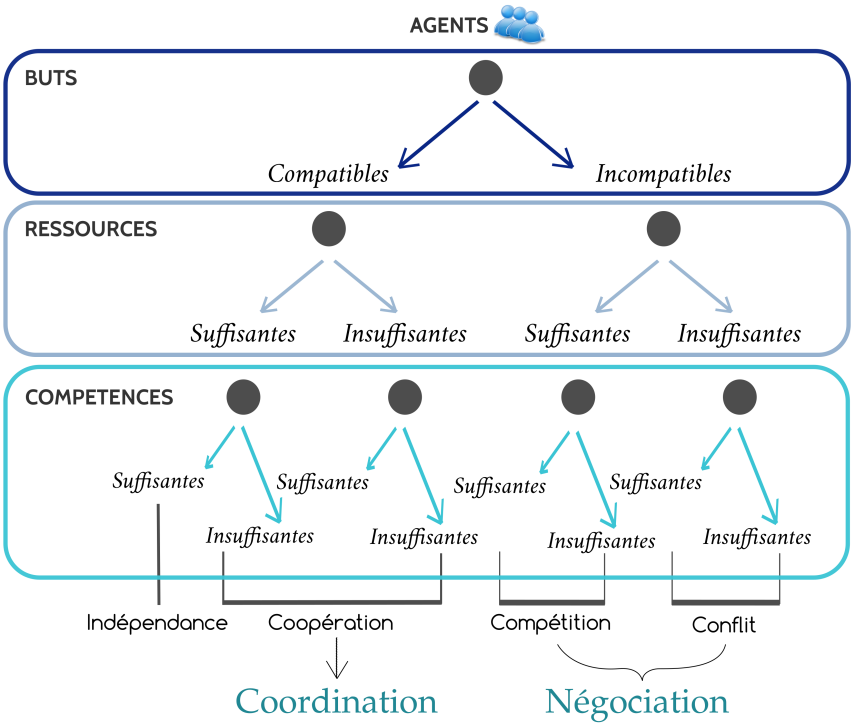


FIGURE 3.6 – *Diagramme de détermination des comportements sociaux en fonction des buts, des ressources et des compétences des agents concernés.*

Les protocoles de négociation et de coordination des systèmes multi-agents permettent de faire émerger une solution optimale en peu de temps, dans des environnements dynamiques.

3.3.1 Coordination et négociation

La coordination des agents peut être considérée comme un ensemble d'actions supplémentaires à accomplir en coopération avec les autres agents du SMA. Ces actions correspondent à une programmation, d'une part de l'utilisation des ressources communes, d'autre part d'un enchaînement de compétences pour permettre l'accomplissement des buts compatibles.

Pour établir cette programmation, il existe quatre types de coordination : par synchronisation, par planification, par réglementation et par réaction. La synchronisation correspond à des protocoles généralement implémentés à l'avance. Les agents échangent de nombreuses informations de manière à détecter les conditions d'application du protocole de synchronisation. La planification peut être la charge d'un seul agent qualifié de superviseur. Autrement, elle est distribuée entre les différents agents, chacun proposant une planification, avec une mise en commun permettant de faire émerger une entente globale. Cette dernière se base sur des "promesses d'action" de la part des agents. Le principe de réglementation correspond à un ensemble de règles connues par tous les agents leur permettant de savoir quel comportement adopter vis-à-vis des autres à l'instar d'un code de la route par exemple. Enfin, une coordination réactive permet de s'adapter à des agents eux-même réactifs avec un principe de stimuli/réponse.

En termes de temps de réponse, seule la coordination par planification peut être qualifiée de "lente". Elle est cependant la meilleure solution en termes de prédictibilité, c'est à dire d'organisation à long terme. La synchronisation ne représente un avantage que pour sa rapidité à obtenir une solution [Coates et al., 2000]. Cette dernière reste efficace sur le court terme et laisse la place à peu d'adaptabilité. Quant à la coordination par règlement, il s'agit d'une méthode relativement efficace offrant un bon compromis.

Les travaux utilisant la coordination sont variés. Par exemple, dans le domaine de la robotique, nous retrouvons ces algorithmes sur le sujet de la sécurité avec la plateforme RMASBench. Cette dernière fut développée afin de servir de référence en matière de coordination dans le cadre de la recherche et du sauvetage urbain [Kleiner et al., 2013]. D'autres travaux se basent sur une coordination locale basée sur le regret, issue de la norme de Tchebychev, dans un contexte de simulations de crise de la RoboCupRescue [Boussard et al., 2007]. Dans le domaine de la gestion de l'énergie pour des bâtiments intelligents, on retrouve également l'utilisation d'algorithmes de coordination multi-agents se basant sur une approche centralisée [Stavropoulos et al., 2014], ou pour diminuer directement les coûts en consommation de l'électricité [Zheng et al., 2014].

La négociation est un moyen d'amener les agents à résoudre un état de conflit ou de compétition à travers la communication. Cette approche permet à chaque agent de faire valoir son point de vue et ainsi de satisfaire au mieux les intérêts respectifs de chacun. La négociation permet, elle, une prise en compte des objectifs personnels pour atteindre un accord avec le meilleur compromis. Il existe deux formes de négociation agent : compétitive ou coopérative. Pour la première, les agents vont négocier en partageant des informations afin de tenter des choix de groupe pour chaque action conflictuelle. Ils évaluent alors l'utilité des propositions en considérant jusqu'où ils sont prêts à "payer" pour débloquer la situation par rapport au "coût" de la proposition. Un ensemble de négociation correspond alors à tous les accords ayant une utilité positive pour chaque agent et représentant autant de solutions possibles au problème. Cette forme de négociation peut se baser sur la théorie des jeux [Pereau, 2009], les heuristiques [Paletta and Herrero, 2010], la vente aux enchères [Lavendelis and Grundspenkis, 2014] ou encore sur l'argumentation [Rahwan and Dignum, 2009]. La négociation coopérative implique, elle, une collaboration des agents impliqués. Dans ce cadre, chaque agent apporte des offres directement au prix le plus élevé pour eux. Ils considèrent même la possibilité d'avoir une utilité négative dans une certaine mesure afin de débloquer le problème. Divers protocoles suivent cette approche, comme par exemple une proposition de protocole liée à une méthode d'ordonnancement de lignes de production [Berrandjia et al., 2015]. Ce type de négociation reste principalement appliqué avec le protocole d'interaction de "réseau contractuel" (*Contract Net Protocol*, CNP).

3.3.2 Protocole de négociation

Le protocole Contract Net est un standard de communication pour des agents asynchrones où un agent initiateur fait face à un besoin et envoie des requêtes à d'autres agents pour obtenir de l'aide. Le protocole se découpe en quatre étapes : l'Annonce, l'Offre, l'Attribution et le Contrat [Smith, 1980]. Lors de la phase d'Annonce, l'agent initiateur envoie un message à tous les autres agents dits contractants, c'est à dire en capacité d'établir un contrat pour cette demande. Ce message d'annonce se compose d'une description de la demande, de la formalisation du type d'offre devant être retournée, c'est à dire les spécifications liées à cette réponse, et un délai d'expiration à partir duquel la demande n'est plus d'actualité. L'étape d'Offre correspond aux réponses des agents contractuels intéressés et en capacité de fournir une offre.
La troisième étape, l'Attribution, correspond à la décision de l'agent initiateur. Parmi les réponses reçues, il choisit celle qu'il estime être la meilleure selon ses critères de décision propre. Il informe l'agent contractuel à l'origine de cette réponse que son offre a été retenue. Enfin, l'étape de Contrat scelle l'accord établi entre les deux agents et leur permet d'échanger de plus amples informations si nécessaire avant mise en application du contrat.

Ce protocole de négociation a connu plusieurs propositions d'évolution dont une version formalisée par la fondation FIPA : le protocole d'interaction Contract Net [Figure 3.7 [FIPA00029]] et le Protocole Itératif d'interaction Contract Net (CNIP) [Et-Tolba et al., 2015]. Comme nous avons choisi de suivre le standard FIPA pour la réalisation de notre plateforme, nous choisissons de respecter cette version du protocole.

CNP implique un agent initiateur réalisant un appel à propositions (*Call for proposals*, Cfp) à destination de m agents participants. Ce premier contact correspond à l'étape d'Annonce. Lors de l'étape d'Offre, n agents envoient une réponse avant le délai d'expiration. Pour que le protocole se poursuive, il faut obtenir un nombre d'offres j, équivalent au nombre réponses n soustrait du nombre de retours négatifs i, positif. Ensuite, FIPA établit l'étape d'Attribution en informant chaque agent contractant du devenir de sa proposition, acceptée ou refusée, avec l'utilisation des performatives Reject-proposal et Accept-proposal. Ici pour que le protocole se poursuive, il faut un nombre de propositions acceptées l positif. Enfin, nous retrouvons l'étape de Contrat où les agents participants dont la proposition a été acceptée notifient l'agent initiateur du résultat.

La version itérative du protocole établit la possibilité de répéter l'opération d'appel à propositions, avec une demande mise à jour en fonction des offres reçues, lors de l'étape d'Attribution [FIPA00030]. Le nombre d'itérations

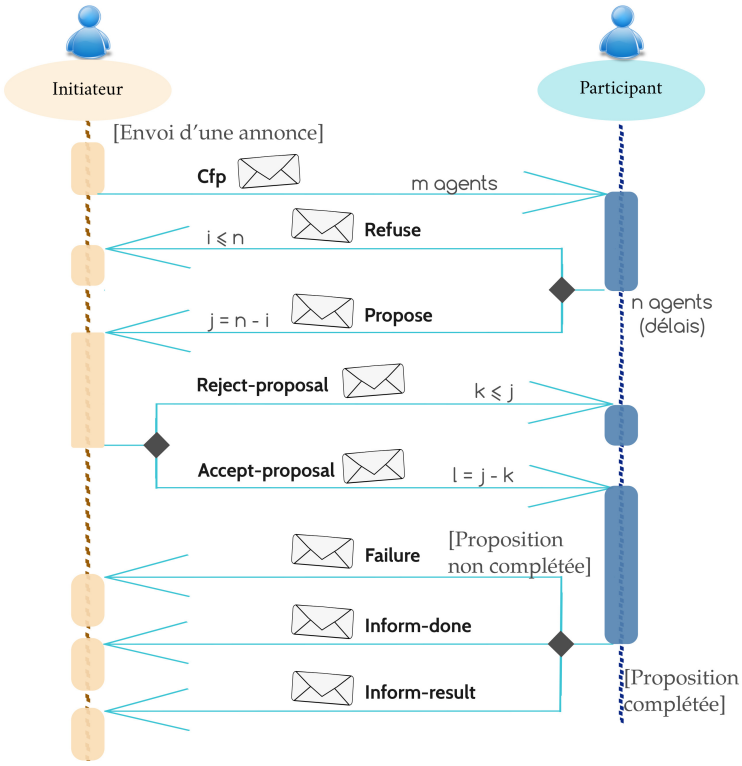


FIGURE 3.7 – *Diagramme du protocole d'interaction Contract Net FIPA utilisé dans le cadre de négociations.*

est déterminé au départ et les nouveaux Cfp se font à destination des agents contractants dont la proposition de l'itération en cours a été acceptée. Cette version permet une sélection plus précise pour une négociation reposant sur de nombreux paramètres.

Nous retrouvons l'utilisation du protocole CNP dans de nombreux travaux comme, par exemple, dans le domaine de la robotique où il est utilisé pour répartir les tâches entres différents robots [Liekna et al., 2012]. D'autres travaux l'ont utilisé pour simuler un environnement évoluant rapidement [Kaur et al., 2013]. Ils ont relevé certaines limites dans ce cadre au niveau du dynamisme du protocole. En effet, une fois l'étape de Contrat terminée, l'initiateur ne peut pas modifier sa requête auprès du contractant. Il doit relancer un nouveau CNP. Ces travaux proposent ainsi une modification du protocole pour permettre de nouveaux échanges dans la phase de Contrat avec modification possible de l'Annonce.
Pour nos travaux nous cherchons à déléguer des tâches complètes, notre Annonce ne sera donc pas modifiée. Nous n'intégrons donc pas cette modification.

3.4 | Évaluation des travaux dans les systèmes multi-agents

Dans le domaine des systèmes multi-agents, nous pouvons distinguer trois étapes de recherche : la conception ou formalisation d'un modèle, d'une plateforme et enfin d'un SMA dédié à une application spécifique. Chaque étape peut être évaluée indépendamment. A travers notre travail de recherche, nous proposons la conception d'une plateforme multi-agents et de plusieurs systèmes multi-agents applicatifs. Nous nous intéressons donc tout particulièrement à leurs critères d'évaluation.

Pour l'évaluation d'une plateforme, certains travaux proposent une analyse des performances à travers le temps moyen de transfert d'un échange de messages entre deux agents désigné par l'acronyme RTT (*Round-Trip Time*), ainsi qu'une estimation de l'impact engendré par cet échange, c'est à dire l'éventuelle surcharge de la plateforme [Jurasovic et al., 2006]. Cette méthode d'évaluation permet, à travers diverses expérimentations impliquant différentes tailles de messages, de déterminer la limite de transfert de données d'une plateforme et d'ainsi comparer plusieurs plateformes. D'autres travaux ont ajouté à ces évaluations de performance la particularité de rechercher

volontairement à surcharger un agent destinataire pour analyser l'impact sur le RTT [Mulet et al., 2006]. Ces travaux proposent également une évaluation d'un service spécifique à la norme FIPA : l'évaluation de l'agent administratif DF en évaluant les temps de réponse des fonctions d'enregistrement, de désenregistrement et de recherche selon le nombre d'agents sur la plateforme. Le principe d'évaluation repose sur le contact d'un agent administratif et considère DF et AMS comme équivalents sur ce point. Il peut donc s'appliquer sur l'un ou l'autre.

Pour le SMA, l'évaluation est effectuée par une comparaison selon deux possibilités. La première correspond à une comparaison avec une autre approche logicielle pour réaliser le même travail. Par exemple dans le cadre d'une application de gestion de l'énergie, on comparera les résultats et performances obtenus par le SMA avec les algorithmes utilisés dans le domaine. La deuxième possibilité implique de comparer notre solution avec un autre SMA existant capable de réaliser la même application. Par exemple la réalisation d'un nouveau système multi-agents pour simuler une évacuation d'urgence sera comparée aux SMA d'évacuation déjà reconnus dans le domaine afin de faire ressortir les apports de la nouvelle solution.

Dans les deux cas, ils pourront être évalués selon trois critères fréquemment employés lors des travaux de comparaison : leur performance, leur stabilité et leur capacité d'extension [Bonnet and Tessier, 2008]. Dans le cadre de la performance, on retrouve les indicateurs du domaine des systèmes embarqués : le temps de réponse pour un calcul ou une communication, la consommation des ressources CPU ou mémoire, etc. Pour la stabilité, c'est l'état d'équilibre du SMA qui est évalué. Les expérimentations impliquent d'initier un événement pour affecter le SMA. Son degré de stabilité est ensuite déterminé en fonction de sa capacité à revenir à son état d'équilibre et du temps nécessaire pour y parvenir. Pour comparer plusieurs SMA, le même événement est appliqué à tous et son ampleur est augmentée au fur et à mesure. Enfin, l'évaluation de la capacité d'extension d'un système repose d'une part sur sa possibilité d'accueillir de nouveaux agents en cours d'exécution et d'autre part sur l'évolution de sa performance en fonction de ces nouveaux agents.

Pour ce travail de thèse, nous déterminons le RTT de notre plateforme et évaluons les temps de réponse relatifs aux services de l'agent AMS. Pour nos SMA, nous comparons les performances obtenues en temps de réponse avec des programmes correspondant à d'autres approches.

Bibliographie

[Adam et al., 2017] Adam, C., Dugdale, J., and Garbay, C. (2017). Modélisation multi-agent de la cohésion sociale en situation de crise. In *Journées francophones sur les systèmes multi-agents (JFSMA)*, pages 1–4. Cépaduès Éditions.

[Bellifemine et al., 2005] Bellifemine, F., Bergenti, F., Caire, G., and Poggi, A. (2005). *Jade — A Java Agent Development Framework*, pages 125–147. Springer US.

[Bergenti and Ricci, 2002] Bergenti, F. and Ricci, A. (2002). Three approaches to the coordination of multiagent systems. In *2002 ACM Symposium on Applied Computing (SAC)*, pages 367–372, New York, NY, USA. ACM.

[Berrandjia et al., 2015] Berrandjia, M. L., Ourari, S., and Chalal, R. (2015). Un protocole de négociation pour l'ordonnancement distribué multi-agents d'un atelier job-shop. In *Séminaire National en informatique à Biskra (SNIB)*, page 1–8.

[Bonnet and Tessier, 2008] Bonnet, G. and Tessier, C. (2008). Evaluer un système multiagent physique - retour sur expérience. In *Systèmes Multi-Agents, Communautés virtuelles et naturelles - Journées francophones sur les systèmes multi-agents (JFSMA)*, pages 13–22. Cépaduès Éditions.

[Bordini et al., 2006] Bordini, R. H., Braubach, L., Dastani, M., El, A., Seghrouchni, F., Gomez-sanz, J. J., Leite, J., Pokahr, A., and Ricci, A. (2006). A survey of programming languages and platforms for multi-agent systems. *Informatica*, 30(1) :33–44.

[Boussard et al., 2007] Boussard, M., Bouzid, M., and Mouaddib, A.-I. (2007). La décision multi-critère pour la coordination locale dans les systèmes multi-agents. *ANNALES DU LAMSADE*, 8.

[Brazier et al., 1997] Brazier, F. M. T., Dunin-keplicz, B. M., and Jennings, N. R. (1997). Desire : Modeling multi-agent systems in a compositional formal framework. In *International Journal of Cooperative Information Systems*, volume 6, pages 67–94. World Scientific Publishing Co.

[Brazier et al., 2002] Brazier, F. M. T., Jonker, C. M., and Treur, J. (2002). Principles of component-based design of intelligent agents. *Data and Knowledge Engineering*, 41(1) :1–27.

[Calvaresi et al., 2018] Calvaresi, D., Appoggetti, K., Lustrissimini, L., Marinoni, M., Sernani, P., Dragoni, A. F., and Schumacher, M. (2018). Multi-agent systems' negotiation protocols for cyber-physical systems : Results from a systematic literature review. In *10th International Conference on Agents and Artificial Intelligence (ICAART 2018)*, pages 224–235.

[Carbo et al., 2016] Carbo, J., Sanchez-Pi, N., and Molina, J. M. (2016). Agent-based simulation with netlogo to evaluate ambient intelligence scenarios. *Journal of Simulation*, pages 1–10.

[Chin et al., 2014] Chin, K. O., Gan, K. S., Alfred, R., Anthony, P., and Lukose, D. (2014). Agent architecture : An overview. *Transactions on Science and Technology*, 1(1) :18–35.

[Coates et al., 2000] Coates, G., Whitfield, R. I., Duffy, A. H. B., and Hills, B. (2000). Coordination approaches and systems – part ii : An operational perspective. *Research in Engineering Design*, 12(2) :73 – 89.

[consortium, 2003] consortium, F. (2003). *FIPA Communicative Act Library, Specification and FIPA ACL Message Structure Specification*. Technical report.

[Demazeau, 1995] Demazeau, Y. (1995). From interactions to collective behaviour in agent-based systems. In *1st European Conference on Cognitive Science*, pages 117–132.

[Demazeau and Costa, 1996] Demazeau, Y. and Costa, A. C. R. (1996). Populations and organizations in open multi-agent systems. In *1st National Symposium on Parallel and Distributed AI*, pages 1–13.

[Et-Tolba et al., 2015] Et-Tolba, E. H., Ouassaid, M., and Maaroufiand, M. (2015). An implementation of fipa contract net interaction protocol adapted for smart home agents simulation. pages 1083–1087. IEEE Computer Society Press.

[Ferber, 1995] Ferber, J. (1995). *Les Systèmes multi-agents : vers une intelligence collective*. InterEditions.

[Ferber, 1997] Ferber, J. (1997). Les systèmes multi-agents : un aperçu général. *Technique et Science Informatiques*, pages 979–1012.

[Ferber and Gutknecht, 1998] Ferber, J. and Gutknecht, O. (1998). A meta-model for the analysis and design of organizations in multi-agent systems. pages 128–135, Paris, France. IEEE Computer Society Press.

[FIPA, 2002] FIPA (2002). Fipa standard status specifications. http://www.fipa.org/repository/standardspecs.html. Accessed : 2018-04-16.

[Georgeff et al., 1999] Georgeff, M. P., Pell, B., Pollack, M. E., Tambe, M., and Wooldridge, M. (1999). The belief-desire-intention model of agency. In *5th International Workshop on Intelligent Agents V, Agent Theories, Architectures, and Languages*, pages 1–10, London, UK, UK. Springer.

[Grignard et al., 2013] Grignard, A., Taillandier, P., Gaudou, B., Vo, D. A., Huynh, N. Q., and Drogoul, A. (2013). Gama 1.6 : Advancing the art of complex agent-based modeling and simulation. In Springer, editor, *Pacific Rim international Conference on Multi-Agents (PRIMA)*, volume 8291, pages 117–131, Dunedin, New Zealand.

[Gupta and Kansal, 2011] Gupta, R. and Kansal, G. (2011). A survey on comparative study of mobile agent platforms. *International Journal of Engineering Science and Technology (IJEST)*, 3(3) :1943–1948.

[Gutknecht and Ferber, 2000] Gutknecht, O. and Ferber, J. (2000). The madkit agent platform architecture. In *In Agents Workshop on Infrastructure for Multi-Agent Systems*, pages 48–55. Springer.

[Herrmann et al., 2014] Herrmann, B., Lang, C., and Rousset, A. (2014). Étude comparative des plateformes parallèles pour systèmes multi-agents. pages 1–12, Neuchâtel, Suisse.

[Jagga et al., 2015] Jagga, A., Juneja, D., and Singh, A. (2015). Updates in knowledge query manipulation language for complex multiagent systems. *International Journal of Computing Academic Research (IJCAR)*, 4(1) :19 – 26.

[Jamont, 2005] Jamont, J.-P. (2005). *DIAMOND : Une approche pour la conception de systèmes multi-agents embarqués*. PhD thesis, Institut National Polytechnique de Grenoble - INPG.

[Jamont and Occello, 2007] Jamont, J.-P. and Occello, M. (2007). Designing embedded collective systems : The diamond multiagent method. In *19th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 91–94, Washington, DC, USA. IEEE Computer Society Press.

[Jurasovic et al., 2006] Jurasovic, K., Jezic, G., and Kusek, M. (2006). A performance analysis of multi-agent systems. *International Transactions on Systems Science and Applications (ITTSA)*, 1(4) :335–342.

[Kaur et al., 2013] Kaur, S., Kaur, H., and Sehra, S. K. (2013). Modification of contract net protocol(cnp) : A rule-updation approach. *International Journal of Advanced Computer Science and Applications (IJACSA)*, 4(11) :40–46.

[Kinny et al., 1996] Kinny, D., Georgeff, M., and Rao, A. (1996). A methodology and modelling technique for systems of bdi agents. In *European Workshop on Modelling Autonomous Agents in a Multi-agent World : Agents Breaking Away : Agents Breaking Away (MAAMAW)*, pages 56–71, Secaucus, NJ, USA. Springer.

[Kravari and Bassiliades, 2015] Kravari, K. and Bassiliades, N. (2015). A survey of agent platforms. *Journal of Artificial Societies and Social Simulation (JASSS)*, 18(1) :11–28.

[Kubera et al., 2010] Kubera, Y., Mathieu, P., and Picault, S. (2010). Everything can be agent ! pages 1547–1548. International Foundation for Autonomous Agents and Multiagent Systems.

[Kubera et al., 2011] Kubera, Y., Mathieu, P., and Picault, S. (2011). Ioda : An interaction-oriented approach for multi-agent based simulations. *Journal of Autonomous Agents and Multiagent Systems (AAMAS)*, 23(3) :303–343.

[Lavendelis and Grundspenkis, 2014] Lavendelis, E. and Grundspenkis, J. (2014). Multi-agent auction based simulation tool for an insurance policy market. *Applied Computer Systems*, 15(1) :5–13.

[LIFL, 2011] LIFL (2011). Ioda ants. http://www.lifl.fr/SMAC/projects/ioda/ioda/models/iodaants/index.html. Accessed : 2018-04-16.

[Minar et al., 1996] Minar, N., Burkhart, R., Langton, C., and Askenazi, M. (1996). The swarm simulation system : A toolkit for building multi-agent simulations. Technical report, Santa Fe Institute.

[Mulet et al., 2006] Mulet, L., Such, J. M., and Alberola, J. M. (2006). Performance evaluation of open-source multiagent platforms. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1107–1109, New York, NY, USA. ACM.

[Nikolai and Madey, 2009] Nikolai, C. and Madey, G. (2009). Tools of trade : A survey of various agent based modeling platforms. *Journal of Artificial Societies and Social Simulation (JASSS)*, 12(2) :2–64.

[Paletta and Herrero, 2010] Paletta, M. and Herrero, P. (2010). A mas-based negotiation mechanism to deal with saturated conditions in distributed environments. *International Conference on Agents and Artificial Intelligence (ICAART)*, 2 :1–7.

[Pereau, 2009] Pereau, J. C. (2009). Négociation et théorie des jeux : les « dessous » d'un accord acceptable. *Négociations*, 12(2) :37–51.

[Picault, 2013] Picault, S. (2013). *De la simulation multi-agents à la simulation multi-niveaux. Pour une réification des interactions*. Habilitation à diriger des recherches, Université des Sciences et Technologie de Lille - Lille I.

[Poslad, 2007] Poslad, S. (2007). Specifying protocols for multi-agent systems interaction. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 2(4) :51–75.

[Rahwan and Dignum, 2009] Rahwan, I. and Dignum, F. (2009). Formal analysis of interest-based negotiation. *Annals of Mathematics and Artificial Intelligence*, 55(3) :253–276.

[Russell and Norvig, 2009] Russell, S. J. and Norvig, P. (2009). *Artificial Intelligence : A Modern Approach (3rd Edition)*, chapter number 2 : Intelligent agents, pages 34–59. Prentice Hall.

[Smarsly et al., 2012] Smarsly, K., Law, K. H., and Hartmann, D. (2012). A multi-agent-based collaborative framework for a self-managing structural health monitoring system. *Journal of Computing in Civil Engineering*, 26(1) :76–89.

[Smith, 1980] Smith, R. G. (1980). The contract net protocol : High-level communication and control in a distributed problem solver. *Transactions on Computers*, C-29(12) :1104–1113.

[Stavropoulos et al., 2014] Stavropoulos, T. G., Rigas, E. S., Kontopoulos, E., Bassiliades, N., and Vlahavas, I. (2014). A multi-agent coordination framework for smart building energy management. In *25th International Workshop on Database and Expert Systems Applications (DEXA)*, pages 126–130. IEEE Computer Society Press.

[Subagdja and Tan, 2014] Subagdja, B. and Tan, A.-H. (2014). On coordinating pervasive persuasive agents. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1467–1468, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

[Wood and Deloach, 2000] Wood, M. and Deloach, S. A. (2000). An overview of the multiagent systems engineering methodology. In *The First International Workshop on Agent-Oriented software Engineering (AOSE)*, pages 207–222. Springer.

[Wooldridge et al., 2000] Wooldridge, M., Jennings, N. R., and Kinny, D. (2000). The gaia methodology for agent-oriented analysis and design. *Journal of Autonomous Agents and Multiagent Systems (AAMAS)*, 3 :285–312.

[Zheng et al., 2014] Zheng, R., Xu, Y., Chakraborty, N., and Sycara, K. (2014). Multiagent coordination for demand management with energy generation and storage. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1587–1588. International Foundation for Autonomous Agents and Multiagent Systems.

Synthèse

Au cours de ces deux chapitres, nous avons présenté un état de l'art de nos deux domaines de travail : les systèmes embarqués et les systèmes multi-agents.

Nous avons ainsi pu préciser les quatre défis régissant le domaine des systèmes embarqués : leur fiabilité, leur sécurité, leur réactivité et leur optimisation. Nous avons particulièrement détaillé ce dernier, directement lié à la problématique fixée. Nous avons décrit les différentes unités ressources du domaine ainsi que leur organisation au sein des SE, puis nous avons précisé le mécanisme de délégation des tâches que nous souhaitons appliquer. Nous avons de plus présenté les mécanismes d'allocations existant pour nous positionner par rapport à notre solution.

Au niveau des systèmes multi-agents, nous avons détaillé les différents modèles d'agents en détaillant en particulier les modèles DIAMOND et IODA. Le premier propose une approche complémentaire de notre travail en spécifiant une méthode de co-design pour les SMA en embarqué. Pour notre part, nous partons d'un SE existant, aux composants fixes, auquel nous appliquons les algorithmes multi-agents. Nous travaillerons donc en nous inspirant principalement du deuxième, le modèle centré interactions IODA. Lors de ce chapitre, nous avons offert un aperçu des nombreuses plateformes multi-agents existantes et avons spécifié nos critères concernant le choix de l'une d'entre elles. L'étude des spécificités de chacune nous a permis de constater que les plateformes existantes considérées ne correspondaient pas à nos besoins. Nous proposerons donc dans les chapitres à venir une plateforme multi-agents adaptée aux SE pour notre travail de recherche. Nous avons ensuite présenté en détail le standard FIPA et notamment sa proposition FIPA-ACL concernant les actes de langage ainsi que les agents administratifs AMS, DF et ACC. Cette norme sera appliquée pour la proposition de notre plateforme afin d'assurer son interopérabilité. Enfin, nous avons présenté les différents algorithmes de négociation permettant de gérer des conflits sociaux et en particulier le protocole de réseau contractuel CNP que nous expérimenterons pour l'optimisation de notre SE

Nous allons à présent développer nos propositions concernant la formalisation de systèmes multi-agents embarqués et l'élaboration de notre plateforme MERMAID.

Partie II

Propositions transversales

A travers l'état de l'art [Partie I], nous avons détaillé le domaine des systèmes embarqués. Nous avons distingué les aspects spécifiques des SE ainsi que leurs principaux besoins à prendre en compte pour la suite de nos travaux. Nous avons également identifié les caractéristiques des agents et des SMA puis présenté différents modèles agents et plateformes multi-agents existantes.

Pour cette deuxième partie, nous proposons une formalisation de systèmes multi-agents embarqués en accord avec les particularités du domaine. Nous montrons comment ces besoins ont été spécifiés et aboutissons ensuite à la formalisation d'une nouvelle plateforme multi-agents embarquée, MERMAID.

Dans le cadre de nos recherche, nous travaillons avec des systèmes embarqués aux composants fixés et dans un contexte potentiellement hétérogène aussi bien du point de vue matériel que du côté des SMA avec lesquels nous pourrions être amenés à communiquer. En effet, nous prenons en considération l'expansion de l'Internet des objets : notre environnement s'enrichit de plus en plus de systèmes embarqués connectés. Bien que les interactions de notre SMA ne soient pas nécessairement reliées au réseau Internet, il est tout à fait intéressant d'envisager pouvoir intégrer des agents au sein de diverses IoT pour permettre le partage de tâches.
Lors de nos études, nous n'avons pas trouvé de plateformes correspondant à notre axe de recherche. Par ailleurs, l'adaptation des nombreux protocoles de communication agents à un contexte limité en ressources est encore peu étudié. Ces aspects sont au centre de notre proposition MERMAID.

Dans notre démarche, nous reprenons les spécificités du domaine multi-agents pour présenter nos agents et notre plateforme tout en formalisant une adéquation aux critères des systèmes embarqués. La partie se découpe en deux chapitres. Nous détaillons tout d'abord les caractéristiques de nos systèmes multi-agents pour une adaptation à l'embarqué. Ensuite, nous formalisons notre plateforme avec son architecture, son administration et son organisation. Nous effectuons une évaluation de MERMAID pour achever notre proposition.

Chapitre 4

Formalisation d'un SMA embarqué

"J'aime apprendre. C'est un art et une science."

Katherine Johnson
Physicienne, mathématicienne et ingénieure spatiale (1918 -)

A travers le chapitre 2, nous avons présenté les principaux défis régissant le domaine des systèmes embarqués. Nous y avons notamment détaillé les contraintes de ressources limitées propres à ces systèmes. Dans le chapitre 3, nous avons décrit les caractéristiques des agents : leur comportement, leur autonomie, leur capacité d'interaction, leurs objectifs et leurs compétences.

En chapitre 1 nous avons spécifié notre problématique. Elle questionne l'apport d'une adaptation dynamique au contexte d'utilisation d'un SE par l'intégration de modèles multi-agents.
Pour cette intégration nous proposons de formaliser des systèmes multi-agents embarqués. Nous avons alors soumis une première hypothèse de recherche pour guider nos choix de formalisation :

Rappel hypothèse #1

L'approche centrée interaction de la formalisation de nos agents embarqués nous permettra de conserver une forme de généricité dans notre proposition.

En section 3.1.2 nous avons présenté différents modèles agents existants. Parmi eux, nous nous sommes particulièrement intéressés aux modèles centrés interaction. Ces derniers laissent plus de liberté de conception pour nos agents et facilitent une adaptation aux spécificités des systèmes embarqués.

Dans ce chapitre, nous expliquons nos choix de formalisation pour nos agents évoluant dans un contexte embarqué. Nous reprenons les caractéristiques agents et les adaptons aux contraintes matérielles. Nous présentons particulièrement leur cadre d'évolution et leurs spécificités individuelles au sein du SMA. Ensuite, nous proposons une formalisation d'étapes de réalisation d'un SMA embarqué.

4.1 | SMA embarqué

Pour définir un SMA embarqué, nous détaillons d'abord son environnement et les entités avec lesquelles il doit évoluer.
Les agents embarqués le composant regroupent les caractéristiques des agents [Definition 10] et des processus [Définition 7]. Nous définissons ainsi le terme de processus agent [Définition 12].

Définition #12 : *Processus agent*

Un **processus agent** propose plusieurs services réalisés en parallèle. Autonome, il interagit avec d'autres entités pour combiner des compétences et connaissances au sein d'un ou plusieurs systèmes embarqués

Les processus agents composant le SMA évoluent sur le SE avec les autres entités présentes dans leur environnement. Le terme entité peut désigner :

- un **agent** ;
- un **processus** ;
- une **ressource** [Définition 3] ;
- un **objet électronique** capable d'interagir avec le système embarqué. Les objets électroniques englobent les systèmes embarqués mais ne présentent pas forcément les mêmes contraintes.

Les agents embarqués peuvent interagir avec les autres entités partageant leur environnement. Nous définissons deux environnements en particulier :

- l' **environnement immédiat** d'un agent : il correspond à l'ensemble des processus, des ressources et des autres agents partageant le même système embarqué.
- l'**environnement hétérogène** de l'agent : l'ensemble du contexte en interaction avec son système embarqué, comprenant les utilisateurs, d'autres systèmes embarqués et leurs agents, ou tout autre objet avec lequel une interaction peut être établie. Cet environnement inclut l'environnement immédiat.

Sur la Figure 4.1, nous considérons un agent évoluant sur un système embarqué, de type Set-Top Box pour cet exemple, avec d'autres agents. Son environnement immédiat est le SE avec ses ressources, ses processus non agents et les autres agents présents sur ce même système. La Set-Top Box est reliée par raccordement réseau à d'autres objets électroniques dont certains embarquent également des agents. L'environnement hétérogène de l'agent considéré est composé de l'ensemble de ces systèmes embarqués distants et des agents présents sur certains. Ainsi, le système multi-agents peut être réparti sur différents contextes matériels partageant le même environnement hétérogène.

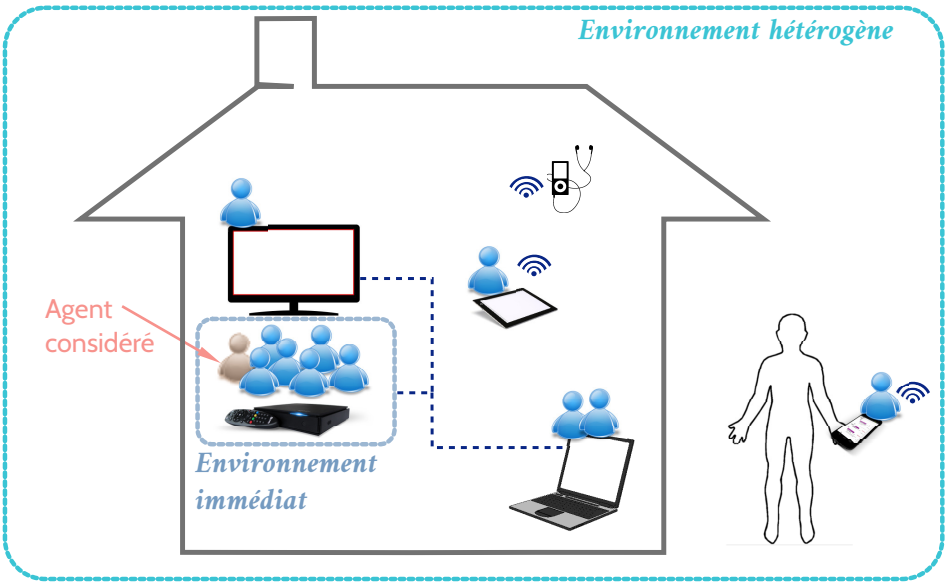


Figure 4.1 – *Environnement immédiat et environnement hétérogène d'un agent embarqué sur un SE de type Set-Top Box.*

4.2 | Caractéristiques d'un agent embarqué

A l'instar des SMA utilisés en simulation, nos systèmes multi-agents embarqués peuvent être composés de différentes familles d'agents. Ces familles partagent des bases de connaissances identiques au lancement et possèdent les mêmes compétences. Chaque entité agent possède une identification unique et évolue de façon autonome. Les spécificités liées aux systèmes embarqués impactent certaines caractéristiques décrites ci-après.

Autonomie

Les agents embarqués évoluent indépendamment du reste du SE. Ils peuvent agir sur ce dernier en fonction des informations qu'ils reçoivent de leurs environnements par le biais de leurs systèmes de perception. Ils mettent à jour leur base de connaissances de façon autonome et contrôlent les compétences qu'ils proposent aux autres entités partageant leur environnement hétérogène. Les agents interagissent librement avec les autres entités selon leurs besoins. Pour assurer la fiabilité du SE, nous essayons de garder des comportements les plus reproductibles possibles au niveau de leur impact sur le système.

Capacité d'interaction

La capacité d'interaction d'un agent embarqué passe par les systèmes de communication de sa plateforme. Elle peut être amenée à s'adapter à plusieurs systèmes de communication différents selon le contexte matériel.

Les agents embarqués peuvent réagir à des interruptions. Une interruption sera soit due à des indicateurs systèmes (capteurs, signaux, etc.), soit due à des interactions [Définition 13] avec d'autres agents.

Définition #13 : *Interaction*

Une **interaction** correspond à un échange d'informations dans le but d'acquérir de nouvelles connaissances ou de déclencher un service. Cet échange se fait par le biais de messages normalisés.

Une interaction initiale ne vise qu'au déclenchement d'une compétence d'un agent et n'attend pas de retour d'informations direct. Il est question de "requête". Si le traitement généré par la compétence ciblée produit un retour d'informations, un nouveau message est envoyé en tant que "réponse" [Définition 14]. Nous nous concentrons sur ces deux types de messages pour alléger l'empreinte mémoire de nos agents embarqués et ainsi favoriser l'optimisation du SE.

Définition #14 : *Requête et Réponse*

Une **requête** (*Ask*) est un message envoyé à un agent pour déclencher une de ses compétences. Il peut contenir des informations selon les besoins du traitement lié à cette compétence.

Si le traitement concerné nécessite un retour d'information ou plusieurs échanges d'informations, la suite de la communication se fera par le biais d'autres messages : des **réponses** (*Answer*).

Dans le cadre d'un système embarqué, la communication demande une synchronisation nécessitant du traitement en parallèle. En effet, dans la partie 2.1 nous avons détaillé un besoin de réactivité des SE. Il implique qu'un message doit pouvoir être traité assez rapidement pour ne pas bloquer l'agent initiateur de l'interaction.

De plus, les systèmes de communication embarqués permettent d'intégrer une confirmation de réception après l'envoi d'un message. Cette particularité est relative à la robustesse du système. En effet en intégrant des vérifications à plusieurs niveaux de programmation (ici au niveau du système de communication et au niveau de l'algorithme qui l'utilise), il est possible de détecter une éventuelle erreur avec plus de précision et de pouvoir la corriger plus rapidement. Le système d'interaction de l'agent est bloqué en attendant la réception de cette confirmation. Il est donc important que cette dernière se fasse à la réception du message et non à la fin du traitement du contenu du message. Par ailleurs, lorsqu'un agent effectue l'une de ses compétences, il doit toujours pouvoir recevoir des messages d'autres agents.

Pour répondre à ces exigences, chaque agent intègre un ou plusieurs thread "d'écoute". Ces threads, un pour chaque système de communication supporté par le SMA, sont dédiés à la réception des messages. Cette réception génère une brève interruption de type *callback*, c'est à dire un appel à une fonction générique d'une couche supérieure. Elle permet l'enregistrement du message dans la base de connaissances de l'agent et se charge de le diriger vers la compétence ciblée.

Compétences

Un agent possède des compétences spécifiques selon sa famille, correspondant à sa capacité à résoudre des tâches précises. L'agent embarqué propose ses compétences aux processus non-agents partageant son environnement en plus des autres agents du SMA. Nous nommons ces compétences des "services" [Définition 15].

Les traitements liés à ces services ne représentent pas toujours la même durée. La réactivité du système est évaluée notamment en fonction du temps d'attente de nos processus. Nous devons donc minimiser le blocage d'un agent en attente d'une réponse après l'activation d'un service court. Pour assurer une synchronisation la plus efficace possible, chaque service est réalisé par un thread différent. Ainsi le temps de réponse des services de courte durée n'est pas impacté par des services plus étendus.

Définition #15 : *Service*

Un **service** représente la capacité d'un agent à réaliser une tâche donnée. Un agent propose ses services au reste du SMA ainsi qu'aux processus partageant son environnement immédiat.

Un service est déclenché par une requête provenant d'un processus ou d'un autre agent à travers les interfaces de communication du SMA.

Pour assurer que les requêtes soient reçues par le service adéquat, chacun est relié à une file d'attente dédiée. Lors de l'enregistrement des messages dans la base de connaissances de nos agents embarqués, les informations sont sauvegardées en tant qu'élément dans des Files d'Attente (FA). Plus précisément, deux files sont considérées : une pour les requêtes, l'autre pour les réponses. Cette répartition permet un premier tri : les nouvelles requêtes parallèles à un service en cours peuvent s'empiler dans la file dédiée sans venir perturber les échanges d'informations dus au traitement. En effet, ces échanges sont enregistrés dans la FA de réponses. Le tri des messages a lieu dans la fonction *callback* de réception évoquée avec les capacités d'interaction de nos agents embarqués.

Dans la Figure 4.2, nous présentons l'exemple d'un agent proposant trois services différents. Chaque service est réalisé par un thread issu du processus agent et est rattaché à deux FA de réception pour les messages. Le premier service correspond à un traitement très court. Il ne nécessite pas d'échange d'informations et ne génère pas de réponse. Du fait de son temps de réponse rapide, les requêtes reçues par le service 1 sont traitées immédiatement. Le deuxième service nécessite un traitement plus long, entrecoupé d'interactions. Il envoie deux fois des requêtes et reste bloqué en attendant les réponses correspondantes. A la fin de son traitement, il ne génère pas de réponse. Les requêtes peuvent s'accumuler dans la file d'attente correspondante. Le dernier service a un temps de traitement moyen ne nécessitant pas d'interactions. Il est le seul à générer une réponse en fin de traitement.

Par défaut, chaque message est traité selon son ordre d'arrivée. Nos agents embarqués ont cependant la possibilité de trier les messages de la file des réponses, notamment s'ils attendent une information spécifique.

Par exemple, prenons un SMA composé d'un agent de registre et de plusieurs agents de température : un pour chaque pièce d'un bâtiment. L'agent de registre pourra trier les réponses reçues des autres agents selon les pièces auxquelles ils correspondent.

Nous n'avons actuellement pas rencontré ce besoin de tri pour les requêtes. Toutefois la manipulation est possible et pourrait servir si une perspective de priorité des requêtes venait à être mise en place.

La mise à disposition des compétences de l'agent au sein du SMA est directement liée à son comportement.

Comportement

Dans la sous-section 3.1.1, nous avons vu trois catégories d'agents du réactif, agissant selon des règles pré-établies en fonction des conditions perçues, au cognitif, capable de planifier ses actions et de mémoriser leurs effets sur l'environnement, en passant par l'hybride et son offre de compromis. Plutôt que définir nos agents selon ces types, nous préférons considérer une façon de décrire directement leur comportement pour plus de liberté. Nous avons détaillé divers modèles agents décrivant des comportements orientés selon des buts ou des objectifs d'applications. Ces choix impliquent bien souvent une ré-écriture complète de l'agent pour l'intégration de nouvelles compétences.

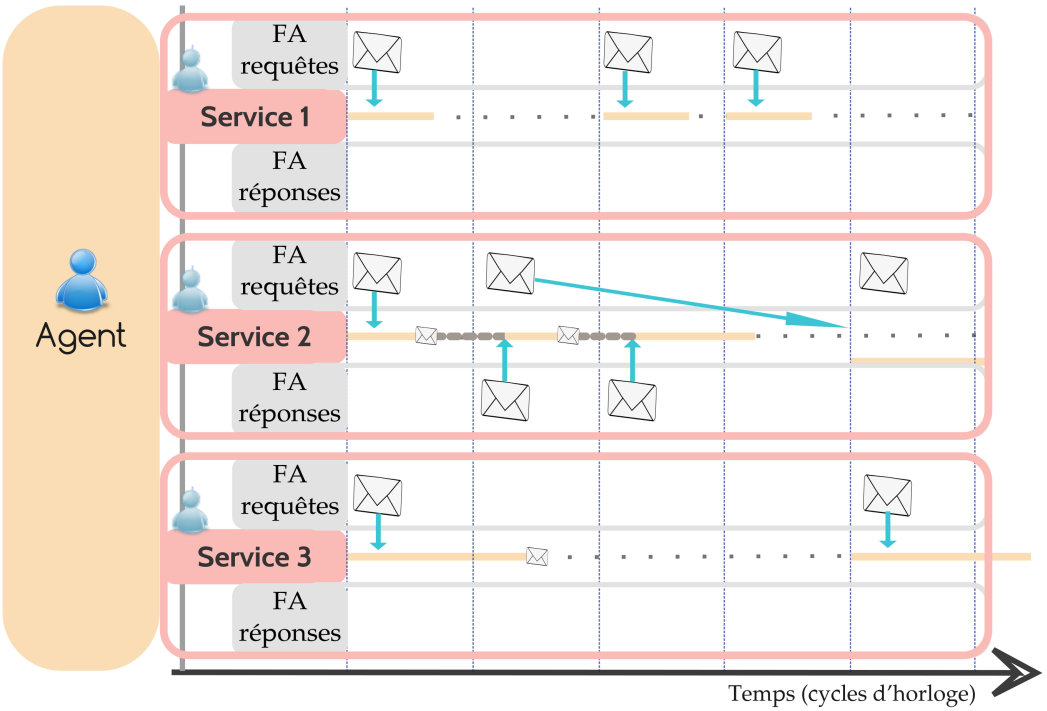


FIGURE 4.2 – *Diagramme de fonctionnement d'un agent embarqué proposant 3 services.*

Nous considérons notre SMA comme un ensemble ouvert auquel un agent embarqué peut être ajouté à tout instant sans gêner la cohérence de l'ensemble. Nous choisissons donc, à l'instar du travail réalisé pour le modèle IODA, de nous concentrer sur une façon de décrire des comportements génériques permettant des ajouts et retraits aisés de compétences.

Nos agents embarqués suivent un comportement périodique lié au cycle de vie de leurs services. Cette notion de cycle de vie est semblable au fonctionnement d'une machine d'état d'un système temps-réel. Les machines d'état proposent une représentation du fonctionnement d'un système sous forme de graphe représentant un nombre fini d'états et les conditions de passage d'un état à un autre.

La norme FIPA a ainsi détaillé le cycle de vie d'un agent [FIPA00023] tel que représenté en Figure 4.3. Nous y retrouvons 5 états décrivant le comportement d'un agent selon son évolution au sein d'un SMA. A partir d'un statut indiqué comme "Inconnu" par la spécification, un agent peut être créé. Il passe alors à l'état "Initialisé" à partir duquel une invocation pourra le rendre "Actif". Cet état est au centre du comportement d'un agent. Les autres états, "Bloqué" "Attente" et "Transport", représentent des statuts particuliers voués à retourner à l'état "Actif". Le statut "Transport" est spécifique aux agents mobiles, c'est à dire ayant la capacité de modifier leur localisation au cours de leur évolution. "Attente" et "Bloqué" représentent tout deux des états d'attente. Cependant, si le premier est initié par l'agent lui-même, le second est forcé par l'agent de gestion AMS.

Au cœur des systèmes embarqués, la réactivité est un besoin important. Nous devons donc assurer des services en parallèle pour augmenter cette réactivité au maximum. Par ailleurs, nos threads de service possèdent un comportement propre distinct du cycle de vie du processus agent dont ils sont issus. Nous présentons donc notre adaptation de ce cycle pour nos agents embarqués. Les états que nous proposons sont également légèrement distincts. Nous retrouvons la notion d'"Attente" souhaitée et la notion de "Bloqué" correspondant à une attente forcée. Nous retrouvons également le statut "Actif" représentant l'état principal pour l'évolution d'un agent ou d'un service. Enfin, nous séparons l'état "Initialisé" en deux états distincts : "Créé" et "Prêt". Cette séparation nous permet de distinguer la création d'un processus agent ou d'un service et sa reconnaissance au sein du SMA.

La Figure 4.4 représente les diagrammes d'état de nos agents et de leurs services. Nous détaillons en premier lieu le cycle de vie d'un processus agent.

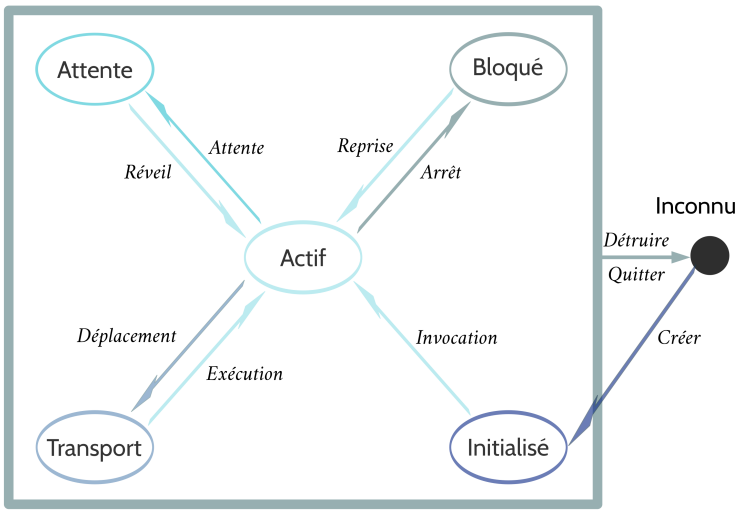


Figure 4.3 – *Cycle de vie d'un agent selon la norme FIPA.*

Le premier état présenté est "Créé". Il indique que le processus agent a obtenu l'allocation de ressources matérielles pour son fonctionnement. Il existe en tant qu'entité logicielle. Une fois initialisé, il passe à l'état "Prêt". Son initialisation implique l'obtention d'un identifiant, une connexion aux systèmes de communication reconnus par le SMA et la mise à niveau de sa base de connaissances. Il existe alors en tant qu'entité agent. Une activation le fait passer à un nouvel état au cours duquel il va générer ses threads de service. Ces derniers possèdent leur propre cycle de vie décrit plus bas. Une fois tous ses services ainsi lancés, le processus agent passe en "Attente". Les threads [Définition 8] ont accès à la même zone mémoire que leur processus parent. C'est donc eux qui mettront à jour la base de connaissances de l'agent et le feront ainsi évoluer au sein du SMA.
Si tous les threads se sont terminés, le processus agent retourne à l'état "Prêt". Il pourra de nouveau être activé, ou être lui-même terminé, auquel cas il repassera à l'état "Créé" afin que les ressources lui ayant été allouées soient libérées.

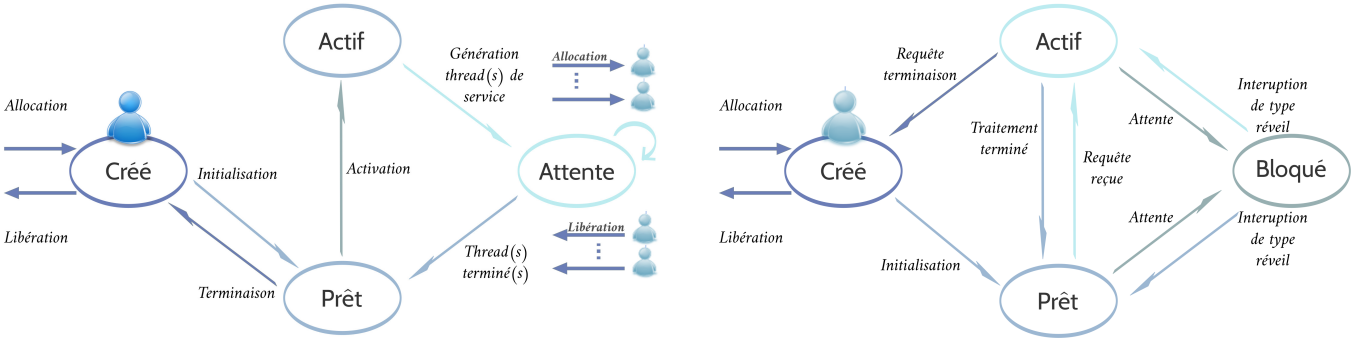


Figure 4.4 – *Cycles de vie d'un processus agent (à gauche) et de ses services (à droite) par leur diagramme d'état.*

Le cycle de vie d'un service dénote certaines différences. Son initialisation implique d'être raccordé aux deux files d'attente de message lui étant dédiées. Une fois "Prêt", il ne pourra être activé qu'à la réception d'une requête. Son état Prêt se résume donc à vérifier sa FA de requête. Si la file d'attente est vide et pour éviter un problème de monopolisation du processeur (attente active), le service passe en état "Bloqué" correspondant à une attente passive, dans un état de "sommeil". Au moment de la réception d'un message, la fonction *callback* envoie une notification au thread lors de l'enregistrement du message dans la file pour le "réveiller".

Si des requêtes sont en attente au moment où le service vérifie la file en état "Prêt", il passe en état "Actif" et entame le traitement. Lors d'un traitement, en état "Actif", le thread peut également passer en état "Bloqué" pour l'attente d'une réponse.

Un changement d'état permet de re-donner la main à l'ordonnanceur. Si l'enchaînement des états se fait en théorie, en pratique l'ordonnanceur garde le contrôle des ressources. Le retour à l'état "Prêt" permet donc d'éviter un blocage du système embarqué.

Lorsque le contenu d'une requête à traiter est "arrêt", le service se termine et repasse à l'état "Créé" pour être libéré.

Objectif

L'objectif propre à un agent embarqué est la satisfaction des requêtes reçues.

Chaque agent possède des caractéristiques personnelles et une base de connaissances de son environnement immédiat. Ces données lui permettent de déterminer sa capacité à remplir ses objectifs. Chaque famille d'agents implique des critères pouvant influer sur leur estimation de leur capacité à satisfaire une requête reçue. Un agent évolue en collaboration avec les autres agents du SMA. S'il s'estime incapable de remplir son objectif seul, il déclenche une demande d'aide auprès des autres entités agents du SMA selon les principes de négociation et de coopération vus en section 3.3.1.

La Figure 4.5 présente un diagramme résumant le fonctionnement d'un processus agent. Nous y retrouvons le processus principal à partir duquel sont lancés les threads d'écoute reliés aux systèmes de communication ainsi que les threads de service hérités de la famille d'agents.

4.3 | Méthodologie de formalisation d'un SMA embarqué

Dans cette partie, nous proposons de détailler notre formalisation de SMA embarqués. Certaines étapes sont inspirées du modèle IODA décrit en 3.1.2.2.
Pour réaliser un SMA, nous proposons de considérer une tâche suivant une programmation linéaire pour la transformer en programmation multi-agents. Nous pouvons également partir d'un SMA existant et y ajouter les familles d'agents ou les services nécessaires pour l'adapter à l'embarqué. La réflexion proposée est également adaptée pour réaliser un SMA à partir d'un cahier des charges.

Au cours de cette partie, nous utilisons une tâche théorique pour illustrer nos étapes. Cette tâche permet de traiter un signal numérique en lui appliquant une Transformée de Fourier Discrète (TFD). Ce calcul mathématique est fréquemment utilisé en traitement d'images. Il permet de ramener une matrice de pixels à une représentation fréquentielle. Cette pratique facilite l'application de filtres. Par exemple, il est possible de dé-bruiter une image en retirant les très hautes fréquences considérées comme parasites ou d'alléger l'image avant une compression. Notre tâche comportera également une interaction avec l'utilisateur et une prise en compte de la mémoire disponible sur le système. En Figure 4.6, nous présentons le diagramme de cette tâche.

La première étape de notre tâche exemple ouvre une fenêtre d'interaction avec l'utilisateur. Il lui est alors demandé de rentrer la taille mémoire du signal numérique qui sera traité. En étape deux, la Capacité Mémoire Disponible (CMD) est récupérée par notre programme. Ce dernier effectue alors une soustraction de la CMD par la taille du signal numérique à traiter. Il vérifie ainsi que la suite du traitement ne dépassera pas la capacité matérielle. L'étape trois consiste en un affichage du résultat de cette opération. Si la capacité mémoire est suffisante, le programme enchaîne avec l'étape quatre. La tâche effectue cette fois une addition pour mettre à jour la valeur de la capacité mémoire utilisée par le système. Il ajoute à l'ancienne valeur la taille du signal numérique à traiter. En cinquième étape, le programme applique la TFD au signal numérique. Enfin le résultat de la TFD est affiché à l'utilisateur en sixième étape.

Notre méthodologie se découpe en 3 phases, représentées en Figure 4.7 : la caractérisation des familles d'agents, l'établissement de la table des besoins et la réalisation d'une matrice des interactions.

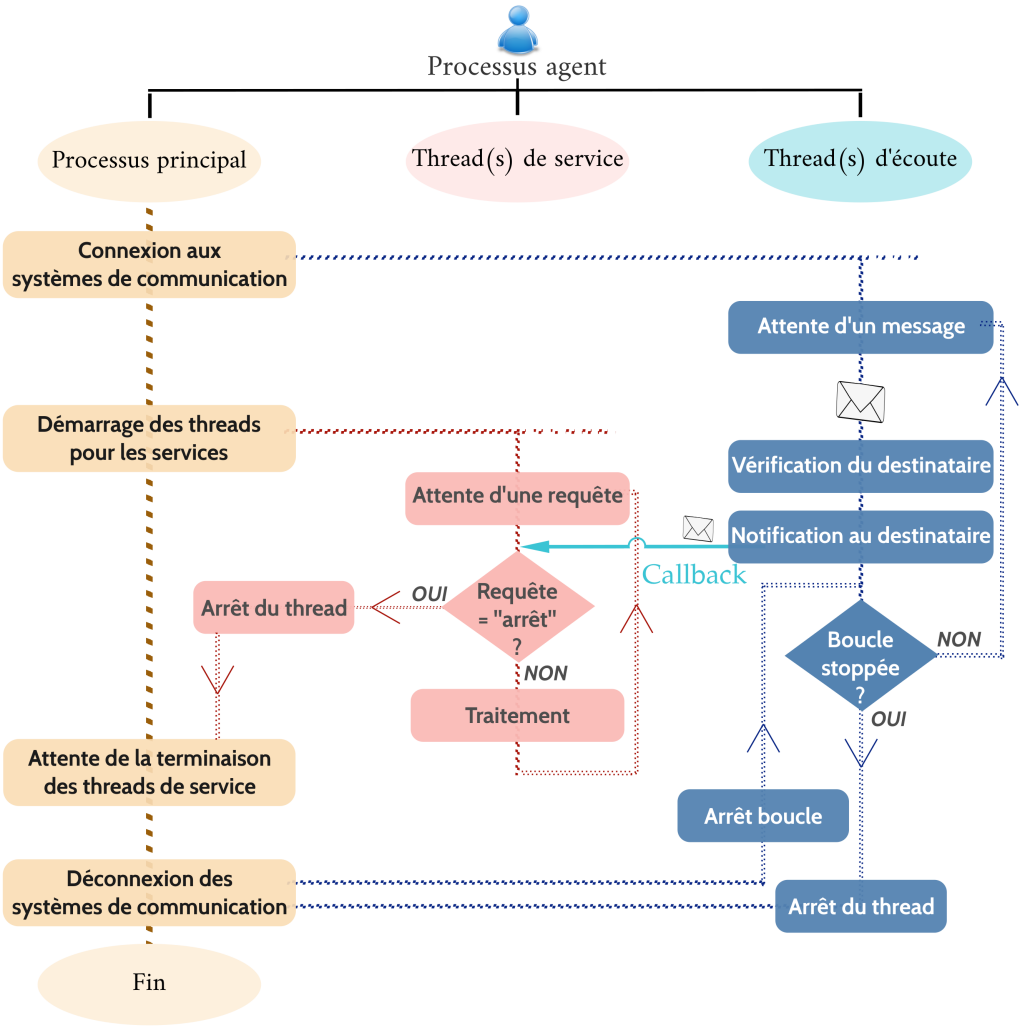


FIGURE 4.5 – *Diagramme de fonctionnement d'un agent embarqué.*

4.3.1 Caractérisation des familles d'agents

Tout d'abord, nous découpons la tâche considérée en sous-tâches selon la forme de service rendu. Ces sous-tâches représentent nos familles d'agents.
Pour établir cette répartition, nous considérons trois points : la base de connaissances requise, la parallélisation du traitement et la résolution des opérations par différentes méthodes.

Base de connaissances

Les familles d'agents doivent chacune posséder une base de connaissances précise et spécifique. Si, après avoir effectué une première répartition, nous constatons que l'une d'entre elles réalise des services nécessitant des connaissances très différentes, il est préférable de la séparer en deux. Selon le même principe, si deux familles regroupent des sous-tâches requérant la même base de connaissances, elles peuvent probablement être regroupées.

Par exemple si nous considérons une action de calcul mathématique et une action d'affichage vidéo. Ces deux actions nécessitent des bases de connaissances différentes : l'une liée à des formules mathématiques précises, l'autre liée aux bibliothèques de fonctions utilisées pour l'affichage et le contrôle de la vidéo (gestion d'interactions de type pause, lecture, retour, etc.). Ces deux actions seront donc gérées par des familles d'agents a priori différentes. Pour le cas inverse, nous pouvons considérer par exemple une famille chargée d'incrémenter un compteur et une autre chargée de décrémenter un autre compteur. Les actions d'incrémentation et décrémentation relèvent toutes deux

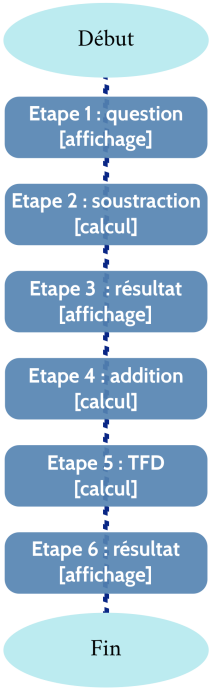


FIGURE 4.6 – *Diagramme de la tâche exemple pour illustrer les étapes de réalisation de nos SMA embarqués.*

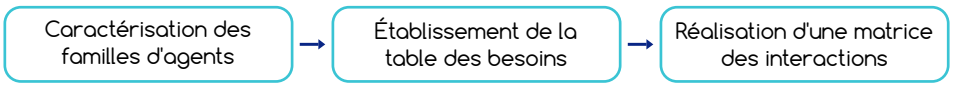


FIGURE 4.7 – *Récapitulatif des 3 étapes de formalisation d'un SMA embarqué.*

d'opérations mathématiques élémentaires. Ces deux familles partagent donc la même base de connaissances et pourraient être réunies.

Il est cependant important de considérer le contexte d'application de notre SMA ; c'est pourquoi la base de connaissances seule ne peut être un critère suffisant.

Parallélisation du traitement

Avec une même base de connaissances, les diverses opérations possibles n'impliquent pas toutes le même temps de réponse. Il est donc intéressant de séparer une opération longue en lui dédiant une famille d'agents. Nous diminuons alors le temps de réponse global du système en traitant les opérations courtes en parallèle.

Par exemple, dans le cadre d'opérations visant à décoder des images, le temps de réponse pour décoder une image en ultra haute définition (3 840 x 2 160 pixels) est plus important que celui pour décoder une image basse définition (480 x 360 pixels). De ce fait, répartir les deux formes de décodage entre deux familles d'agents différentes permettrait de traiter plusieurs images en basse définition en parallèle du traitement d'une image d'ultra haute définition, au lieu d'attendre le traitement de l'image principale avant de commencer le traitement des plus petites.

Par ailleurs, si nous considérons le cadre des communications, il peut être primordial d'être disponible le plus souvent possible pour réceptionner un message. Si nous considérons une famille d'agents chargée de la communication, il peut être alors pertinent de la séparer en deux familles distinctes : une chargée de suivre une discussion et l'autre dédiée à la réception pour assurer une écoute continue du réseau.

En plus des avantages relatifs à la parallélisation du traitement, nous considérons les possibilités d'application.

Différenciation des méthodes

Nous considérons l'éventualité qu'une opération soit résolue de plusieurs façons différentes. Il est alors intéressant de créer plusieurs familles d'agents correspondant à ces méthode de résolution.

Par exemple si nous traitons un signal audio et souhaitons réduire le bruit sur ce signal, plusieurs formules mathématiques différentes pourront nous permettre d'effectuer cette action, chacune possédant ses avantages et inconvénients. Il est alors intéressant de dédier une famille d'agents à chaque formule pour permettre d'effectuer un choix de la méthode à appliquer en prenant en compte plus de paramètres tels que le contexte ou une demande spécifique liée à une formule en particulier.

Dans le cadre d'un contexte embarqué, ces possibilités peuvent être associées aux différentes unités matérielles capables de réaliser les mêmes services. Nous pouvons alors considérer établir une famille d'agents pour chaque ressource afin de permettre l'utilisation de ces composants matériels en fonction des paramètres précédemment évoqués.

Ce critère doit donc être considéré en compromis du premier critère, la base de connaissances. La différenciation des méthodes vise à augmenter la modularité et la flexibilité du SMA, la considération de la base de connaissances vise sa rentabilité et son efficacité en termes d'empreinte mémoire.

Répartition finale

Une fois les trois caractéristiques de décision évaluées, nous choisissons finalement les familles d'agents nous paraissant les plus pertinentes. Nous clarifions leurs sous-tâches en établissant une liste des services qu'elles proposent aux entités de leur environnement immédiat ou hétérogène.

Dans le cadre de la tâche illustrant notre protocole, nous choisissons de regrouper les étapes liées à l'affichage et les étapes liées aux calculs mathématiques élémentaires (soustraction et addition). Nous assignons ainsi trois familles d'agents distinctes : une première capable de gérer l'affichage, une deuxième pour la gestion de la mémoire usant de calculs élémentaires et une troisième pour effectuer la transformée de Fourier [Figure 4.8].

Nous choisissons de nommer nos familles selon les services qu'elles représentent pour faciliter notre compréhension des prochaines étapes. Nous appelons ainsi respectivement la famille "Affichage" (1), la famille "Mémoire" (2) et la famille "TFD" (3). Nous vérifions la cohérence de notre première répartition en étudiant les bases de connaissances nécessaires pour chacune. Nous pouvons constater que les connaissances requises par la famille Mémoire pour les calculs élémentaires sont également possédées par la famille TFD. Cependant, le temps de réponse d'une addition ou d'une soustraction est moindre en comparaison d'une transformée de Fourier. Il est donc tout de même intéressant de distinguer ces deux familles de tâches pour paralléliser leur traitement.

4.3.2 Établissement de la table des besoins

Une fois nos étapes séparées, certaines informations vont manquer. Par exemple dans le cas de notre tâche réalisant une transformée de Fourier, un agent Affichage ne pourra pas indiquer à l'utilisateur le résultat obtenu lors de l'étape deux. Comme les processus sont séparés dans le cadre d'une programmation multi-agents, ils ne partagent plus ces informations. Nos agents doivent donc interagir pour échanger des renseignements. Nous identifions ce manque par un "besoin".

La notion de besoin concerne également les informations détenues par d'autres entités de l'environnement de l'agent. Par exemple nos agents Mémoire doivent interagir avec les ressources de l'environnement immédiat pour connaître l'espace mémoire disponible sur le système embarqué.

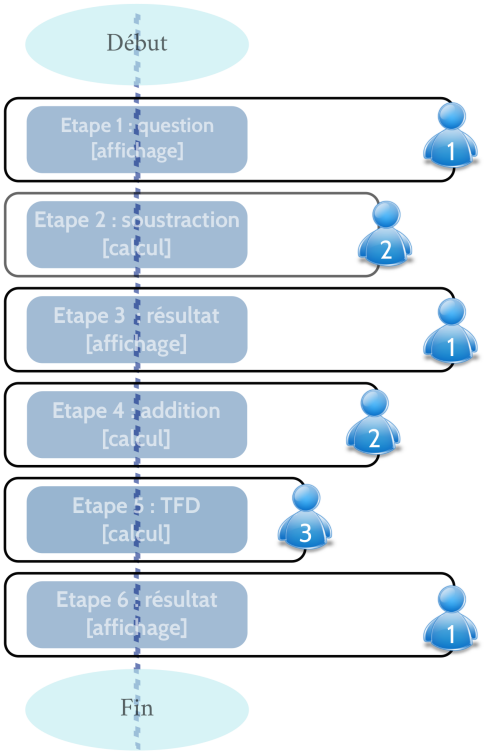


FIGURE 4.8 – *Illustration de la première étape de réalisation d'un SMA embarqué : détermination des familles d'agents.*

Nous considérons la définition suivante :

Définition #16 : *Besoins d'un agent et entités moyen*

Les **besoins** d'un agent correspondent aux informations ou données lui faisant défaut pour mener à bien les services qu'il propose.
Un besoin peut être comblé par une entité "moyen".
Les **entités "moyen"** peuvent fournir les informations manquantes sources du besoin d'un agent.
Une entité moyen peut être un autre agent, un processus ou encore une ressource matérielle.

Nous établissons une table répertoriant les besoins de nos familles d'agents. Cette table précise également quelles entités moyen peuvent combler ces besoins. Cette étape permet de préciser quelles seront les interactions nécessaires au fonctionnement de notre système. Les agents exempts de besoins sont qualifiés d'agents indépendants en opposition aux agents dépendant de moyens extérieurs pour achever leurs services.

Nous représentons cette étape pour le cas illustrant notre protocole de formalisation en Figure 4.9. Nous y constatons que les agents TFD sont indépendants. Ils n'ont en effet besoin que de leur base de connaissances propre pour achever leur service de calcul. Les agents Affichage ont besoin d'obtenir les données à afficher d'un moyen externe (ici les autres agents). Quant à nos agents Mémoire, ils doivent interagir plusieurs fois avec leur contexte matériel.

Pour la suite nous nous intéressons particulièrement aux interactions entre agents.

Familles d'agents	Besoins	Moyens
Affichage	Information pour le 2ème affichage	Mémoire
	Information pour le 3ème affichage	TFD
Mémoire	Informations pour le premier calcul [Informations à mettre à jour]	Contexte matériel : Ressource mémoire
TFD	*Agent indépendant*	

FIGURE 4.9 – *Table des besoins.*

4.3.3 Réalisation d'une matrice des interactions

Une fois notre table des besoins déterminée, nous analysons quelles interactions doivent avoir lieu entre nos agents. Comme évoqué dans la section 4, les agents embarqués peuvent utiliser deux types de messages pour la communication : les requêtes et les réponses. Dans ce cadre, les agents possédant des besoins soumettront des requêtes aux agents possédant les moyens nécessaires. Ces derniers renverront par la suite des réponses avec les informations manquantes.

Nous résumons ces interactions grâce à une matrice. Inspirée de celle proposée par le modèle IODA, elle expose des types d'agents Source et Cible. Une interaction est toujours initiée par une Source à destination d'une ou plusieurs Cible(s). Un agent appartenant à une famille peut contacter les autres instances d'agents de celle-ci.

Cette matrice nous permet également de lister les compétences de nos agents : les services. Nous utilisons le symbole ∅ pour les représenter dans la première colonne de notre matrice. Nous indiquons ainsi qu'ils n'ont pas de cible mais représentent les fonctions internes d'une famille d'agents. Les interactions de type requête ou réponse sont toujours liées à un service. Une requête permet de le déclencher et, si elle avait pour but de combler un besoin, le service se conclura par l'envoi d'une réponse.

Source \ Cible	∅	Agent Affichage	Agent Mémoire	Agent TFD
Agent Affichage (1)	+ Affichage utilisateur		+ Ask Gestion mémoire	+ Ask Calcul TFD
Agent Mémoire (1)	+ Gestion mémoire	+ Answer Gestion mémoire		
Agent TFD (1)	+ Calcul TFD	+ Answer Calcul TFD		

Liste des services des agents ⟷ *Application*

FIGURE 4.10 – *Matrice des interactions pour la partie applicative du SMA embarqué.*

Lors de cette étape, nous déterminons également les besoins en termes de nombre d'implémentations des agents pour chaque famille spécifiée. A moins que la tâche complexe ne comprenne explicitement un traitement parallèle, une seule instance d'agent dans chaque famille est suffisante. La possibilité d'intégrer plusieurs instances au sein d'une même famille d'agents sera traitée plus en détail dans la partie III de notre mémoire, lorsque nous aborderons des expérimentations concrètes pour les apports des SMA dans les SE.

La Figure 4.10 présente la construction d'une matrice pour le SMA illustrant notre protocole de formalisation. Nous proposons une seule instance d'agent pour chaque famille. Nous retrouvons les interactions nécessaires pour le partage d'informations. La première étape de notre tâche était l'affichage pour l'utilisateur. Par conséquent le service "Affichage utilisateur" représente le point de départ de notre SMA. Il peut être déclenché par une interaction de type requête initiée par une entité non-agent.

Une fois cette matrice réalisée, nous disposons de toutes les informations nécessaires pour la conception de notre SMA embarqué. Il nous reste à proposer une plateforme pour fournir un contexte d'exécution et d'évolution au sein des systèmes embarqués.

Chapitre 5

MERMAID : une plateforme agent embarquée

"Dans notre langue, l'expression la plus dangereuse est "nous l'avons toujours fait de cette façon". C'est pourquoi j'ai une horloge sur mon mur qui va dans le sens inverse des aiguilles d'une montre."

Grace Hopper
Mathématicienne, Informaticienne (1906 - 1992)

Les plateformes multi-agents permettent de fournir le contexte intégrant des systèmes de communication, une structure organisationnelle avec gestion administrative du SMA, des fonctions génériques et des ontologies (meta-données, identifiant d'un agent, etc.). Dans la partie 3.1.3 nous avons comparé plusieurs plateformes multi-agents existantes selon les critères pertinents pour notre cadre de recherche :

- **le langage de programmation**. Notre démarche est de pouvoir intégrer le SMA au plus bas niveau matériel possible. Ce souhait nécessite un langage de programmation adapté, tel que le C. Nous mettons ainsi de côté les langages objets courants dans le domaine multi-agents, comme par exemple Java.
- **le modèle agent supporté**. Notre méthodologie de conception s'appuie sur des modèles orientés interactions et s'inspire notamment du modèle IODA.
- **le domaine d'application**. Notre plateforme vise une intégration sur des systèmes embarqués hétérogènes.
- **la normalisation de la plateforme**. En section 3.2, nous avons souligné l'importance de standardiser nos communications. Nous suivons la norme FIPA-ACL.

Les plateformes multi-agents existantes étudiées se focalisent sur d'autres critères d'expérimentation. Nous avons donc décidé de proposer notre propre plateforme agent embarqué : MERMAID.

Pour la réalisation de cette plateforme, en plus des critères de sélection établis, nous prenons garde aux besoins du domaine des SE décrit dans la section 2.1 (fiabilité, réactivité et optimisation). Nous prenons notamment particulièrement garde à :

- **l'architecture**. Nous souhaitons à terme pouvoir faire fonctionner notre système multi-agents dans un contexte matériel hétérogène. Selon les composants matériels des SE considérés, il peut être nécessaire d'établir des fonctions de communication différentes ou encore d'adapter le langage de programmation afin d'assurer l'optimisation de l'ensemble. Il est par conséquent primordial d'utiliser une plateforme possédant une architecture modulaire avec des bibliothèques de fonctions indépendantes les unes des autres. Par ailleurs, cette séparation permet également une maintenance facilitée et augmente ainsi la fiabilité de notre système.

- **les moyens de communication proposés**. Nous considérons le fait que notre solution multi-agents sera appliquée à un SE sans modifier les composants de ce dernier. Cette optique implique de rendre notre plateforme la plus légère possible en termes d'empreinte mémoire afin de limiter son impact en consommation de ressources. Cette démarche s'inscrit également dans l'optimisation de notre ensemble. Nous utilisons donc des moyens de communication existant dans l'embarqué potentiellement déjà intégrés dans le SE considéré. De plus, en évitant d'ajouter une couche logicielle de gestion des communications nous augmentons la réactivité de notre plateforme.

En chapitre 1 nous avions formulé une hypothèse renforçant cet apport de moyens de communications adaptés au domaine de l'embarqué :

Rappel hypothèse #2

Le suivi des standards d'administration et d'architecture multi-agents nous permettra de proposer de nouveaux modèles de communication pour une plateforme embarquée.

Dans ce chapitre, nous présentons MERMAID à travers son architecture, son administration et l'organisation des différents SE l'intégrant.

5.1 | Architecture

Une architecture modulaire est primordiale dans le domaine des systèmes embarqués. Elle permet d'augmenter la fiabilité du système en regroupant les fonctions selon leur utilité et leur généricité. En effet, il est alors possible de valider le fonctionnement des librairies au fur et à mesure. Par ailleurs, les bibliothèques de fonctions pourront être ré-utilisées pour d'autres projets. Cette forme d'architecture facilite également l'ajout de systèmes de redondance. De plus, nous souhaitons pouvoir étendre notre système multi-agents à plusieurs SE, homogènes ou hétérogènes. Dans ce cadre, une architecture en plusieurs parties permet d'adapter les couches impactées par le changement matériel sans forcément devoir modifier l'ensemble de la plateforme. Nous avons par conséquent adopté ce type d'architecture pour assurer l'indépendance des différents outils apportés par la plateforme.

L'architecture de la plateforme MERMAID est structurée en deux parties : une couche de communication et une couche d'administration et de gestion. Cette dernière regroupe d'une part une interface de programmation avec toutes les fonctions relatives à l'évolution des agents et d'autre part les agents administratifs de la norme FIPA. En effet, a contrario d'un agent purement applicatif, ces trois agents ne varient pas entre deux SMA. Nous considérons également une couche applicative représentant les agents basés sur MERMAID. La Figure 5.1 présente l'ensemble de notre architecture. La partie administrative de MERMAID avec les agents AMS, ACC et DF issus de FIPA est considérée comme un bloc à part. Les autres blocs sont regroupés sous le terme de noyau de MERMAID. A partir de ce noyau, notre plateforme offre d'ores et déjà un contexte de déploiement avec systèmes de communication et structure organisationnelle. Il doit donc être inclus dans chaque SE intégrant nos agents pour assurer l'interopérabilité de l'ensemble du SMA. Le bloc d'administration, quant à lui, doit être inclus dans une seule plateforme MERMAID dans un contexte multi-cartes.

5.1.1 Couche de communication

Cette couche représente le plus bas niveau de notre architecture. Elle comporte l'ensemble des fonctions d'utilisation des systèmes de communication. Ces derniers permettent les interactions entre deux processus agents ou entre un processus agent et une autre entité partageant son environnement (immédiat ou hétérogène).

Considérant ces deux environnements, nous avons déterminé que notre plateforme devait supporter au moins deux protocoles de communication distincts : l'un adapté à des interactions échangées dans l'environnement immédiat, l'autre dédié à des interactions dans un environnement hétérogène. Parmi les différents moyens d'échanges inter-processus décrits dans la partie 2.4, nous avons retenu deux solutions pour MERMAID : D-Bus et les IPC sockets.

Dans l'environnement hétérogène, pour les interactions externes, nous utilisons le mécanisme IPC **socket**. Consommant peu de ressources et offrant un service très rapide, les sockets supportent une forme de communication asynchrone avec leur mode TCP. L'utilisation de ce mode nous permet d'assurer l'intégrité des paquets d'informations. Le mécanisme des sockets est adapté à un environnement multi-plateforme hétérogène grâce à son support par la majorité des systèmes. Pour des interactions entre différents SE, nous les relions donc sur un même réseau. Nous nous basons sur un adressage de type IP et numéro de port.

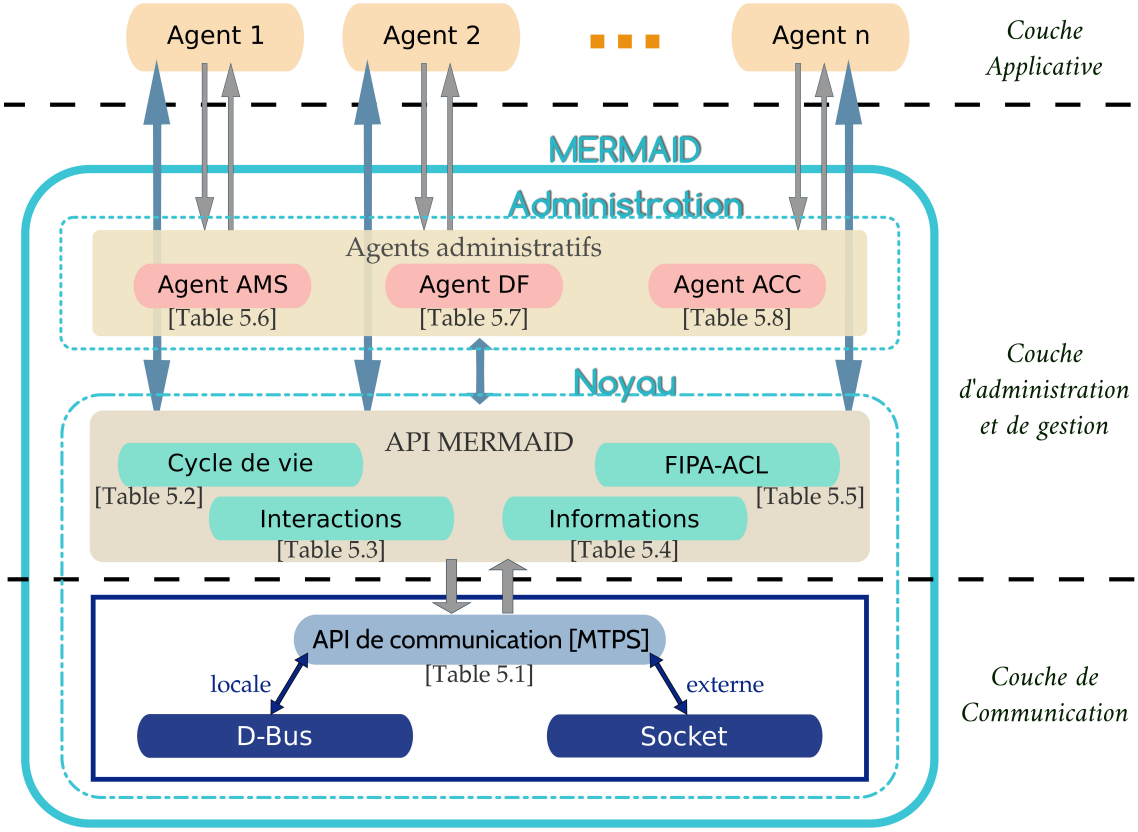


FIGURE 5.1 – *Architecture de MERMAID intégrant un noyau et une partie administrative.*

Dans l'environnement immédiat, nous utilisons le protocole de communication interne **D-Bus**. Le fonctionnement du protocole D-Bus est basé sur des sockets. Il s'agit d'un protocole intégré par défaut dans de nombreux systèmes embarqués incorporant un système d'exploitation de type UNIX et ses dérivés comme Linux. Il permet l'inter-communication des processus sous forme de messages à la fois dans les couches logicielles et matérielles du système. L'utilisation de ce protocole de communication embarqué apporte aux processus agents l'avantage de pouvoir communiquer avec les autres processus de la carte par le biais d'un bus dédié au SMA. L'interaction avec l'environnement matériel est donc directement possible.
Par ailleurs, ce protocole permet de communiquer soit par l'envoi d'un message vers un destinataire ciblé, soit via l'émission d'une information à plusieurs destinataires simultanément. Léger quant à la consommation des ressources matérielles, il supporte des communications asynchrones. Pour utiliser ce protocole, les processus s'enregistrent sur le bus dédié à leur application en indiquant les services qu'ils proposent aux autres processus utilisant le même bus. C'est ensuite le protocole D-Bus qui se charge du routage des messages. Grâce à cet enregistrement sous forme de blocs "service", D-Bus permet de travailler de façon modulaire et est donc adapté à des applications découpées en briques communicantes. Si une brique rencontre un dysfonctionnement, elle n'impacte pas les autres et peut se reconnecter au bus sans affecter l'ensemble du système. Le fonctionnement de ce protocole est donc en adéquation avec la formalisation de nos agents embarqués.

Nous possédons une bibliothèque de fonctions pour chacun de nos deux systèmes de communication. Souhaitant garder l'accès à nos fonctions de communication génériques quel que soit le système de communication supporté, nous établissons une autre bibliothèque de fonctions pour servir d'interface de programmation applicative (*Application Programming Interface*, API) [Table 5.1]. Cette dernière est appelée par la couche d'administration et de gestion et se charge de rediriger ces appels vers le système de communication le plus adapté. Ainsi, le mode de routage du message devient bien parfaitement indépendant du fonctionnement même de nos agents embarqués. Les fonctions correspondent à plusieurs états du cycle de vie d'un agent détaillé en section 4.2. Nous détaillons leur rapport à ce cycle dans leur description. Cette API correspond à la couche d'abstraction du transport des messages MTPS spécifiée par la norme FIPA.

Fonction	Description
context_ creator	Cette fonction crée une variable "contexte de communication" pour l'agent. Ce contexte regroupe les informations requises par le système de communication. Pour les agents, il s'agit d'une structure opaque dont ils ne connaissent pas le contenu. Ce contexte est constamment conservé et passé en paramètre pour utiliser le système de communication.
register_ on_ com_ system	Une fois le contexte de communication obtenu, l'agent s'enregistre auprès du système de communication. Il obtient ainsi une "adresse" à laquelle il devient joignable via ce système. Cette fonction est appelée pour chaque système de communication supporté par MERMAID. Elle complète les informations du contexte de communication de l'agent. Cette fonction et la précédente sont utilisées pour initialiser l'agent lors du passage de l'état "Créé" à l'état "Prêt".
quit_ com_ system	A l'inverse, cette fonction permet de se déconnecter des systèmes de communication. Les adresses enregistrées dans le contexte sont alors libérées. Cette fonction est appelée lors de la terminaison d'un agent pour retourner de l'état "Prêt" à l'état "Créé".
wait	Cette fonction permet de créer un thread d'écoute pour chaque système de communication. Lors de la réception d'un message, le thread active la fonction *callback* pour enregistrer le message dans la file d'attente de l'agent. Il revient ensuite se placer en écoute du réseau. Cette fonction est déclenchée lorsque l'agent est en état "Actif".
stop	Appliquée lors de la terminaison d'un agent, cette fonction permet de stopper la boucle d'écoute du réseau et de récupérer le thread correspondant.
send_ message & *send_ data*	Utilisée pour envoyer un message à travers l'un des systèmes de communication, cette fonction est activée par les services d'un agent durant leur état "Actif". Le thread passe ensuite en état Bloqué si le message demande une réponse. La décision du système de communication dépend des données du message et de sa destination. Le choix est déterminé au sein de cette fonction. Un envoi de données correspond à une deuxième fonction impliquant automatiquement un transport via sockets. Elle reprend le déroulement de la fonction précédente tout en envoyant des trames supplémentaires liées à la préparation de la réception d'un flux conséquent (indication du type de structuration des données et de leur taille). Cette fonction est employée si la structure n'est pas lisible dans un format de chaîne de caractères (exemple : tableau à double entrée).

Table 5.1 – *Fonctions de l'API de communication de MERMAID.*

5.1.2 Couche d'administration et de gestion

5.1.2.1 Description des librairies agents

La couche d'administration et de gestion de notre SMA regroupe les bibliothèques de fonctions des agents, c'est à dire les fonctions communes à tout agent évoluant sur MERMAID. Nous y retrouvons les fonctions génériques liées au cycle de vie d'un agent, à ses interactions et les fonctions concernant la récupération d'informations dans un message.

Les fonctions relatives au cycle de vie de l'agent [Table 5.2] concernent son intégration au sein de la plateforme et du système embarqué [Section 4.2]. Elles servent à l'allocation de l'espace mémoire nécessaire à son fonctionnement, son enregistrement auprès des différents systèmes de communication et enfin l'activation de ses threads de service et d'écoute. De façon symétrique, ces fonctions permettent également d'effectuer la suppression de l'agent après son arrêt en libérant la mémoire assignée et en désenregistrant l'agent des systèmes de communication supportés par MERMAID.

Fonction	Description
init_ agent	Une fois le contexte de communication obtenu, l'agent s'enregistre auprès du système. Cette fonction alloue l'espace mémoire nécessaire au fonctionnement de l'agent. Cette allocation concerne les services supportés par l'agent et les files de message associées. La fonction s'occupe également d'enregistrer l'agent auprès des services de communication de MERMAID en utilisant la fonction "*register_ on_ com_ system*" [Table 5.1] de l'API de communication. L'agent est ensuite en état "Prêt".
terminate_ agent	Parfait antagoniste de la fonction précédente, celle-ci libère la mémoire allouée à la fois au niveau du système embarqué (services et files de messages) et au niveau de la plateforme MERMAID (contexte de communication) en faisant notamment appel à la fonction "*quit_ com_ system*" [Table 5.1]. L'agent revient ensuite à son état initial : "Créé".
activate_ agent	Cette fonction est appelée lorsque l'agent est en état "Actif". Elle génère tous les threads de service de l'agent et fait appel à la fonction "*wait*" [Table 5.1] pour démarrer les threads d'écoute. A partir de là, l'agent embarqué passe en état "d'Attente" et poursuit son évolution au sein du SMA via ses threads.
deactivate_ agent	Cette fonction est appelée au début de l'état "d'Attente" de l'agent. Elle attend la terminaison de tous les services de l'agent pour repasser ce dernier en état "Prêt". Juste avant ce changement d'état, elle envoie une interruption à la couche inférieure pour arrêter les threads d'écoute.

TABLE 5.2 – *Fonctions correspondant au cycle de vie d'un agent embarqué.*

Pour interagir, les agents s'échangent des informations à travers des messages de type requête ou réponse [Définition 14]. Les fonctions liées aux interactions réalisent la gestion de ces messages et des files d'attentes associées [Table 5.3]. Ces fonctions sont employées par les threads d'un processus agent selon le cycle de vie de ces derniers.

Enfin, les fonctions relatives aux informations contenues dans les messages sont regroupées dans une troisième bibliothèque [Table 5.4]. Elles permettent la transmission, l'encapsulation ou la désencapsulation d'un certain nombre de variables. Elles comportent également une vérification du nombre de variables attendues, permettant ainsi un renforcement de la fiabilité du SMA. Toutes les fonctions de cette bibliothèque sont employées par les services en état "Actif" au préalable ou suivant une interaction.

Fonction	Description
get_related_queue	Cette fonction permet de relier une file d'attente de messages à un service proposé par l'agent. Elle est utilisée pour l'initialisation du thread lors de son passage de l'état "Créé" à l'état "Prêt".
check_queue	L'inspection d'une file d'attente liée au service de l'agent est réalisée par cette fonction. Elle détermine si la FA comporte au moins un message enregistré. Si c'est le cas, le service récupère le message et passe le thread en état "Actif". Sinon, il passe en état "Bloqué" pour attendre passivement une interruption de type réveil suite à la réception d'un message.
message_callback	Cette fonction est appelée non pas directement par les agents mais par le biais des threads d'écoute. Elle permet l'enregistrement d'un message reçu dans la file d'attente du service indiqué par les informations contenues dans l'encapsulation. Elle se charge également d'envoyer une interruption au thread pour l'activer s'il est en état "prêt".
release_queue_element	Cette fonction supprime un élément message d'une file d'attente. Cette radiation s'effectue en libérant la mémoire allouée pour cet élément et est effectuée une fois l'élément traité par le service "Actif" concerné.
perform_sending & *perform_sending_data*	Ces deux fonctions préparent l'encapsulation du message à transférer puis l'envoient grâce aux fonctions de l'API de communication "*send_message*" ou "*send_data*" (toujours selon le type de structuration des données) [Table 5.1]. Elles sont employées par un thread en état "Actif". Si le message requiert une réponse, le thread passe en état "Bloqué" pour l'attendre passivement.

Table 5.3 – *Fonctions d'interaction de MERMAID.*

Fonction	Description
get_mess_content	Cette fonction permet de récupérer les variables contenues dans le message sous forme de chaînes de caractères distinctes. La fonction gère la mémoire nécessaire à cette opération.
set_mess_content	A l'inverse, cette fonction est utilisée pour encapsuler un certain nombre de variables en une seule chaîne de caractères. La taille mémoire de la chaîne de caractères finale est pré-allouée en fonctions des variables à intégrer.
receive_data	Dans le cadre d'un transfert de données, les informations concernant les données transmises sont envoyées au préalable dans une chaîne de caractères. Cette fonction permet de récupérer ces informations spécifiques. Encore une fois, la gestion de la mémoire est assurée par la fonction.

Table 5.4 – *Fonctions de MERMAID utilisées pour manipuler les informations contenues dans un message.*

5.1.2.2 Normalisation des échanges

Dans la partie 3.2 nous avons détaillé le standard FIPA-ACL de normalisation des échanges dans un système multi-agents. Nous avons également développé l'ensemble de la norme FIPA pour ses apports concernant l'administration au sein d'une plateforme. Notre cadre de recherche impose des communications entre divers systèmes embarqués hétérogènes avec la mise en place de l'intéropérabilité des plateformes. Nous pourrons être amenés à communiquer avec des SMA tournant sur d'autres plateformes agents. Il est donc essentiel de normaliser notre administration et nos messages afin de rendre compatible notre plateforme avec un maximum d'autres plateformes multi-agents.

La norme FIPA-ACL propose une encapsulation des messages adaptée pour des interactions entre un agent et les diverses entités présentes dans ses deux environnements : immédiat et hétérogène. Les entités ainsi reconnues sont les processus non-agent, les utilisateurs et le contexte matériel. FIPA-ACL utilise également une identification spécifique, l'AID, pour chaque agent embarqué. L'AID représente l'ensemble des informations nécessaires pour contacter un agent sur MERMAID par le biais des systèmes de communication supportés. Nous ajoutons ainsi à notre couche d'administration et de gestion une bibliothèque de fonctions relatives à cette norme [Table 5.5].

Fonction	Description
administrative_registration	Appelée lors de l'initialisation de nos agents embarqués, cette fonction permet d'envoyer un message aux agents administratifs AMS et DF pour procéder à l'inscription de l'AID de l'agent ainsi que des services qu'il propose.
set_AID_encapsulation	Cette fonction intègre en une seule chaîne de caractères les informations correspondant à l'identification de l'agent. Nous regroupons ainsi son nom et son adresse IP ainsi que le numéro de port qui lui a été attribué.
get_AID_encapsulation	Le procédé inverse de la précédente fonction consiste à récupérer les informations relatives à l'AID d'un agent à partir d'une unique chaîne de caractères. C'est le travail effectué par cette fonction.
set_mess_encapsulation	Cette fonction complète l'encapsulation d'un message à partir des informations passées en paramètres.
get_mess_encapsulation	Cette fonction permet de récupérer toutes les informations du message reçu. Elle est utilisée notamment lors de la réception d'un message pour déterminer le service destinataire.
get_performative	Cette dernière fonction est spécifique au suivi d'une suite de messages. De nombreuses performatives sont proposées pour guider les interactions entre les agents par des suites de messages. Cette fonction permet de vérifier la performative employée dans le message reçu pour guider la suite des échanges.

Table 5.5 – *Fonctions administratives de MERMAID.*

L'ensemble des spécifications vues jusqu'ici forment le noyau de notre plateforme MERMAID. Pour parfaire la gestion de notre SMA, il nous faut ajouter un moyen de gérer administrativement nos agents. Pour un SMA étendu à plusieurs SE, seul l'un de ces derniers doit comporter cette partie administrative, les autres au niveau du noyau de MERMAID.

5.1.3 Agents administratifs

La norme FIPA propose trois agents de gestion : AMS, DF et ACC. Présentés en section 3.2.3, ils assurent l'organisation administrative et le contrôle de l'ensemble du système multi-agents. Ces trois agents administratifs se basent sur les bibliothèques de fonctions de l'API MERMAID de la même façon que nos autres agents embarqués. Cependant, ils sont les premiers à être lancés sur l'ensemble du SMA. En effet, les services qu'ils proposent vont permettre d'assurer le bon fonctionnement du système. Si la norme FIPA impose la présence de ces agents avec le respect des rôles qui leur sont associés, elle laisse cependant la possibilité de les adapter au contexte d'utilisation. Nous présentons donc ici nos adaptations spécifiques de ces trois agents.

5.1.3.1 AMS : le système de gestion d'agents

D'après les spécifications de la norme FIPA, l'agent AMS se charge de tenir à jour la liste de tous les agents évoluant sur la plateforme. Lors de son initialisation, un agent s'enregistre auprès d'AMS en lui indiquant ses informations dans un message. Il se désenregistre lors de sa terminaison par simple indication de son nom. La liste comporte les AID des agents, ces derniers regroupant deux informations : le nom de l'agent et son adresse IP avec un numéro de port. Dans le cadre d'un contact par D-Bus, le nom de l'agent suffit. L'adresse IP et le numéro de port sont utilisés pour un contact par Socket. Notre couche MTPS se charge de récupérer les informations nécessaires au transfert à partir de l'AID en fonction du système utilisé.

L'agent AMS peut également envoyer une requête d'arrêt à un agent. Nous adaptons cette capacité proposée par FIPA pour intégrer un service d'arrêt du SMA. Le principe est d'envoyer une requête d'arrêt à l'ensemble des agents enregistrés dans la liste de l'agent AMS. Actuellement, notre plateforme MERMAID ne dispose pas d'une interface graphique permettant de suivre le fonctionnement de nos agents. Nous incluons donc un service optionnel d'affichage de la liste facilitant le suivi de l'évolution du système multi-agents. Enfin, l'agent AMS permet de récupérer une identification AID en fonction du nom d'un agent. Ce service est particulièrement utilisé par l'agent administratif ACC lors du routage d'un message. La Table 5.6 détaille ces différents services proposés pour notre adaptation de l'agent AMS.

5.1.3.2 DF : le facilitateur d'annuaire

A l'instar de l'agent AMS, l'agent DF tient à jour une liste d'éléments correspondant aux agents évoluant sur la plateforme. Cette fois, un élément correspond à un service et à l'agent qui le propose. Par conséquent, un même agent peut être enregistré plusieurs fois dans la liste pour chacun de ses services. De même, un service peut être enregistré plusieurs fois si plusieurs agents le mettent à disposition.
Lors de son initialisation, un agent va donc envoyer un message à l'agent DF pour chacun des services qu'il propose afin de les enregistrer séparément dans la liste. A la terminaison d'un agent ce dernier envoie un message avec son nom au service de désenregistrement de l'agent DF. Il retire alors de sa liste tous les éléments associés au nom de cet agent.

Dans la norme FIPA, l'agent DF n'a pas de spécifications ajoutées à cette gestion des capacités des agents. Nous lui attribuons tout de même un service d'affichage de sa liste pour faciliter le suivi de l'évolution du SMA et notamment proposer une redondance en vérifiant que les noms enregistrés existent sur les listes d'AMS et de DF.

L'agent DF permet de récupérer le ou les noms des agents proposant un service donné. Comme plusieurs agents peuvent correspondre, l'agent DF peut renvoyer plusieurs réponses pour une seule requête. Dans ce cadre, il termine son interaction réponse en envoyant un dernier message de fin. Ce service est principalement utilisé par l'agent ACC pour le routage des messages. La Table 5.7 détaille les services offerts par notre agent DF.

Service	Description
register_AMS	Ce service requiert l'AID de l'agent. Par le biais de ce service, l'agent AMS intègre un nouvel élément agent dans sa liste. Ce service ne renvoie pas de messages à moins qu'une erreur n'empêche l'enregistrement effectif de l'agent.
achieve_AMS	Ce service est l'antagoniste du précédent. Il requiert uniquement le nom d'un agent. AMS retrouve l'élément correspondant au nom donné et le supprime de sa liste. Ce service ne renvoie pas de messages à moins qu'une erreur n'empêche le retrait effectif de l'agent.
send_add	Comme le service précédent, celui-ci requiert le nom d'un agent pour permettre à AMS de retrouver l'AID correspondant dans sa liste. Il renvoie ensuite cette information par le biais d'un message de type réponse.
print_list_AMS	Ce service est optionnel et ne requiert aucune information spécifique. Il affiche la liste des AID des agents actuellement enregistrés. Il ne renvoie aucun message après traitement.
quit_list_AMS	Ce service ne requiert pas non plus d'information particulière. Il parcourt sa liste et envoie un message d'arrêt à chaque agent y étant enregistré. Les agents viendront d'eux-mêmes se désenregistrer auprès du service "*achieve_AMS*" et seront donc retirés de la liste au fur et à mesure. Ce service ne renvoie aucun message après traitement.

TABLE 5.6 – *Services proposés par notre agent administratif AMS.*

5.1.3.3 ACC : le canal de communication entre agents

L'agent ACC est responsable du routage d'un message à un agent destinataire. En effet, pour limiter leur empreinte mémoire, les agents du SMA ne connaissent pas spécifiquement les autres agents de la plateforme. Toute interaction se fait à l'intention d'un service nécessaire à l'avancement de l'agent. Par contre, ils connaissent par défaut les trois agents administratifs. Dans le cadre de l'envoi d'une requête, le service de routage de l'agent ACC (*send_mess*) est contacté.

L'agent ACC propose également un service permettant l'envoi de plusieurs requêtes simultanées, une pour chaque réponse renvoyée par DF. Pour ce service, nous utilisons les informations d'encapsulation optionnelles détaillées en sous-section 3.2.3. Nous utilisons principalement ce service dans le cadre du protocole CNP et l'indiquons via l'option "Protocole". Pour permettre le suivi de plusieurs interactions simultanées, nous utilisons également l'option "Identifiant de conversation". La Table 5.8 résume les services spécifiques à notre agent ACC.

Nous illustrons le routage d'un message par ACC avec un exemple en Figure 5.2. Pour cet exemple, nous considérons que tous ces agents évoluent sur le même système embarqué et appartiennent donc au même environnement immédiat. Par conséquent, le système de communication utilisé est le protocole D-Bus. Ce transport est transparent pour les agents.

Service	Description
register_DF	Ce service effectue l'intégration d'un nouvel élément à la liste des services du SMA. Il requiert un nom de service et le nom de l'agent le proposant. Ce service ne renvoie pas de messages à moins qu'une erreur n'empêche l'enregistrement effectif du service.
achieve_DF	A l'inverse, celui-ci correspond à la suppression des services associés au nom d'agent reçu en paramètre. Plusieurs éléments pourront donc être retirés de la liste lors d'une seule requête. Ce service ne renvoie pas non plus de message après traitement à moins d'avoir rencontré une erreur.
send_name	Ce service permet d'obtenir le nom du ou des agents proposant un service donné. Comme le nombre d'agents pouvant correspondre est variable selon les cas, plusieurs réponses pourront être envoyées : une par agent concordant plus une indiquant la fin de la recherche.
print_list_DF	Ce service est optionnel. Il permet à la fois de faciliter le suivi de la liste des services supportés par le SMA et d'offrir une certaine redondance d'informations concernant les noms des agents enregistrés. En effet, ces derniers doivent également exister dans la liste gérée par AMS. Aucune réponse n'est générée par ce service.

Table 5.7 – *Services proposés par notre agent administratif DF.*

Service	Description
send_mess	Ce service permet le routage d'un message vers un agent destinataire à partir du nom d'un service que l'on souhaite contacter. Le message doit contenir 3 informations : le service cible, le type du message à transférer et le contenu du message à délivrer. Ce service se sert des traitements réalisés par le service "*send_name*" de l'agent DF [Table 5.7] et "*send_add*" de l'agent AMS [Table 5.6]. Si une erreur intervient à n'importe quel stade du traitement, le routage du message est compromis et un message d'erreur est renvoyé à l'agent initiateur de la requête.
send_Cfp	Ce service tire son nom de l'étape *Call for proposal* du protocole CNP présenté en sous-section 3.3.2. Il permet l'envoi de plusieurs requêtes simultanées à différents agents destinataires. Ce service se base sur le même mécanisme que le précédent tout en permettant la prise en compte de la totalité des retours de l'agent DF. Si une erreur intervient pour l'un des agents contactés, les interactions avec les autres agents destinataires ne sont pas impactées.

Table 5.8 – *Services proposés par notre agent administratif ACC.*

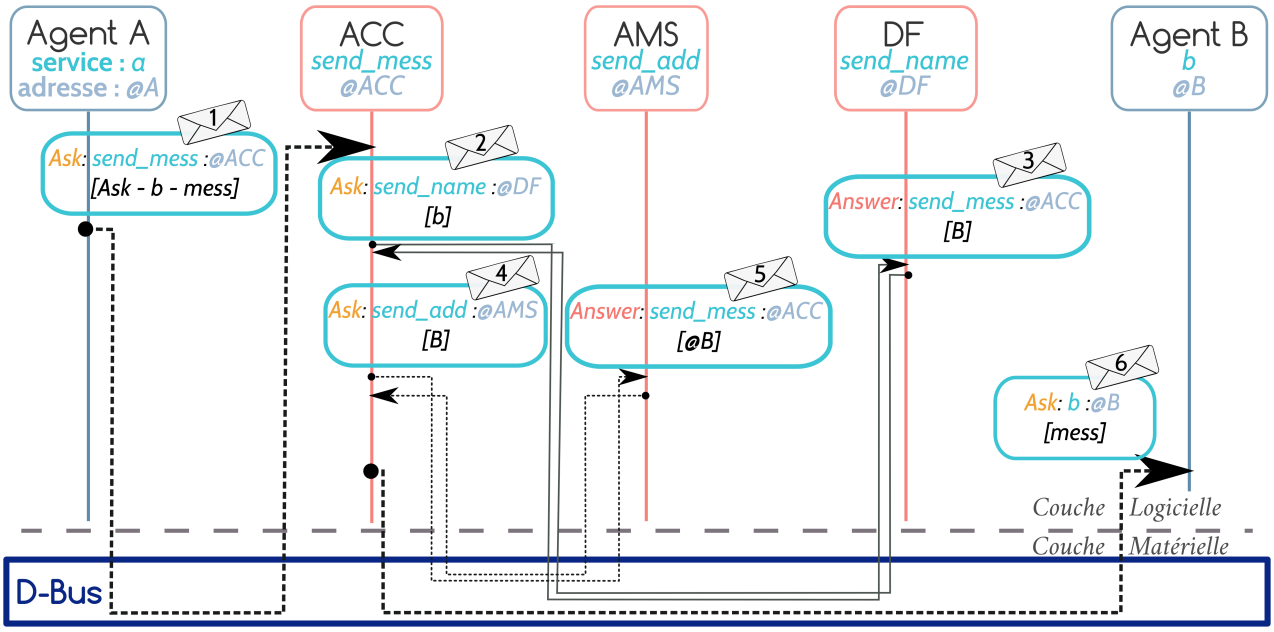


FIGURE 5.2 – *Diagramme présentant l'exemple du routage d'un message.*

Description de l'exemple présenté en Figure 5.2

Nous considérons deux agents applicatifs, A et B, proposant chacun un service différent : "a" et "b", joignables aux adresses "@A" et "@B". Les agents administratifs ACC, AMS et DF sont illustrés avec leurs services "send_mess" [Table 5.8], "send_add" [Table 5.6] et "send_name" [Table 5.7]. Nous indiquons leurs adresses respectives comme étant "@ACC", "@AMS" et "@DF".

L'agent A souhaite contacter le service b pour une requête avec le message "mess". A fait appel à ACC et lui transmet une requête avec les indications nécessaires au transfert *[message n° 1]*.
A partir du nom du service, l'agent ACC récupère en premier lieu le nom d'un agent offrant le service b. Il envoie à cette fin une requête au service "send_name" de DF *[message n° 2]*. Ce dernier traite sa demande et lui retourne en réponse le nom "B" *[message n° 3]*.

Une fois le nom de l'agent destinataire obtenu, ACC doit encore récupérer l'adresse pour le contacter. Il envoie une requête au service "send_add" d'AMS qui lui retourne l'information "@B" *[message n° 4 et 5]*.

Lorsque l'agent ACC a réuni toutes les informations nécessaires au routage du message d'origine, il reconstitue le message à transférer à partir des informations reçues lors de l'activation de son service par l'agent A (le type de message à envoyer et son contenu). Il transmet ensuite ce message final au service b de l'agent B *[message n° 6]*.

5.1.4 Couche applicative

La dernière couche représente les programmes agents basés sur MERMAID. En plus d'utiliser les bibliothèques de la couche d'administration et de gestion, chaque agent embarqué possède une base de connaissances et les fonctions des services de la famille d'agents à laquelle il appartient. Leur base de connaissances personnelle évolue ensuite de façon autonome.

En section 4.3.3 du précédent chapitre, nous avions présenté la formalisation de nos SMA embarqués. Leur implémentation dans MERMAID apporte de nouvelles interactions avec la prise en compte de nos agents administratifs. Dans la réalisation de notre matrice des interactions, nous devons donc intégrer ces nouveautés spécifiques à l'administration de notre SMA. Avec la Figure 5.3, nous reprenons le SMA utilisé pour illustrer notre formalisation pour représenter notre nouvelle matrice des interactions. Cette dernière inclut une part dédiée à la gestion administrative que nous distinguons de notre partie applicative. Au niveau de la liste des services, nous retrouvons ceux décrits dans cette section. Nous n'avons pas indiqué les services optionnels d'affichage de liste d'AMS et de DF car ils n'impactent pas l'évolution du SMA. Nous retrouvons pour chacun de nos agents applicatifs les enregistrements et désenregistrements auprès des agents AMS et DF. Par ailleurs, le contact entre l'agent Affichage et les agents Mémoire et TFD passe automatiquement par le service de routage d'ACC.

Source \ Cible	∅	AMS	ACC	DF	Agent Affichage	Agent Mémoire	Agent TFD
AMS	+ register_AMS + send_add + quit_list_AMS + achieve_AMS		+ Answer send_add				
ACC	+ send_mess + send_Cfp	+ Ask send_add		+ Ask send_name		+ Ask Gestion mémoire	+ Ask Calcul TFD
DF	+ register_DF + send_name + achieve_DF		+ Answer send_name				
Agent Affichage (1)	+ Affichage utilisateur	+Ask register_AMS + Ask achieve_AMS	+ Ask send_mess	+Ask register_DF + Ask achieve_DF			
Agent Mémoire (1)	+ Gestion mémoire	+Ask register_AMS + Ask achieve_AMS		+Ask register_DF + Ask achieve_DF	+Answer Gestion mémoire		
Agent TFD (1)	+ Calcul TFD	+Ask register_AMS + Ask achieve_AMS		+Ask register_DF + Ask achieve_DF	+ Answer Calcul TFD		

Liste des services des agents — *Gestion administrative* — *Application*

Figure 5.3 – *Matrice des interactions représentant la formalisation d'un SMA embarqué intégré dans MERMAID.*

5.2 | Organisation

Selon l'administration FIPA, détaillée en section 3.2.3, nous pouvons déployer plusieurs agents administratifs DF ou ACC, mais un seul agent AMS au sein d'un même système multi-agents. Pour limiter l'empreinte mémoire de notre SMA, nous avons choisi de conserver une seule occurrence de chaque agent administratif. Par conséquent, une seule partie administrative suffit à l'ensemble du SMA, même si ce dernier est étendu à plusieurs systèmes embarqués. De ce fait, pour l'extension de notre SMA à plusieurs systèmes embarqués, nous considérons deux blocs de l'architecture de MERMAID : le noyau fournissant le cadre de fonctionnement d'un agent embarqué et le bloc d'administration possédant en plus les agents AMS, ACC et DF du SMA.

Les SE sont connectés à un même réseau avec un SE central possédant la part d'administration à laquelle tous les autres SE du réseau sont reliés pour tout ce qui concerne l'administration (abonnement d'un agent, routage d'un message à un agent inconnu, etc.). Pour le reste, les agents sur les différents systèmes peuvent tout à fait interagir entre eux sans passer par ce SE central. Notre organisation peut s'appliquer à un ensemble hétérogène. De plus, MERMAID permet l'interopérabilité de ses plateformes : n'importe quel SE peut remplir le rôle central du moment qu'il embarque le bloc d'administration du SMA.

Les agents administratifs sont pris en compte pour le choix du système de communication. Jusqu'à présent, nous avons spécifié l'utilisation du réseau de sockets pour une interaction dans l'environnement hétérogène et du protocole D-Bus dans le cadre de l'environnement immédiat. Sur MERMAID, les agents ACC, AMS et DF sont connus de tous les autres agents. Cependant le SE les intégrant n'est, lui, pas une donnée par défaut ; il peut s'agir de n'importe quel système embarqué. Par conséquent, une communication à destination d'un agent administratif passera automatiquement par une socket pour assurer son routage. Une fois la communication établie entre deux agents, ces derniers pourront interagir via le réseau de communication adapté à leur environnement. Nous illustrons les liaisons aux différents systèmes de communication de nos agents dans le cadre d'un SMA étendu à plusieurs SE en Figure 5.4.

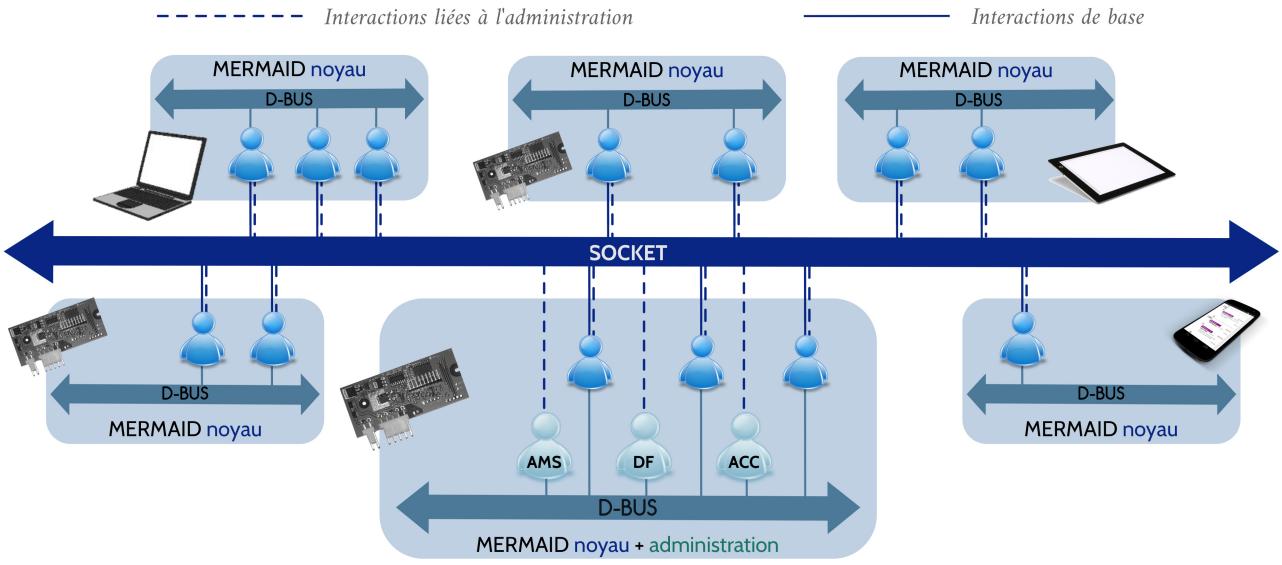


Figure 5.4 – *Schéma d'ensemble des communications entre plusieurs agents fonctionnant sur différents SE intégrant MERMAID.*

5.3 | Évaluation de la plateforme

5.3.1 Mesure du temps de transfert moyen de MERMAID

Notre première expérimentation vise à évaluer les performances de notre plateforme multi-agents embarquée. Pour analyser nos résultats, nous prenons en compte le critère de taille des données transférées [Jurasovic et al., 2006]. Nous déployons deux agents pour générer une interaction et relevons le temps de transfert moyen de cette dernière.

Pour cette expérience, nous avons en premier lieu effectué ce transfert au sein d'un même système. Nous avons ensuite réitéré notre expérience en répartissant nos agents sur deux systèmes pour mesurer le temps de transfert d'une communication externe.

Pour chaque cas d'étude, nous avons fait varier le poids des données transférées. Nous évaluons ce poids en considérant l'empreinte mémoire générée. Nos messages suivent l'encapsulation déterminée par le standard FIPA-ACL et présentée en sous-section 3.2.2. Nous varions ainsi d'un message encapsulé avec une empreinte minimale pour une taille de 312 bits [Table 5.9], à une matrice de 1024x1024 pixels composés chacun de trois nombres entiers soit 50331648 bits.

Paramètre	Type	Performative	Expéditeur	Destinataire	Répondre-à	Contenu
Contenu minimal	"Ask"	"0"	"a"	"b"	"c"	" "
Taille (bits)	24	16	8	8	8	8

Paramètre	Langage	Encodage	Ontologie	Protocole	ID conversation	Répondre-avec	En-réponse-à	Répondre-avant
Contenu minimal	"NONE"	"NONE"	"NONE"	"CNP"	"0"	"NONE"	"NONE"	"NONE"
Taille (bits)	32	32	32	32	16	32	32	32

Taille totale : 312 bits

Table 5.9 – *Empreinte mémoire minimale pour un message encapsulé sous MERMAID.*

Description de la Figure 5.9

Description des contenus obligatoires :
Le Type d'un message correspond à une requête ou à une réponse. La performative est représentée par un numéro. Les champs expéditeurs, destinataires et répondre-à correspondent à des noms de service de nos agents. Ils doivent comporter au moins un caractère. Enfin le contenu représente au moins un caractère même dans le cas d'un message vide.

Description des contenus optionnels :
Ces paramètres doivent être présents dans l'encapsulation même s'ils ne sont pas utilisés. Ils ont donc un contenu "NONE" de 4 caractères par défaut. Si le protocole CNP est employé son nom remplace cette valeur par défaut. L'identifiant de conversation est représenté par un nombre.

Les messages de petite taille correspondent à un contenu allant jusqu'à environ 600 caractères. Nous appliquons alors la fonction *perform_sending* présentée en sous-section 5.1.2. Au delà de cette limite, une simple chaîne de caractère n'est plus adaptée pour contenir les informations à transférer. Ces messages plus importants doivent donc se traduire par une structure de données à transférer. Nous utilisons alors la fonction *perform_sending_data*.

En interne, la fonction *perform_sending* passe par D-Bus. Avec la fonction *perform_sending_data* et dans le cadre des communications externes, le message passe par socket.

Il est à noter que ce temps de transfert comprend le travail de routage de l'agent ACC [Sous-section 5.1.3] et donc les interactions de nos agents administratifs. Les données transférées dans ce cadre ne sont cependant pas comptabilisées : nous considérons uniquement la taille du message d'origine.
Les résultats obtenus sont affichés en Figure 5.5.

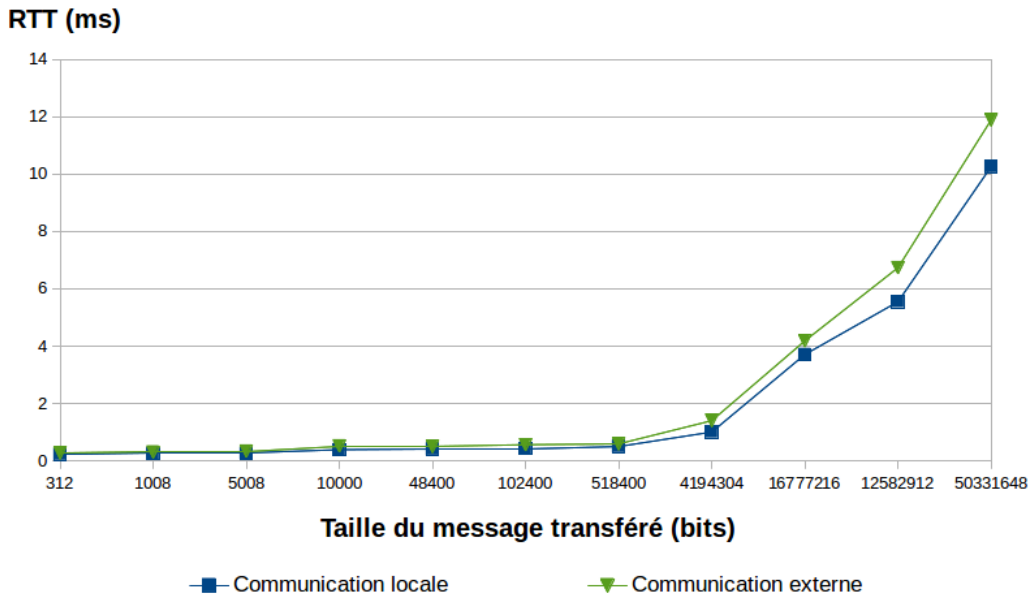


Figure 5.5 – *Mesure du RTT de la plateforme multi-agents embarquée MERMAID.*

Nous observons un temps légèrement plus important pour les communications externes. Pour les petits messages, la différence est infime. Pour les messages plus importants l'écart de performance augmente. Cependant, tous nos transferts restent inférieurs à 13 ms. En effet pour notre cas de message le plus coûteux, le temps d'un transfert externe monte à 11,8 ms contre 10,2 ms en locale. Cet écart correspond à une augmentation d'environ 16 % du temps de réponse pour une communication externe.

Nous constatons donc que les communications externes sont légèrement plus coûteuse en terme de temps de réponse, tout en gardant un écart non significatif vis-à-vis de nos temps de transferts moyens.

5.3.2 Mesure du temps de réponse des services de l'agent AMS

Nous évaluons à présent le temps de traitement des services principaux de notre agent administratif AMS. Nous mesurons ainsi les temps de réponse des services Enregistrement, Désenregistrement et Envoi d'un AID. Les temps de réponse sont relevés grâce à une instrumentation du code.

Pour cette expérimentation, nous déployons un nombre d'agents applicatifs variant de 5 à 50 agents. Nous étudions deux cadres d'interactions : l'activation d'une requête unitaire et l'activation de plusieurs requêtes simultanées pour le même service. Ce deuxième contexte correspond à une surcharge de demande pour l'agent AMS.

Pour le service d'enregistrement, nous créons une surcharge en démarrant nos agents en boucle. Nous utilisons le service *quit_list_AMS* [Sous-section 5.1.3] pour envoyer une demande de terminaison à tous les agents de la liste. Ces derniers vont alors tous venir se désenregistrer simultanément. Pour le service d'envoi de l'AID, nous démarrons un protocole CNP pour activer le service *send_Cfp* de l'agent ACC [Sous-section 5.1.3]. Ce dernier va ainsi demander en boucle à l'agent AMS les AID de tous les agents concernés par le protocole.
Nous affichons nos résultats en Figure 5.6.

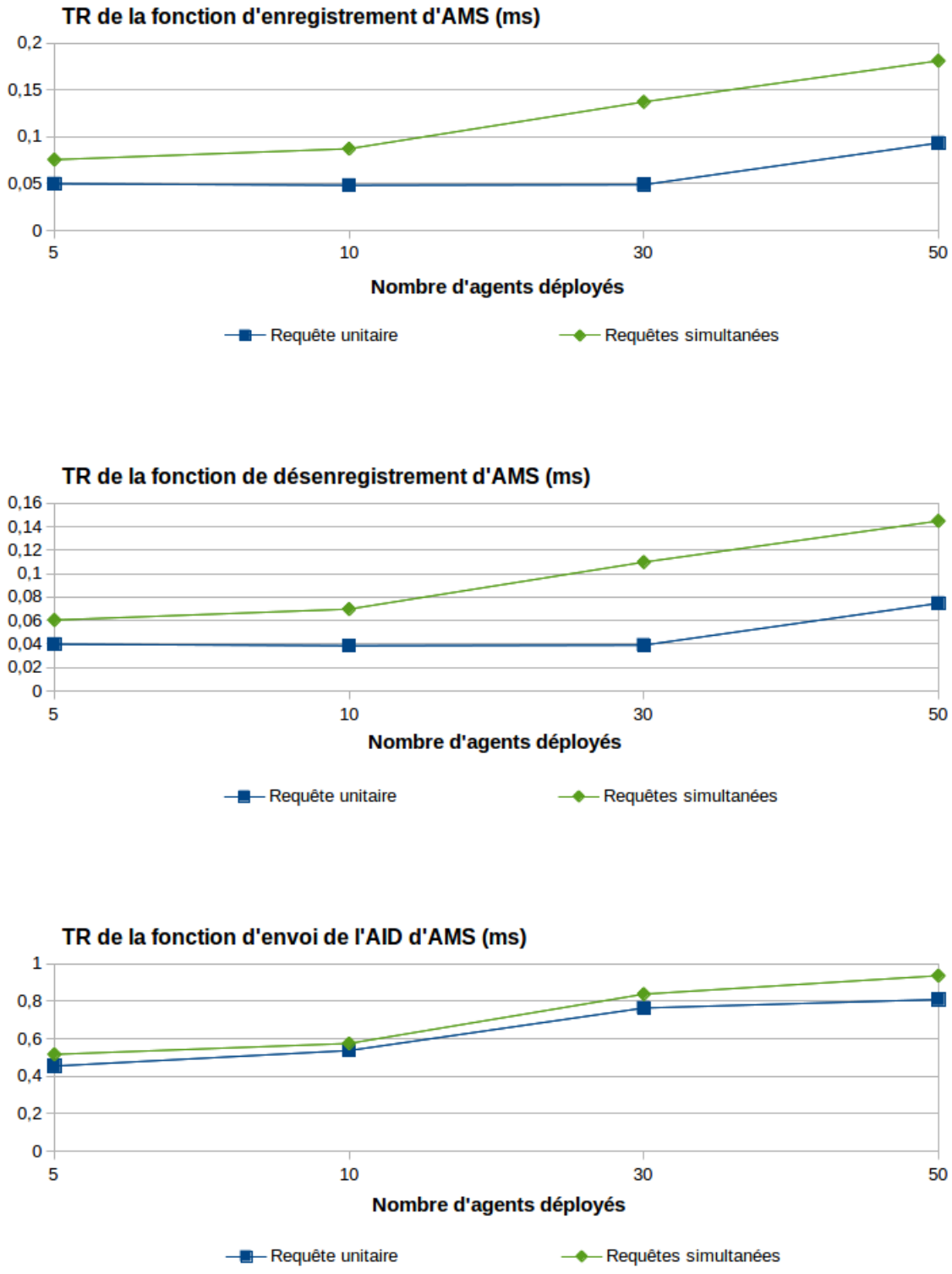


Figure 5.6 – *Évaluation des temps de réponse des services principaux de l'agent administratif AMS.*

L'ensemble de nos trois services proposent un temps de réponse rapide ne dépassant pas les 1 ms. Dans le cadre des services d'enregistrement et de désenregistrement, nous ne dépassons même pas les 0,2 ms. Globalement, nos temps de traitement augmentent avec le nombre d'agents déployés sur MERMAID au moment de la requête. En effet, les services d'AMS correspondent à la gestion d'une liste de ces agents. Plus ils sont nombreux et plus la liste est longue, augmentant ainsi le temps de recherche d'un élément. La surcharge de requêtes augmente légèrement le temps de réponse mais ne créée pas de pic correspondant à un goulot d'étranglement. La surcharge est particulièrement bien gérée dans le cadre de l'envoi de l'AID.

Les performances de temps de notre plateforme MERMAID expérimentées au cours de cette section valident la rapidité de notre plateforme. Les temps de transfert de communications locales ou externes sont faibles, avec un RTT légèrement plus élevé dans le second cas sans pour autant représenter un écart significatif. De plus, les services de nos agents administratifs représentent un coût moindre à l'échelle des multiples interactions de la plateforme.
Nous cherchons à présent à expérimenter des algorithmes multi-agents au sein des systèmes embarqués grâce à MERMAID.

Synthèse

Dans cette partie, nous avons formalisé nos SMA embarqués à travers leur cadre d'évolution, les spécificités de leurs agents et la présentation d'une méthodologie de conception.

La présentation du cadre d'évolution nous a permis de délimiter les environnements immédiats et hétérogènes pour des interactions respectivement au sein d'un même SE (locales) ou entre différents supports matériels (externes). Avec la spécification de nos agents embarqués, nous avons détaillé leur découpage en différents threads : un par service supporté par l'agent ainsi qu'un thread d'écoute du réseau. Le cycle de vie de ces threads se rapproche de celui d'une machine d'état temps réel pour ne pas entraver la réactivité du système embarqué.

Enfin, nous avons avancé notre méthodologie de conception d'un SMA embarqué en trois étapes. La caractérisation des familles d'agents prend en compte la limitation de l'empreinte mémoire de nos agents tout en respectant un compromis lié aux apports en termes de parallélisation du travail et de modularité. La table des besoins permet de déceler les informations manquantes à l'accomplissement des services d'un agent. Elle met en lumière les interactions qui seront nécessaires avec les autres entités partageant l'environnement d'un agent embarqué. Enfin, directement inspirée du modèle IODA, la réalisation d'une matrice des interactions nous permet de résumer les interactions inter-agents de notre SMA.

Dans un second chapitre, nous avons détaillé notre proposition d'une plateforme multi-agents embarquée : MERMAID. Nous avons présenté nos choix d'architecture, administratifs et organisationnels.

L'architecture se découpe en plusieurs couches indépendantes. La couche de communication est impactée par le cadre d'évolution de nos agents embarqués. Elle supporte deux systèmes : D-Bus pour les communications locales et les sockets pour les communications externes. Elle est rendue accessible via une API de communication correspondant à la couche MTPS de la norme FIPA.

La couche de communication est surplombée par la couche d'administration et de gestion. Regroupant différentes bibliothèques de fonctions, elle fournit aux agents embarqués des outils relatifs à leur cycle de vie, leurs interactions et la gestion des informations. Elle propose également des fonctions de normalisation suivant la norme FIPA-ACL. De plus, pour l'administration de la plateforme, cette couche intègre les agents de gestion AMS, DF et ACC issus de la norme FIPA. Nous considérons également une couche applicative regroupant les SMA embarqués basés sur MERMAID. Ensuite, nous avons détaillé l'organisation des différents SE employant MERMAID. En effet, cette plateforme est adaptée au contexte d'utilisation hétérogène. La spécificité d'un SMA ainsi étendu à différentes unités matérielles est la présence d'une seule occurrence de nos agents administratifs. Nous avons ainsi découpé l'architecture de MERMAID en deux blocs : un noyau comportant toutes les bibliothèques de fonctions et une partie administrative.

Enfin, nous avons évalué notre plateforme MERMAID en mesurant le temps de transfert moyen d'un message entre deux agents. Le routage de ce message était assuré par l'agent ACC et la couche MTPS de notre plateforme. Nous avons expérimenté ce transfert au sein d'un même système et entre deux systèmes distincts. Nous avons fait varier l'empreinte mémoire du message transféré. Nous avons observé un RTT plus rapide pour les communications locales tout en gardant un temps de transfert très faible, toujours inférieur à 13 ms, avec les communications externes.

Nous avons également mesuré les temps de réponse des fonctions administratives de l'agent AMS. Pour l'enregistrement, le désenregistrement et l'envoi d'un AID, les temps de réponse ne dépassent pas 1 ms. Le temps de traitement administratif est donc rapide notamment en comparaison du RTT de notre plateforme.

La formalisation de nos propositions étant achevée, nous souhaitons à présent vérifier nos hypothèses à travers des expérimentations.

Partie III
Expérimentations et analyses

Dans la section 2.1 nous avons détaillé les quatre défis principaux des travaux liés aux systèmes embarqués. Nous avons pu préciser en particulier les besoins liés à l'optimisation de la gestion des ressources de ces systèmes en sous-section 2.1.4. Ainsi, cette optimisation peut être effectuée via la gestion des tâches et en particulier par le biais de la délégation des tâches. Par ailleurs, nous avons relevé le manque de flexibilité des choix d'ordonnancement effectués dans les systèmes embarqués. Les critères tels que la réactivité et la robustesse sont également importants à prendre en compte.

Au cours de la partie précédente, en chapitres 4 et 5, nous avons décrit nos travaux relatifs aux systèmes multi-agents afin d'établir une plateforme agent embarquée : MERMAID. Nous avons évalué cette plateforme par le relevé du temps moyen de transfert d'un échange de messages entre deux agents (le RTT) et à travers l'analyse du temps de réponse des services de l'agent administratif AMS.

Nous présentons maintenant les expérimentations effectuées de SMA embarqués pour répondre aux besoins de flexibilité et de gestion des tâches des SE. Toutes nos expériences sont soumises au cas d'étude du traitement d'images avec la réalisation d'un zoom.

Dans un premier chapitre, nous éprouvons notre hypothèse concernant l'ajout de **flexibilité** dans les systèmes embarqués. Nous précisons le terme en définissant son apport aux SE et nous détaillons ensuite son application grâce à un SMA embarqué.

Dans un deuxième chapitre, nous proposons d'optimiser la gestion des ressources de notre SE, avec la **délégation des tâches**. Nous spécifions l'objectif de ce mécanisme et son intérêt dans le cadre des SE puis nous précisons notre hypothèse concernant l'application des SMA embarqués pour réaliser cette délégation. En effet, les agents se partageant le zoom de l'image pourront déléguer leur travail dans le cadre d'un contexte matériel désavantageux tel qu'un CPU surchargé.

Cette étude est menée sur un système embarqué de famille Cannes, puis est considérée dans le cadre d'une extension de notre SMA sur plusieurs SE hétérogènes. La délégation a alors lieu également entre les différents SE.

Chapitre 6

Distribution d'une tâche grâce aux SMA : apport de flexibilité

"La connaissance s'acquiert par l'expérience, tout le reste n'est que de l'information."

Albert Einstein
Mathématicien, Physicien, Scientifique (1879 - 1955)

6.1 | Contexte d'expérimentation

6.1.1 Cadre applicatif des expériences : le traitement d'images

Le choix du contexte applicatif vient de notre partenariat entre l'entreprise STMicroelectronics et le laboratoire CREN-Le Mans. L'entreprise développe, entre autres, des cartes embarquées dédiées à des traitements multimédia. Elle souhaitait expérimenter nos hypothèses dans le cadre applicatif de cette gamme de produits. Notre contrat de thèse CIFRE nous a donc permis de travailler dans un contexte d'évaluation et d'expérimentation industriel.

Dans le domaine du traitement d'images, le rendu est visuellement exigeant. Un mauvais calcul au niveau des pixels formant un contour de l'image, par exemple, sera visible par l'utilisateur. Une allocation adaptée des ressources est donc nécessaire.

Pour la gestion vidéo, part importante du domaine multimédia, le temps de réponse du travail de traitement d'une image devient crucial. Il faut respecter un certain nombre d'images par seconde (*frame rate*) nécessaire à un rendu fluide pour la perception visuelle humaine [Chen and Thropp, 2007]. Nous pouvons ainsi évaluer nos résultats en termes de temps de réponse avec une plage cible liée à ce frame rate standard.

6.1.2 Description du matériel utilisé : SE Cannes

Pour nos recherches, nous travaillons sur une carte STMicroelectronics de la famille Cannes, dédiée à des applications multimédia. En Figure 6.1 nous présentons un schéma de l'architecture du système intégré sur une puce, la SoC, de cette famille. Ce système embarqué intègre plusieurs composants liés à un traitement multimédia : un GPU MALI-400 quatre cœurs et plusieurs accélérateurs matériels utilisés pour l'encodage et le décodage d'images et de vidéos. Plusieurs formats de décodage sont supportés tels que le H264 et le HEVC en haute définition (1080 pixels de largeur), ou encore le JPEG, MP4, VC1, etc. Les formats d'encodage utilisés sont le H264 et le HEVC en haute définition.

Le CPU principal utilisé est un double cœur symétrique basé sur un multiprocesseur ARM de type cortex A9. Il incorpore un système d'exploitation Linux ainsi qu'un ensemble de drivers associés. Il est dédié au lancement des applications de la carte.

Les composants de type GPU et CPU ont été détaillés en section 2.1.4. Le CPU principal de cette carte embarquée, cadencée à une fréquence de 1,5 GHz, peut délivrer jusqu'à 7500 DMIPS. Le SE possède également plusieurs unités processeur, de type ST231, qualifiées de CPU secondaires et tournant à une fréquence de 600 MHz. Ces unités viennent

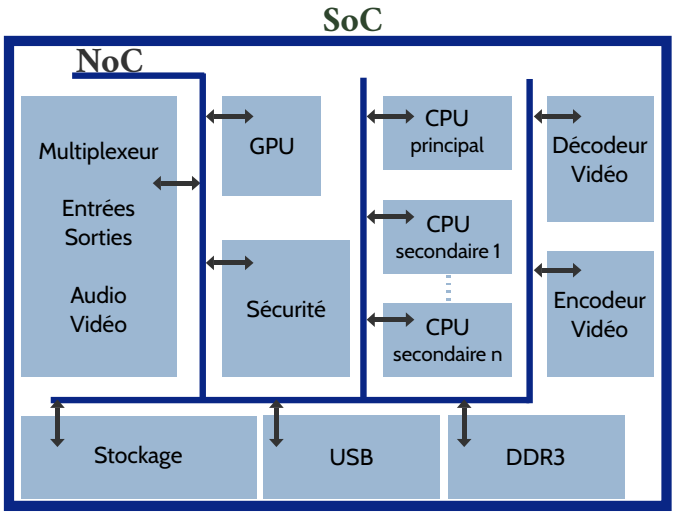


Figure 6.1 – *Architecture d'une carte embarquée de la famille Cannes produite par STMicroelectronics.*

en complément des autres composants du système. Elles sont dédiées à des tâches précises comme par exemple le décodage du format vidéo Delta non supporté par les accélérateurs intégrés, ou encore le traitement des codecs audio.

La plateforme Cannes utilisée pour les expérimentations contient un disque dur mémoire de type DDR3 de 32 bits capable d'un taux de transfert d'information maximal de 1866 Mbps (*million bits per second*) pour une fréquence de 933 MHz. Ce SE intègre également des ports de stockage de mémoire flash pour intégrer des cartes SD ainsi que des ports USB de type 2 et 3.

Un bloc de composants matériels est dédié à la sécurité de l'ensemble du système embarqué ainsi qu'à la gestion des données sensibles. Enfin, de nombreux ports d'entrées-sorties pour la vidéo, le son ou d'autres types de données permettent d'assurer la liaison avec l'environnement extérieur du SE.

Actuellement, la plateforme MERMAID tourne sur le processeur principal, le CPU ARM Cortex A9.

Pour certaines expérimentations, nous utilisons également un ordinateur doté d'un processeur Intel Core i7-2820QM montant jusqu'à 3,40 GHz avec une mémoire cache niveau 3 de 8 Mo cadencée à 2,30 GHz. L'appareil comprend également un disque dur mémoire de type DDR3 cadencé à 1333 MHz. Les spécifications de cet ordinateur se démarquent suffisamment de notre SE pour établir un contexte hétérogène d'expériences.

6.2 | Flexibilité des choix dans les SE

Dans la section 2.3 nous avons présenté différents algorithmes d'ordonnancement. Nous avons pu constater que ces algorithmes sont programmés de façon figée pour établir un tri selon des critères pré-définis. Ils ne prennent pas en compte les variations de leur contexte matériel ou de l'environnement utilisateur. Nous qualifions cet état de fait par un manque de flexibilité du système [Définition 17].

Définition #17 : *Flexibilité*

La **flexibilité** correspond à la capacité d'adaptation à un contexte en ayant plusieurs choix à disposition, fournis ici par les différents agents.

Pour réaliser un service nécessitant plusieurs actions, l'approche multi-agents permet de séparer ce service en plusieurs étapes, une pour chaque type d'action à effectuer et de rattacher un agent à chacune de ces étapes. Les agents interagissent pour échanger les informations nécessaires à la réalisation de chaque étape. Cette méthode permet d'effectuer certaines actions en parallèle, impliquant une réduction du temps de réponse du service, là où une approche séquentielle traite chaque action l'une après l'autre.

Pour chaque étape, plusieurs solutions matérielles peuvent être disponibles. Dans le cadre d'une approche de programmation déterministe, le choix de la solution à utiliser est pré-établi. Avec un système multi-agents, certains agents peuvent intégrer des solutions différentes pour une même étape. La solution employée, et donc l'agent appelé lors de l'exécution, peut ainsi varier en fonction du contexte ou des préférences de l'utilisateur. Ce choix est effectué par l'agent activant l'étape concernée en prenant en compte ce contexte d'exécution. Dans le cadre d'une programmation séquentielle, la solution matérielle à employer doit être prédéfinie. Pour éviter une empreinte mémoire de l'algorithme trop importante, il est rare de définir tous les contextes et leur solution spécifique. Le choix le plus fréquent est de déterminer la solution la plus adaptée à la majorité des cas et de s'y tenir. Une modification des choix réalisés nécessite alors la modification de l'ensemble du système.

La capacité d'interaction des agents permet de prendre en compte de nouvelles informations au fur et à mesure de l'évolution du SMA. Par exemple si une nouvelle solution matérielle était ajoutée, l'ajout d'un nouvel agent capable d'utiliser cette solution au SMA suffirait pour la rendre accessible. Avec la Figure 6.2, nous illustrons l'organisation multi-agents. Le système contient trois agents, chacun lié à une solution matérielle différente pour un même traitement *Sm*. Un quatrième agent chargé de déterminer la solution à employer pourra ainsi faire appel à l'agent le plus apte à combler les besoins du système en fonction du contexte. Par ailleurs, cette répartition des capacités favorise le traitement parallèle : l'agent prenant la décision peut choisir d'appliquer plusieurs solutions en même temps pour raccourcir le temps de réponse ou à des fins de robustesse du résultat.

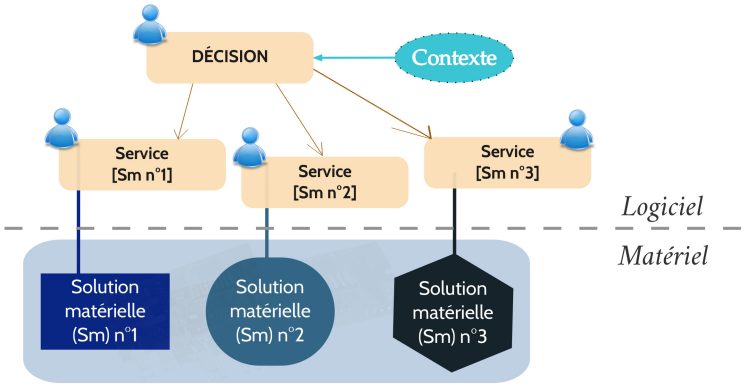


Figure 6.2 – *Flexibilité des systèmes multi-agents pour le choix d'une solution de réalisation.*

L'approche multi-agents permet donc d'obtenir un système évolutif avec une prise en compte dynamique du contexte.

A partir de ces constatations, nous posons l'hypothèse qu'utiliser des SMA au sein des SE permettrait d'apporter aux derniers la flexibilité caractéristique des premiers, ce qui reprend la proposition introduite dans la partie 1.5 de ce document :

Rappel hypothèse #4

La réalisation d'une tâche partagée entre plusieurs processus agents nous apportera de la flexibilité via différents choix d'exécution et l'adaptation au contexte d'exécution.

L'emploi d'un SMA doit nous permettre de nous adapter aux variations des contraintes de ressources de notre système embarqué et à l'évolution des besoins des utilisateurs en termes de résultat. En effet, un SMA peut, grâce à ses capacités d'adaptation, prendre en compte les données à sa disposition pour effectuer un choix d'algorithme de traitement pertinent.

Nous nous concentrons sur le cas du zoom sur une image impliquant une interpolation de pixels. La répartition du travail entre les agents doit pouvoir permettre une flexibilité au niveau du choix de la méthode de calcul pour une même image.

6.3 | Cas d'application : zoom sur une image

6.3.1 Présentation du programme séquentiel

La tâche observée est l'affichage d'une image et la possibilité de zoomer sur cette image en permettant à l'utilisateur de sélectionner une zone ciblée, de déclencher un agrandissement sur cette zone ou de revenir à l'image d'origine. Sur la Figure 6.3, nous présentons le diagramme de cette tâche selon une approche séquentielle.

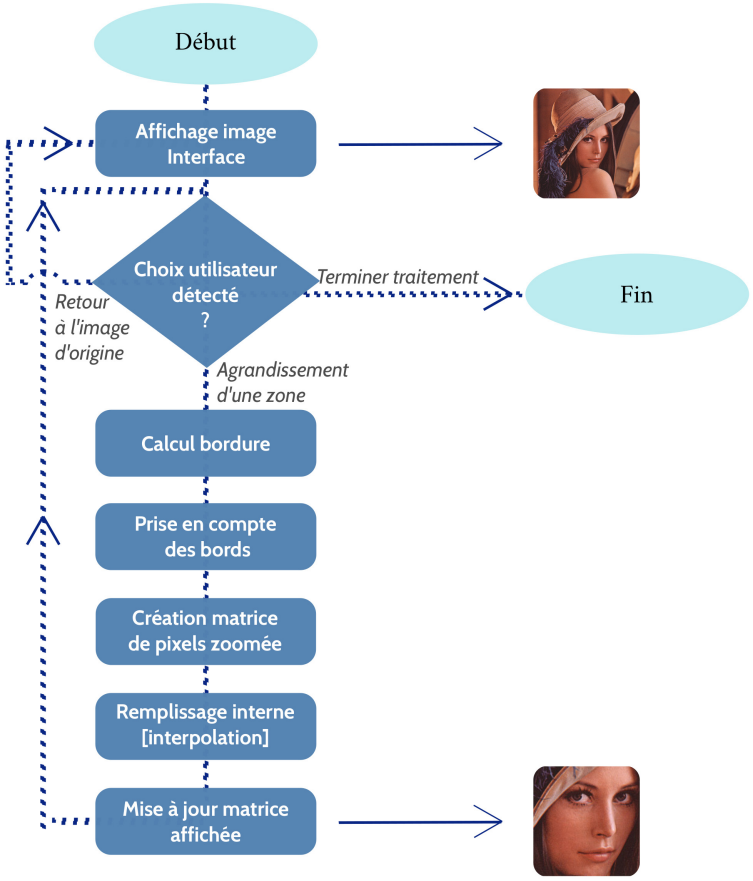


Figure 6.3 – *Diagramme de la tâche "zoom sur une image" traitée en plusieurs étapes selon une approche séquentielle.*

Nous y retrouvons sept étapes. La première correspond à l'affichage de l'image. Le programme se met ensuite en attente d'un choix de l'utilisateur pour passer à une autre étape. Une fois cette condition déclenchée, le programme vérifie si son arrêt n'a pas été demandé. Si ce n'est pas le cas, il analyse la nature de la demande pour déterminer s'il doit revenir à l'affichage d'origine ou effectuer un agrandissement sur l'image actuelle. Le deuxième cas représente la condition de déclenchement de l'ensemble des autres étapes, toutes dédiées à l'application du zoom.

Par l'action du calcul des bordures, le programme établit la taille de la zone à considérer. Elle dépend du niveau du zoom effectué. Dans notre cas, nous effectuons un grossissement de facteur deux. La taille de la zone prise à partir du point de sélection correspond alors à un quart de l'image d'origine. L'étape suivante permet de vérifier que la zone ainsi calculée ne dépasse pas les limites de l'image. Si un dépassement est constaté, la zone considérée doit être décalée. La particularité de cette étape est illustrée en Figure 6.4. Sur l'image de gauche on voit que la zone sélectionnée reste dans le cadre de l'image. Sur l'image de droite, le point de sélection étant trop proche du bord, la zone par défaut dépasse les limites du cadre.

Numériquement, une image correspond à une matrice de pixels. L'étape de création de la matrice de pixels zoomée correspond à la préparation du zoom. Un tableau à double entrée composé de pixels est créé. La taille de ce tableau

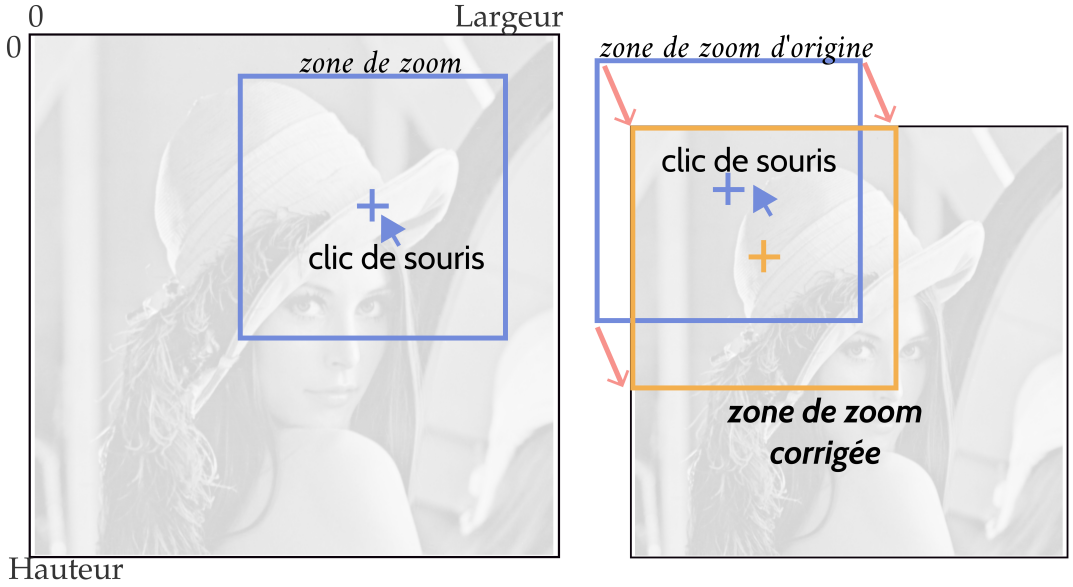


FIGURE 6.4 – *L'étape du calcul des bordures permet de vérifier que la zone ne dépasse pas les limites de l'image.*

correspond à la taille de l'image. Effectuer un zoom numériquement implique une action d'interpolation des pixels. Pour effectuer le zoom, l'ensemble des valeurs présentes dans la zone sélectionnée sont réparties dans le tableau créé en les espaçant par des cases vides. Le nombre de cases vides à intercaler correspond au facteur d'agrandissement moins un. Dans notre cas nous remplissons donc une case sur deux. Nous avons donc à l'issue de cette étape une matrice semi complète. Le calcul d'interpolation réalisé par l'étape de remplissage interne permet de compléter les cases vides à partir des valeurs conservées.

La Figure 6.5 illustre une interpolation sur une matrice de pixels. Au départ, nous avons notre image d'origine. La partie supérieure gauche est sélectionnée pour un zoom. Le deuxième tableau représente l'étape de création de la nouvelle matrice. Nous y retrouvons nos pixels présents dans le coin supérieur gauche de l'image d'origine répartis à raison d'une case sur deux. Le troisième tableau illustre l'application d'une méthode d'interpolation prenant en compte les valeurs des pixels d'origine pour compléter les cases vides. Enfin, le dernier tableau présente le résultat avec une matrice complétée.

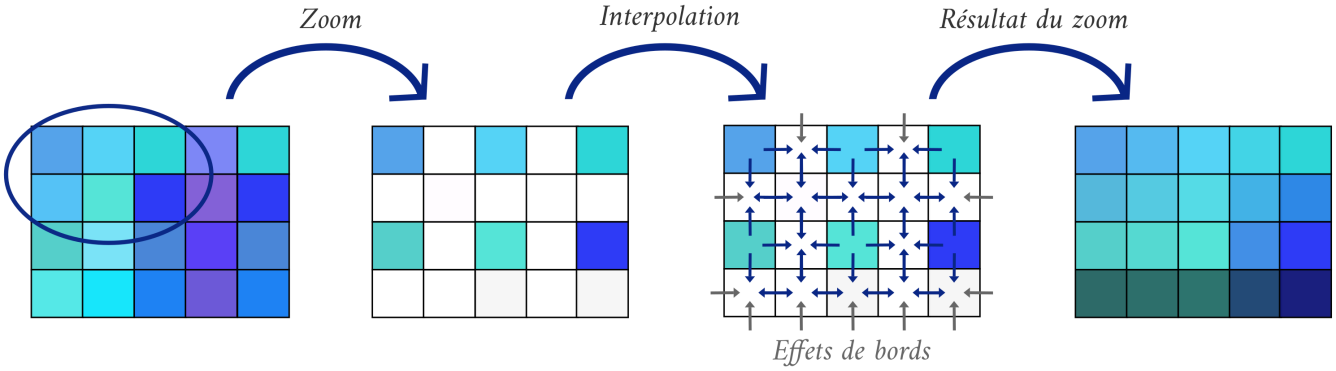


FIGURE 6.5 – *Principe d'interpolation des pixels nécessaire pour un zoom numérique.*

La dernière étape correspond à la mise à jour de la matrice de l'image affichée, l'ancienne matrice de pixels étant remplacée par celle résultant du zoom. Le programme reprend alors la surveillance de l'image pour détecter une nouvelle demande de l'utilisateur.

6.3.2 Méthodes d'interpolation utilisées

Il existe diverses méthodes d'interpolation plus ou moins complexes et donc plus ou moins précises. Dans le cadre de nos recherches, nous ne détaillerons pas spécialement cette partie car nous nous focalisons sur la possibilité d'avoir le choix entre plusieurs méthodes pour traiter une même image, plus que sur la méthode d'interpolation en elle-même. Pour notre cas d'étude, nous considérons ainsi deux méthodes d'interpolation impliquant des complexités différentes [Amri et al., 2009].

Pour évaluer la complexité de ces méthodes nous nous référons à la liste des instructions impliquant un cycle d'exécution et disponibles pour tout processeur. Nous relevons ainsi les affectations de valeur, les additions et les soustractions. Les multiplications et les divisions représentent des instructions spécifiques impliquant normalement plusieurs cycles d'exécution du processeur mais ramenées à un ensemble simplifié. Nous les comptons comme deux cycles d'exécution pour notre calcul de complexité.

- La méthode **du Plus Proche Voisin** PPV (*nearest-neighbor*) :

Cette méthode consiste simplement, pour chaque pixel manquant, à copier la valeur du pixel le plus proche. Par convention, parmi les choix possibles, le pixel voisin situé le plus à gauche est considéré comme le plus proche. Si cette valeur est indisponible, c'est le pixel voisin au dessus qui est sélectionné. Si ce dernier ne possède également pas de valeur, c'est le pixel voisin à droite qui est pris. Le pixel voisin en dessous est considéré en dernier recours. L'Équation 6.1 présente cette règle d'interpolation.

$$\begin{array}{lll} & & P_{x,y} = P_{x-1,y} \\ si & P_{x-1,y} = \emptyset, & P_{x,y} = P_{x,y-1} \\ si & P_{x,y-1} = \emptyset, & P_{x,y} = P_{x+1,y} \\ si & P_{x+1,y} = \emptyset, & P_{x,y} = P_{x,y+1} \end{array} \tag{6.1}$$

Dans le cadre d'une partie de l'image effectivement proche en valeurs de pixels, cette méthode donne un résultat tout à fait acceptable. En revanche, dès qu'un contour se dessine, l'interpolation par plus proche voisin créée des effets de crénelage (*aliasing*), c'est-à-dire un effet d'escalier au niveau des contours, et de mosaïque dans les dégradés. Ces effets sont considérés comme conséquence d'une faible qualité de zoom.

Concernant les instructions processeurs impliquées par cette méthode, nous ne relevons dans tous les cas qu'une mesure d'affectation. La complexité processeur de cette méthode est donc unitaire.

- La méthode d'interpolation **Bi-Linéaire** BL :

Le calcul correspond ici à prendre en compte les valeurs des 2 à 4 pixels les plus proches pour effectuer une moyenne. Pour ce faire, il faut considérer un point de repère, généralement situé en bas à gauche du pixel que l'on souhaite calculer, de coordonnées i pour l'axe de la longueur et j pour la largeur. Nous notons $P1_{i,j}$ ce point repère. Les trois autres points de repère seront les pixels situés en haut à gauche selon le même axe i mais à une largeur de j+1, $P2_{i,j+1}$, en bas à droite selon le même axe j mais à une longueur de i+1, $P3_{i+1,j}$, et enfin en haut à droite à i+1 et j+1, $P4_{i+1,j+1}$.

Nous considérons le pixel P placé aux coordonnées x et y telles que :

$$\begin{cases} i < x < i+1 \\ \quad et \\ j < y < j+1. \end{cases}$$

Par ailleurs pour simplifier la compréhension de la formule finale, nous posons :

$$\begin{cases} A = x - i \\ \quad et \\ B = y - j. \end{cases}$$

La valeur de $P_{x,y}$ est alors calculée à partir de l'Équation 6.2.

$$P_{x,y} = (1-A)*(1-B)*P1_{i,j} + A*(1-B)*P3_{i+1,j} + A*B*P4_{i+1,j+1} + (1-A)*B*P2_{i,j+1} \quad (6.2)$$

Cette méthode apporte un résultat bien plus précis au niveau des contours en proposant un dégradé des nuances là où la méthode d'interpolation PPV ne donne qu'un effet d'étalement des pixels existants.

- Comparaison des coûts algorithmiques des deux méthodes :

La méthode BL implique une formule comportant plus d'instructions processeurs : nous relevons 7 instructions de type addition ou soustraction et 8 multiplications. La résolution de cette interpolation implique donc une complexité processeur de 23 cycles. La méthode PPV représentant, elle, une complexité processeur unitaire, la méthode BL est donc 23 fois plus coûteuse en termes d'utilisation processeur.

(a) Image d'origine
Encodage : PNG
Taille : 180 x 180

(b) [x4] Interpolation par plus proche voisin (PPV)

(c) [x4] Interpolation bi-linéaire (BL)

FIGURE 6.6 – *Exemple d'interpolation numérique par méthode du plus proche voisin (b) et bi-linéaire (c) d'une image donnée (a).*

La Figure 6.6 propose deux exemples pour illustrer les deux méthodes d'interpolation que nous utilisons. Pour cette illustration nous choisissons une image de petite taille de type cartoon pour faire ressortir les différences de rendu des deux méthodes.

Nous proposons un système multi-agents capable de considérer ses connaissances (connaissances propres vis-à-vis des pixels considérés et connaissances du monde extérieur avec le contexte matériel) pour s'adapter et sélectionner la méthode d'interpolation la plus adaptée.

6.4 | Formalisation des agents embarqués

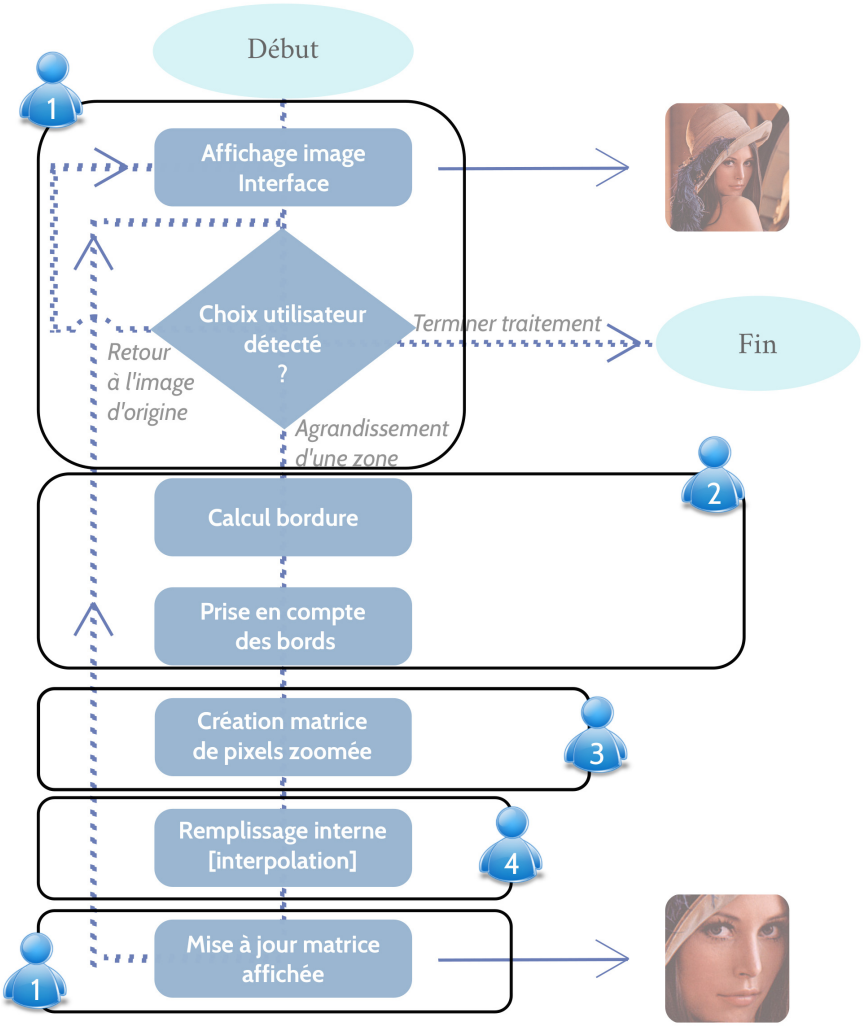


Figure 6.7 – *Application de la première étape : répartition des étapes d'un programme séquentiel en plusieurs familles d'agents.*

Pour réaliser notre SMA, nous suivons la méthodologie de conception présentée en section 4.3. Tout d'abord, nous établissons la distribution des étapes de notre tâche de zoom sur une image entre différentes familles d'agents, selon la formalisation présentée en sous-section 4.3.1. Nous considérons les sous-tâches selon la forme de service rendu, la base de connaissances requise et la possibilité de paralléliser certains traitements [Figure 6.7].
En reprenant notre diagramme du programme séquentiel, nous repérons ainsi les services suivants :

- **Agent n°1** : une gestion de l'affichage visuel et la perception des choix de l'utilisateur. Nous considérons ces actions comme relatives à l'image, et les regroupons sous la gestion d'un premier type d'agent. Nous l'appelons agent **Image** (IM).

- **Agent n°2** : une gestion des bordures du zoom pour éviter un dépassement du cadre de l'image. Calcul mathématique spécifique, cette partie est attribuée à un autre type d'agent que nous nommons agent **Bordure** (BO).

- **Agent n°3** : une gestion de la matrice correspondant à la zone de l'image sélectionnée et précédant les calculs d'interpolation. Nous considérons cette partie comme relative au zoom dans sa globalité et l'attribuons à un troisième type d'agent. Nous le qualifions d'agent **Zoom** (ZO).

- **Agent n°4** : enfin, le calcul d'interpolation des pixels de la matrice est attribué à un quatrième et dernier type d'agent que nous désignons par le terme d'agent **Pixel** (PI).

L'étape suivante, telle que décrite en sous-section 4.3.2, nous permet de constituer une table des besoins. Cette dernière liste les besoins en termes d'information ou de données de nos familles d'agents. Nous établissons quelle entité moyen (agent, processus, matériel, utilisateur, etc.) peut combler les besoins ainsi répertoriés. La Figure 6.8 représente la table des besoins construite pour ce cas d'étude.

Familles d'agents	Besoins	Moyens
Image	Calcul des bordures du zoom	Bordure
	Remplissage de la matrice zoomée	Zoom
Bordure	*Agent indépendant*	
Zoom	Remplissage de la sous-matrice zoomée	Pixel
Pixel	Niveau de disponibilité du CPU	Contexte matériel
	Obtention de la valeur d'un pixel voisin	Pixel

Figure 6.8 – *Deuxième étape : Établissement d'une table des besoins pour nos familles d'agents.*

Pour la famille d'agents Image nous listons deux besoins : le calcul de la zone de l'image à considérer avec la prise en compte des bords et le remplissage de la nouvelle matrice zoomée. Le premier besoin correspond au service rendu par les agents de la famille Bordure. Le deuxième besoin correspond au service rendu par les agents Zoom. Un agent Bordure ne requiert que les paramètres d'entrée de son service pour connaître les dimensions de l'image considérée et sa propre base de connaissances pour les calculs à appliquer. Cette famille est donc composée d'agents indépendants, c'est à dire qu'aucune interaction supplémentaire n'est nécessaire pour achever leur service. La famille d'agents Zoom n'a pas la connaissance requise pour effectuer les calculs d'interpolation. Elle a besoin d'interagir avec un ou plusieurs agents de la famille Pixel pour compléter la matrice zoomée. Enfin, un agent Pixel a besoin de connaître son contexte matériel pour déterminer quelle méthode de calcul est la plus pertinente. Pour éviter au maximum les effets de bords de sa sous-matrice, il doit également pouvoir récupérer les valeurs des pixels en bordure des autres agents Pixel.

A partir de la table des besoins nous relevons les interactions nécessaires inter-agents afin de réaliser une matrice des interactions selon les spécifications apportées en sous-section 4.3.3. Cette dernière résume ainsi les communications inter-agents qui auront cours dans notre SMA, ainsi que le type de communication : requête ou réponse. Cette partie nécessite également de préciser les services qu'une famille d'agents peut proposer aux autres agents du système.

De plus nous déterminons les besoins en termes de nombre d'instanciation d'agents pour chaque famille déterminée. Afin de mener à bien la tâche étudiée, une seule instance de chaque type est tout-à-fait suffisante. Le point considéré pour notre flexibilité de choix étant l'interpolation des pixels, nous choisissons d'avoir plusieurs instances du type d'agent Pixel (au moins 2). Pour nos protocoles d'évaluation, nous faisons varier ce nombre d'instances. Ici, nous indiquons une variable n pour représenter cette valeur. La Figure 6.9 affiche cette matrice.

Nous retrouvons les communications repérées à partir de la table des besoins au niveau applicatif ainsi que les différents services rattachés à chaque famille. Ainsi, l'agent Image propose le service "Zoom Image" retranscrivant sa capacité à présenter une image et le résultat d'un agrandissement effectué sur cette dernière. L'agent Bordure propose le service "Calcul bordure" pour établir les paramètres de la zone de matrice à considérer pour le zoom. L'agent Zoom met le service "Zoom matrice" à disposition. Ce dernier indique que la notion d'agrandissement concerne uniquement la matrice de pixels à gérer en tant que tableau de variables entières. Cet agent sépare sa matrice en plusieurs sous-matrices pour répartir le travail d'interpolation entre les n agents Pixel. Ces derniers proposent chacun un service "Remplir matrice" permettant d'effectuer les calculs d'interpolation nécessaires à la complétion des

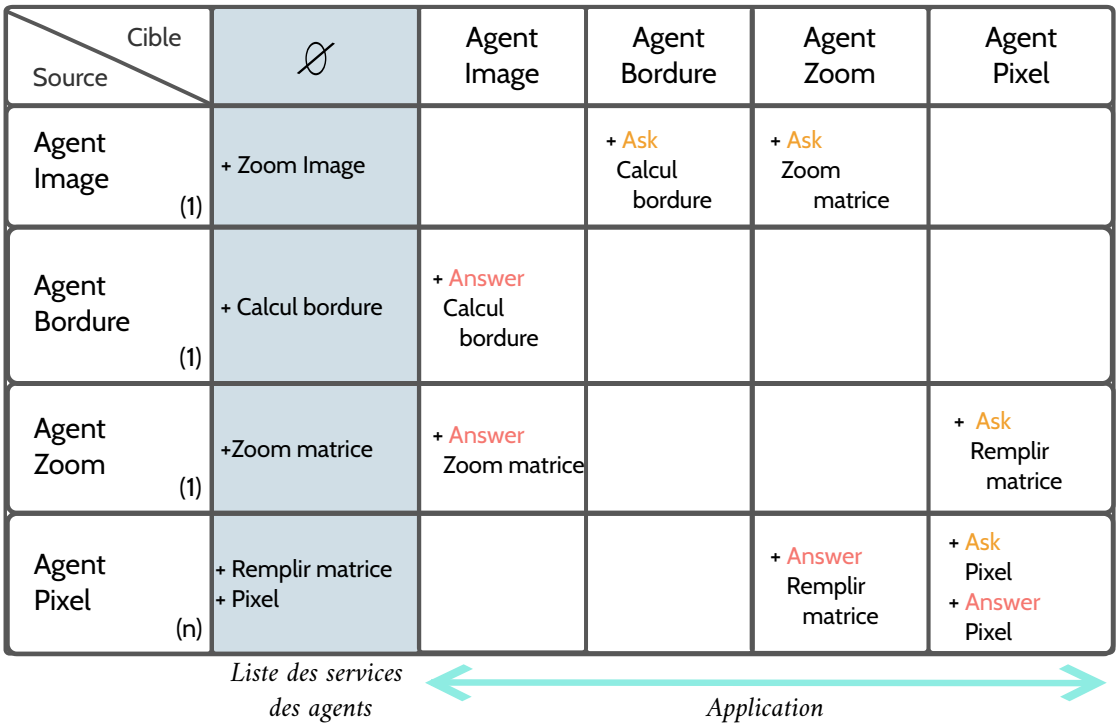


FIGURE 6.9 – *Réalisation d'une matrice des interactions (partie applicative du SMA uniquement).*

sous-matrices. Ils proposent également un service "Pixel" permettant de partager des valeurs de variable pixel entre agents de la même famille afin de compléter le service "Remplir matrice" en limitant les effets de bords. Les agents Pixel retournent leur sous-matrice complétée à l'agent Zoom lors de leur réponse. Celui-ci reforme alors une matrice zoomée complète et la renvoie en réponse à l'agent Image qui peut finalement afficher le résultat final.

L'agent Image envoie des requêtes aux agents Bordure et Zoom pour activer leur service. Une fois ces services achevés, les deux agents contactent l'agent Image avec des réponses pour lui envoyer les résultats obtenus. Entre temps, l'agent Zoom contacte lui-même les agents Pixel et reçoit des réponses de leur part. Les agents Pixel interagissent entre eux pour s'échanger des valeurs de variable si nécessaire.

6.5 | Application du zoom via MERMAID

Pour appliquer notre SMA formalisé au contexte de MERMAID, nous prenons en compte les interactions décrites en section 5.1.3, relatives à la partie administrative de notre plateforme. Nous obtenons ainsi notre matrice complète présentée en Figure 6.10.

Source \ Cible	∅	AMS	ACC	DF	Agent Image	Agent Bordure	Agent Zoom	Agent Pixel
AMS	+ register_AMS + send_add + quit_list_AMS + achieve_AMS		+ Answer send_add					
ACC	+ send_mess + send_Cfp	+ Ask send_add		+ Ask send_name		+ Ask Calcul bordure	+ Ask Zoom matrice	+ Ask Remplir matrice + Ask Pixel
DF	+ register_DF + send_name + achieve_DF		+ Answer send_name					
Agent Image (1)	+ Zoom Image	+Ask register_AMS + Ask achieve_AMS	+ Ask send_mess	+Ask register_DF + Ask achieve_DF				
Agent Bordure (1)	+ Calcul bordure	+Ask register_AMS + Ask achieve_AMS		+Ask register_DF + Ask achieve_DF	+ Answer Calcul bordure			
Agent Zoom (1)	+Zoom matrice	+Ask register_AMS + Ask achieve_AMS	+ Ask send_mess	+Ask register_DF + Ask achieve_DF	+ Answer Zoom matrice			
Agent Pixel (n)	+ Remplir matrice + Pixel	+Ask register_AMS + Ask achieve_AMS	+ Ask send_mess	+Ask register_DF + Ask achieve_DF			+ Answer Remplir matrice	+ Answer Pixel

Liste des services des agents — *Gestion administrative* — *Application*

FIGURE 6.10 – *Application de la troisième étape : matrice des interactions complète avec prise en compte de la partie administrative du SMA.*

La matrice complète ajoute les interactions concernant les agents administratifs AMS, DF et ACC. Tous nos agents applicatifs contactent les services *register* et *achieve* des agents AMS et DF afin de maintenir les listes de services et d'adresses à jour. Tout envoi de requête passe désormais par le service *send_mess* ACC. Ce dernier se charge de router les messages aux agents destinataires en utilisant les services *send_name* de DF et *send_add* de AMS. Les réponses se font toujours directement à destination de l'agent initiateur de la requête.

Une fois notre matrice complétée et avec la complémentarité de notre table des besoins, nous pouvons établir un diagramme du fonctionnement de notre SMA [Figure 6.11]. Nous le suivons pour développer les services de nos agents embarqués.

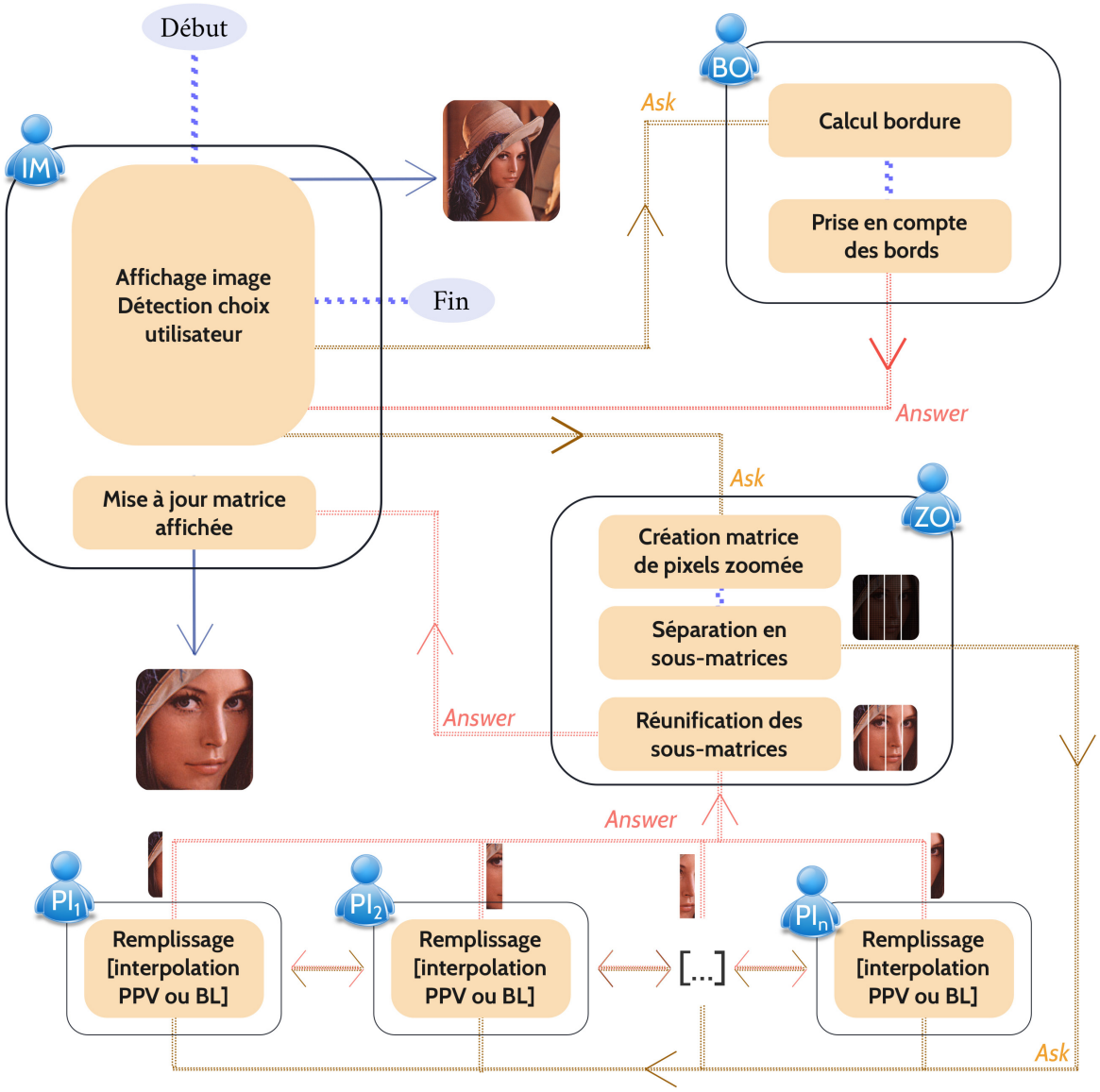


Figure 6.11 – *Diagramme de la tâche "zoom sur une image" traitée en plusieurs étapes selon une approche de programmation multi-agents.*

6.6 | Expérimentation de la flexibilité

6.6.1 Évaluation du temps de réponse du SMA

Protocole

En section 2.5 nous avons présenté les critères de performance du domaine des systèmes embarqués. Du fait de la complexité de relever avec précision la consommation du CPU et de la mémoire, nous choisissons d'évaluer la performance de notre solution au niveau du temps de réponse.

Nous utilisons une fonction d'instrumentation pour déterminer précisément le temps de réponse du programme. Le compteur se déclenche lors de la demande de zoom de l'utilisateur, et se termine à l'affichage de l'image agrandie. Nous nous plaçons en comparaison avec le programme séquentiel à partir duquel nous avons réalisé notre SMA embarqué. Nous récupérons son temps de réponse selon le même procédé.

Cette manipulation est effectuée sur deux images de différentes tailles, une de largeur 128 pixels par une hauteur de 128 pixels (que l'on note 128x128) et une de 512x512 toutes deux d'encodage PNG. Pour chaque image nous créons un nombre d'agents Pixel de deux au départ jusqu'à vingt pour la dernière expérimentation. Ainsi, nous souhaitons observer l'évolution du temps de réponse en fonction du nombre de pixels à traiter et du nombre d'agents Pixel actifs pour effectuer ce traitement. Notre solution multi-agents applique les deux méthodes d'interpolation présentées, le programme séquentiel applique la méthode bi-linéaire. Pour chaque expérience, nous répétons la procédure de zoom afin d'obtenir un temps de réponse moyen.

Pour nous donner une base de comparaison, nous considérons des temps limites à ne pas dépasser correspondant au domaine du traitement d'images. Si le service demandé à l'agent correspond au décodage d'une image pour l'affichage d'une vidéo, nous devrions respecter un seuil minimal nécessaire pour éviter un effet de saccade visuel de 18 images par seconde. Cette cadence requise pour un affichage temps réel correspond à un traitement par image d'une durée maximale de 55 ms.

Dans le cadre d'un traitement sur une image seule, la notion de contrainte sur le temps de réponse du traitement dépend du contexte applicatif. Nous nous plaçons dans le cadre du chargement d'une page Internet. Pour ce contexte, le résultat doit apparaître suffisamment rapidement pour ne pas créer de gêne pour l'utilisateur. Une lenteur d'affichage sera détectée par l'œil humain à partir de 250 ms. A l'inverse, en dessous de 100 ms, l'œil humain ne fera plus la différence et considérera simplement la réponse comme "instantanée". Nous établissons ainsi des temps de réponse repères avec une limite de 250 ms pour le traitement d'images et une limite de 55 ms pour du temps-réel lié à une vidéo.

Résultats

Les résultats des expériences visant à relever le temps de réponse des approches multi-agents et séquentielle, en fonction de la taille de l'image traitée et du nombre d'agents Pixel générés, sont visibles sur la figure 6.12.

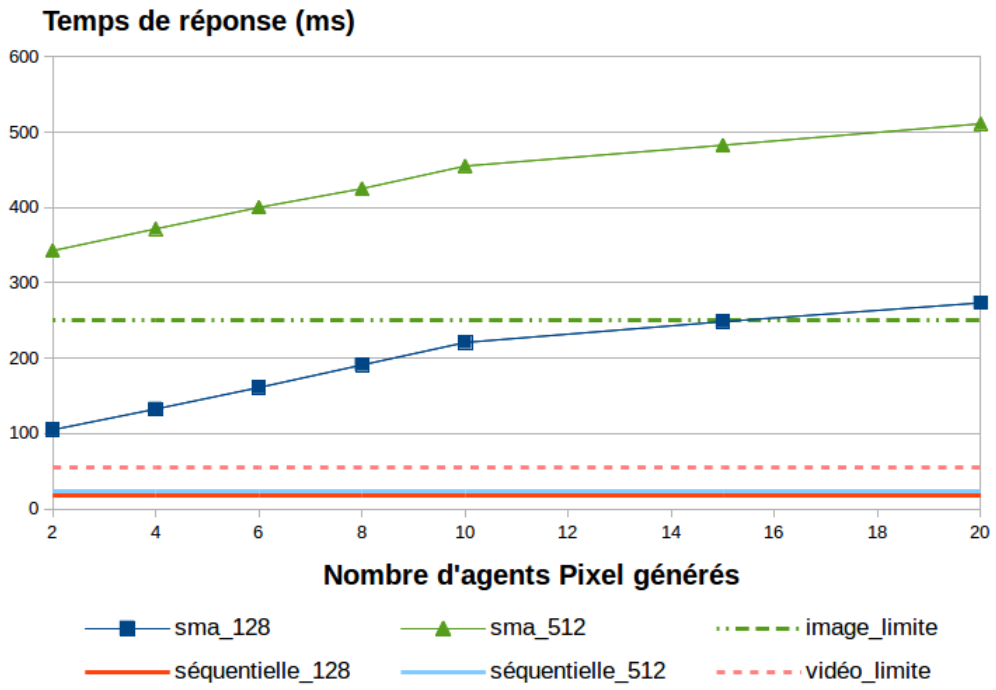


Figure 6.12 – *Mesure du temps de réponse pour la réalisation d'un zoom sur une image selon une approche multi-agents comparée au temps de réponse du même travail selon une approche séquentielle.*

Sur ce graphique, nous constatons que le nombre d'interactions influe grandement sur le temps de réponse. En effet, plus nous créons d'agents, plus le nombre de messages échangés dans notre SMA augmente ce qui impacte

directement le temps de réponse. Par ailleurs, plus la taille de l'image est importante, plus les interactions inter-agents Pixel pour s'échanger des valeurs sont conséquentes. Nous qualifions ce coût en temps de réponse relatif au transfert d'un message de coût administratif de notre système multi-agents. Si ce dernier reste sous notre limite liée à une image pour de petites matrices, il est directement hors champ pour notre image de 512 par 512 pixels. A partir de cette constatation, nous n'avons pas poussé l'expérimentation avec des images de qualité HD (1920x1080) ou supérieure. Par ailleurs, le temps de réponse est toujours au-dessus de notre limite liée au traitement vidéo.

En termes de performance pure, l'approche séquentielle est toujours la plus rapide. Nous considérons cependant que des optimisations peuvent être apportées au système pour diminuer ce coût administratif. Cette éventualité est précisée dans le chapitre 8.2 traitant des perspectives.

6.6.2 Qualité du rendu de l'image

Protocole

Nous proposons d'apporter de la flexibilité à notre zoom en offrant la possibilité d'appliquer deux méthodes d'interpolation différentes au sein d'une même image. Le choix de la méthode de calcul à appliquer est déterminé par l'agent en fonction de l'écart de valeur entre les pixels considérés et du contexte matériel de l'agent Pixel. Nos agents Pixel ont à leur disposition les deux méthodes d'interpolation présentées en sous-section 6.3.2 : l'interpolation par plus proche voisin (PPV) et l'interpolation bi-linéaire (BL).

La méthode PPV correspond à une simple affectation de valeur. Si nous considérons qu'elle représente une consommation CPU faible équivalent à 2 % de la capacité maximale du composant, alors la méthode BL, représentant une complexité en termes d'instructions processeurs 23 fois plus importante, implique une consommation CPU forte équivalent à 46 % de la capacité maximale du processeur.

Pour déterminer au cas par cas quelle méthode appliquer, chaque agent Pixel considère l'écart entre les deux valeurs voisines du pixel manquant. Nous considérons un seuil équivalent à la moyenne entre les valeurs minimales et maximales possibles. Si cet écart est supérieur au seuil considéré, nous estimons avoir affaire à un contour (*sharp edge*). L'agent Pixel applique alors la méthode bi-linéaire pour garder un résultat précis sur ces contours. Dans l'autre cas, les valeurs sont suffisamment proches pour utiliser la méthode du plus proche voisin consistant à recopier les pixels existants. Ces zones sont qualifiées d'uniformes (*smooth texture*).

Nos agents prennent également en compte leur contexte en observant la charge actuelle du processeur sur lequel ils tournent. Dans le cadre d'un CPU déjà utilisé à plus de 60 %, nous augmentons notre valeur de seuil au-dessus de la moyenne. Le résultat est moins précis, mais l'agent s'adapte ainsi à la disponibilité de son environnement matériel en réduisant au maximum l'utilisation de la méthode la plus coûteuse. Nous observons cette adaptation de nos agents en fonction de leur contexte matériel à travers trois expérimentations.

Nous effectuons d'abord une expérience avec un seul agent Pixel pour observer sa décision avec un seuil inchangé sur l'ensemble de la matrice zoomée. Nous réalisons ensuite une deuxième expérimentation avec cette fois-ci trois agents Pixel. Nous simulons une surcharge du CPU pour l'un d'entre eux. Le but est alors d'observer la différence de choix de l'agent Pixel pensant son CPU surchargé. Nous appliquons enfin une troisième évaluation avec dix instances d'agents Pixel, et simulons une surcharge CPU pour trois d'entre eux.

Résultats

Pour illustrer de façon claire les réactions de nos agents Pixel, nous faisons ressortir visuellement la méthode d'interpolation qui aura été utilisée. Pour la méthode du plus proche voisin, nous représentons notre case avec un pixel vert. Nous appliquons un pixel rouge dans le cas d'une interpolation bi-linéaire.

En Figure 6.13, nous présentons l'image d'origine sur laquelle nous effectuons nos agrandissements. Il s'agit d'une représentation de "Lena", une des images de référence pour les évaluations dans le domaine du traitement d'images.

Lena, image d'origine
Encodage : PNG *Taille : 512 x 512*

Zoom par 1 agent Pixel

FIGURE 6.13 – *Résultats d'un zoom, partagé entre plusieurs agents avec une interpolation effectuée par un agent Pixel, sur l'image standard "Lena".*

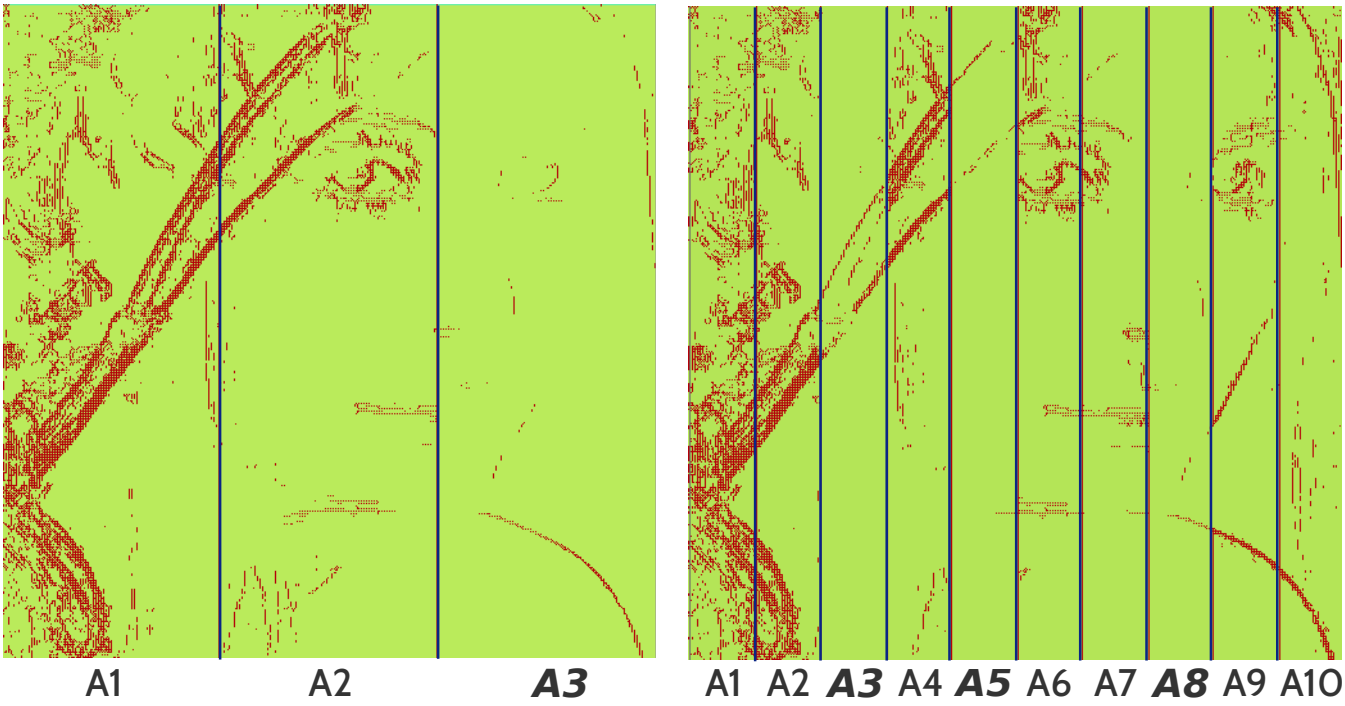
En effet, l'image regroupe plusieurs types de visuel comme par exemple des détails avec de nombreux contours ou des zones uniformisées. L'image possède également des variations de luminosité. L'ensemble représente donc une illustration particulièrement propice pour éprouver des algorithmes de traitement d'images.

La seconde image représente le résultat du premier test : agrandissement sur l'image d'origine par un agent Pixel sur l'ensemble de la matrice. Nous constatons que l'agent Pixel a effectué le choix d'appliquer l'une ou l'autre méthode d'interpolation selon son seuil de décision. Nous retrouvons ainsi visuellement les contours de notre image d'origine. Nous validons également la possibilité d'utiliser plusieurs méthodes au sein d'une même image. Nous pouvons augmenter le nombre de méthodes considérées en proposant de nouvelles familles d'agents maîtrisant ces nouvelles méthodes. Au lieu de reprendre le code existant des agents déjà définis nous intégrons ces nouveaux agents au sein du SMA. Ils sont alors reconnus par la plateforme ce qui rend leurs fonctionnalités directement disponibles.

Ces résultats confirment donc l'apport de nos agents en termes de flexibilité de choix d'algorithmes.

En Figure 6.14, nous montrons les résultats obtenus à gauche en effectuant le zoom avec trois agents Pixel et à droite avec une répartition de l'agrandissement entre dix agents Pixel. Nous pouvons observer le changement de seuil de décision dans le cadre des agents Pixel travaillant dans un contexte de surcharge du CPU : l'agent n°3 à gauche et les agents n°3, 5 et 8 à droite. La méthode d'interpolation bi-linéaire considérée comme plus coûteuse est effectivement beaucoup moins utilisée.

Ces résultats démontrent la capacité des agents à s'adapter à leur contexte et ce durant l'exécution. A travers ces résultats qualitatifs, nous montrons donc bien un apport en flexibilité telle qu'elle est spécifiée en Définition 17.



Zoom par 3 agents Pixel
(Agent n°3 avec CPU à plus de 60 %)

Zoom par 10 agents Pixel
(Agents n°3, 5 et 8 avec CPU à plus de 60 %)

Figure 6.14 – *Résultats d'un zoom, partagé entre plusieurs agents avec une interpolation effectuée par 3 (à gauche) et 10 (à droite) agents Pixel, sur l'image standard "Lena". Certains agents Pixel se voient attribuer un contexte d'exécution peu favorable et réagissent en conséquence.*

6.7 | Synthèse de l'expérimentation

Il est cependant important d'analyser ces résultats en prenant en compte les apports qualitatifs de l'approche multi-agents. En effet, dans le cadre du traitement d'images comme dans d'autres domaines applicatifs, un surcoût administratif peut être envisagé pour obtenir un résultat plus précis ou une exécution plus flexible et plus modulaire.

A travers ces résultats quantitatifs nous soulevons donc l'importance d'utiliser l'approche multi-agents au sein des systèmes embarqués en considérant toujours un objectif d'équilibre entre les avantages et les inconvénients de cette solution.

Dans le cadre de cette expérimentation nous relevons donc un coût administratif élevé estompé par un apport en flexibilité validé. Nous allons à présent nous focaliser sur cet apport pour voir si nous pouvons l'appliquer aux SE en tant qu'ensemble matériel et notamment en ce qui concerne l'optimisation de la gestion des tâches.

Chapitre 7

Délégation de tâches par SMA entre cartes embarquées

"La sagesse est fille de l'expérience."

Léonard De Vinci
Architecte, Artiste, Ingénieur, Peintre, Philosophe, Scientifique, Sculpteur (1452 - 1519)

Nos expérimentations du mécanisme de délégation de tâches par un SMA dans le contexte du traitement d'images se déroulent selon trois étapes :

- dans un premier temps, nous éprouvons la performance du protocole multi-agents pour obtenir une délégation de tâches ;
- ensuite, nous reprenons les évaluations visuelles en considérant la possibilité de coupler la flexibilité présentée au chapitre précédent avec le mécanisme de délégation ;
- enfin, nous étendons cette délégation à un contexte hétérogène en établissant notre SMA sur différentes cartes embarquées.

7.1 | Délégation dynamique des tâches

Nous avons abordé l'optimisation de notre système embarqué par l'angle de la gestion des tâches dans la section 2.1 et plus précisément en introduisant l'idée d'une délégation des tâches dans la section 2.2 [Définition 5]. Nous intervenons alors après le travail de l'ordonnanceur, ce dernier étant intégré en phase de conception au système embarqué. Nous proposons une éventuelle correction des choix de ce dernier une fois les paramètres liés au contexte considéré. Nous incluons le terme "dynamique" pour souligner cet apport en réactivité induit par l'application d'un SMA.

Dans la section 2.2, nous expliquons que certaines ressources des systèmes embarqués sont capables d'effectuer des tâches communes. Ces ressources sont pourtant dédiées à une forme de tâche spécifique. Ce choix de conception s'explique par une préférence de robustesse du produit, c'est à dire l'assurance d'un service rendu, par rapport à une réelle optimisation des capacités du SE.

Nous souhaitons apporter la possibilité pour des unités matérielles partageant les mêmes capacités à résoudre certains types de tâches de pouvoir se répartir le travail. La Figure 7.1 illustre cette possibilité.

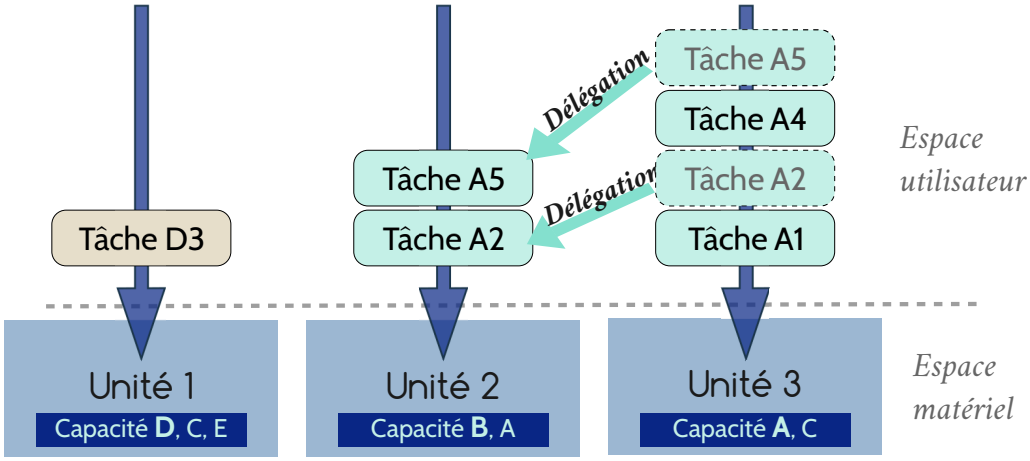


FIGURE 7.1 – *Principe de délégation de tâches de type A entre unités matérielles possédant la capacité pour résoudre ce type de tâche.*

Le schéma présente trois unités matérielles avec des capacités pour différents types de tâches notées A, B, C, D et E. Par souci de robustesse, chaque unité a été dédiée à un type de tâche en particulier :
- l'unité 1 est capable de traiter les tâches de type D, C et E et est dédiée aux tâches de type D ;
- l'unité 2 est capable de traiter les tâches de type B et A et est dédiée aux tâches de type B ;
- l'unité 3 est capable de traiter les tâches de type A et C et est dédiée aux tâches de type A.

Lors de l'utilisation du système embarqué, nous représentons sur la figure le cas d'une surcharge d'affectation de tâches de type A à la troisième unité matérielle. Sans délégation, le temps de résolution total est la somme des temps de résolution de chaque tâche. Avec une délégation, l'unité matérielle peut ré-affecter certaines tâches à une autre unité matérielle possédant la capacité adéquate. L'unité 2 a la capacité de résoudre les tâches de type A. Les tâches A2 et A5 lui sont donc déléguées pour optimiser l'emploi du SE et réduire le temps de résolution total.

Pour effectuer la délégation, nous devons instaurer une possibilité de discussion entre nos unités matérielles. Cette discussion doit permettre d'établir à quelle unité matérielle la tâche sera déléguée ainsi que les paramètres à prendre en compte pour la décision de délégation. Nous rappelons notre proposition concernant la délégation des tâches, introduite dans la partie 1.5 de ce document :

Rappel hypothèse #3

Les processus pouvant négocier une délégation des tâches allouées, par le biais d'interactions multi-agents entre différents systèmes embarqués, optimiseront la gestion des tâches de chaque SE de façon réactive.

En section 3.3, nous avons présenté les différents cas de convergences régissant les SMA. Dans notre cadre d'application, nos composants peuvent posséder des buts différents. Ils partagent des ressources communes qui pourront être suffisantes ou insuffisantes selon la capacité du SE impliqué. Enfin, afin de déléguer certaines tâches face à un afflux de requêtes, nous estimons les compétences de chaque composant comme insuffisantes. Nous nous plaçons donc dans le cadre de comportements sociaux de type négociation.

Nous proposons donc d'utiliser les algorithmes multi-agents de type négociation présentés en section 3.3.1 pour établir la délégation tout en satisfaisant les besoins à la fois d'optimisation et de robustesse des SE.

Nous considérons actuellement la caractéristique "indisponible" pour déclencher la délégation. Concrètement, lorsqu'une tâche est en cours de traitement, l'unité passe en état "indisponible". Si une tâche est assignée à une unité indisponible, cette dernière cherchera à la déléguer.

En perspectives, nous discuterons de l'intégration d'autres paramètres de décision liés aux capacités de l'unité considérée et aux préférences de l'utilisateur. Pour ce travail de thèse, nous nous concentrons sur la possibilité d'utiliser un protocole de négociation multi-agents pour effectuer la délégation. Nous avons choisi d'utiliser le protocole CNP également détaillé dans la partie 3.3.1.

Nous avons présenté l'intérêt de la délégation de tâches pour un système embarqué en prenant l'exemple des unités matérielles. A présent, nous considérons ce processus au niveau de nos agents reliés au même composant matériel : le CPU. Par le biais d'expérimentations entre plusieurs cartes embarquées nous retrouverons l'apport décrit. Pour cette délégation entre agents, les tâches correspondent à la réalisation de services.

La Figure 7.2 présente le schéma du mécanisme de délégation de tâches employé entre les agents sur notre plateforme. Nous y présentons des agents de la même famille offrant un service *"S"*, ainsi qu'un service *"S CNP"* associé. Leur service *S* passe en état indisponible durant le traitement d'une tâche. S'ils reçoivent une autre requête durant ce traitement, ils activent le service CNP associé pour chercher à déléguer la tâche.

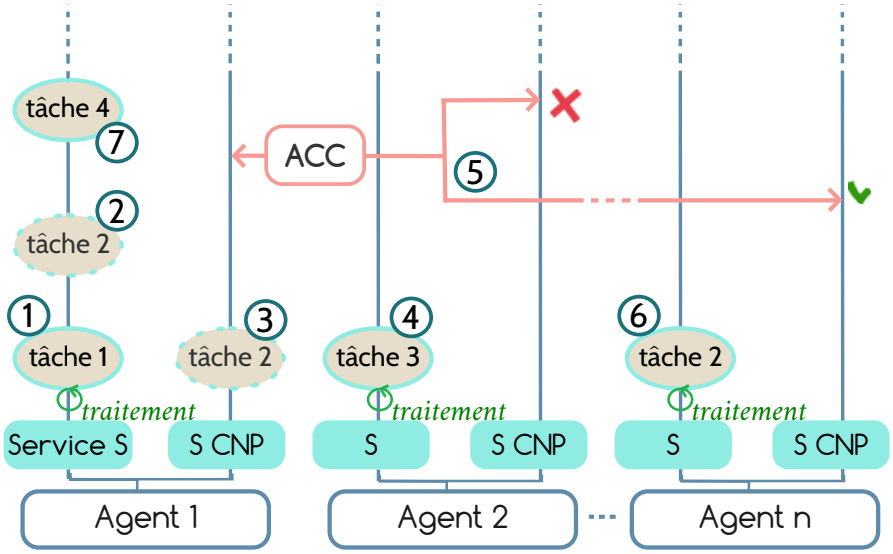


FIGURE 7.2 – *Mécanisme de délégation des tâches entre agents proposant le même service S grâce au protocole CNP.*

Description de la Figure 7.2

L'étape **(1)** correspond à la réception d'une tâche par l'*Agent 1* pour son service *S*. Étant alors disponible, il entame le traitement nécessaire pour l'achever. L'étape **(2)** indique l'arrivée d'une nouvelle requête. Toujours occupé par la première demande, l'*Agent 1* passe cette tâche à son service CNP en étape **(3)**. Il engage ainsi un processus de négociation visant à déléguer la tâche 2 à un autre *Agent* proposant le même service *S*.
Pendant ce temps, en **(4)**, l'*Agent 2* a lui-même reçu une demande et, étant disponible, a commencé à travailler à sa résolution. En étape **(5)**, le processus CNP est activé. L'agent *ACC* se charge de la première étape du protocole pour contacter les autres agents de cette famille. La négociation aboutira à un échec avec l'*Agent 2* déjà occupé, mais à une entente avec l'*Agent n* qui était disponible. En **(6)**, la tâche 2 est donc envoyée par l'*Agent 1* au service S de l'*Agent n*.
En **(7)**, une nouvelle requête arrive pour l'*Agent 1*. Ce dernier suivra le même cheminement de résolution (n'étant toujours pas disponible). Cette fois-ci, il ne recevra cependant que des refus lors de la négociation. La tâche sera donc placée en attente de l'*Agent 1* dans sa file de réception d'origine : celle du service *S*.

Dans un premier temps, nous travaillons sur un seul système embarqué. Nous nous servons de l'application sur une même carte pour valider notre hypothèse concernant le choix du protocole de négociation. Dans un second temps, nous avons étendu la possibilité d'établir notre plateforme MERMAID sur les CPU de plusieurs systèmes embarqués afin que la délégation puisse s'effectuer d'un agent sur une première machine vers un agent sur une seconde carte. Pour expérimenter ce mécanisme, nous adaptons notre précédent cas d'application de traitement d'images : la réalisation d'une interpolation.

7.2 | Évaluation du temps de réponse du protocole de négociation

Notre première expérimentation est l'évaluation en termes de performance du protocole de négociation multi-agents, CNP, utilisé pour effectuer la délégation. Dans un premier temps, nous mesurons le temps de réponse du protocole sans le traitement découlant du zoom. Cette expérience nous permet d'évaluer le coût administratif du mécanisme.

7.2.1 *Formalisation du SMA embarqué et application à MERMAID*

Nous repartons du SMA précédemment réalisé et décrit en section 6.4 et reprenons les étapes de réalisation d'un système multi-agents embarqué décrites en section 4.3 pour construire notre nouveau SMA. Comme nous nous concentrons sur les aspects de délégation, nous plaçons de côté les intérêts liés à la flexibilité au cœur de notre précédente formalisation. Pour limiter les interactions en dehors du protocole CNP, nous regroupons les services des précédentes familles d'agents "Zoom" et "Pixel" dans le service de traitement principal de la famille "Image". Notre agent "Bordure" ayant été déterminé comme indépendant, il est conservé. Par ailleurs, nous intégrons à la famille "Image" un nouveau service pour la mise en place du protocole CNP.

À l'étape de la table des besoins, le service de négociation fait apparaître un nouveau manque : la nécessité de connaître la disponibilité des autres agents partageant la capacité de résoudre un service. En effet, la prise de décision concernant l'agent auquel la tâche sera déléguée nécessite de nouvelles interactions. Des modifications sont apportées en conséquence et la Figure 7.3 représente notre nouvelle table des besoins.

Familles d'agents	Besoins	Moyens
Image	Calcul des bordures du zoom	Bordure
	Disponibilité des autres agents Image	Image
Bordure	*Agent indépendant*	

Figure 7.3 – *Table des besoins pour le cas de délégation de la tâche "zoom sur une image".*

Nous finalisons la formalisation de nos agents embarqués avec la création de la matrice complète des interactions [Figure 7.4]. En comparaison de notre précédent cas d'études, l'évaluation du protocole CNP nécessite de faire varier le nombre d'agents concernés. Nous établissons donc n instances d'agents Image.

Dans cette nouvelle matrice, l'agent ACC se charge d'effectuer la partie "d'appel" du protocole CNP : le *Call for proposal* (Cfp). En effet, il se renseigne auprès de DF et AMS pour établir la liste des agents offrant le même service "Zoom Image" que l'agent initiateur de l'appel. Il est ainsi en mesure de relayer l'appel au service CNP lié pour tous les agents concernés. Les agents continuent ensuite les étapes du protocole de négociation avec des messages de type réponse directement avec l'agent initiateur.

Pour déterminer la poursuite du protocole, il nous fallait déterminer les raisons poussant un agent à accepter ou rejeter une proposition. Nous avons identifié plusieurs paramètres intéressants tels que l'état d'un agent, le pourcentage d'utilisation de la ressource processeur, une méthode de calcul spécifique ou encore la localisation d'un agent dans le cadre du multi-cartes. Pour la validation de notre hypothèse concernant le protocole CNP, nous avons

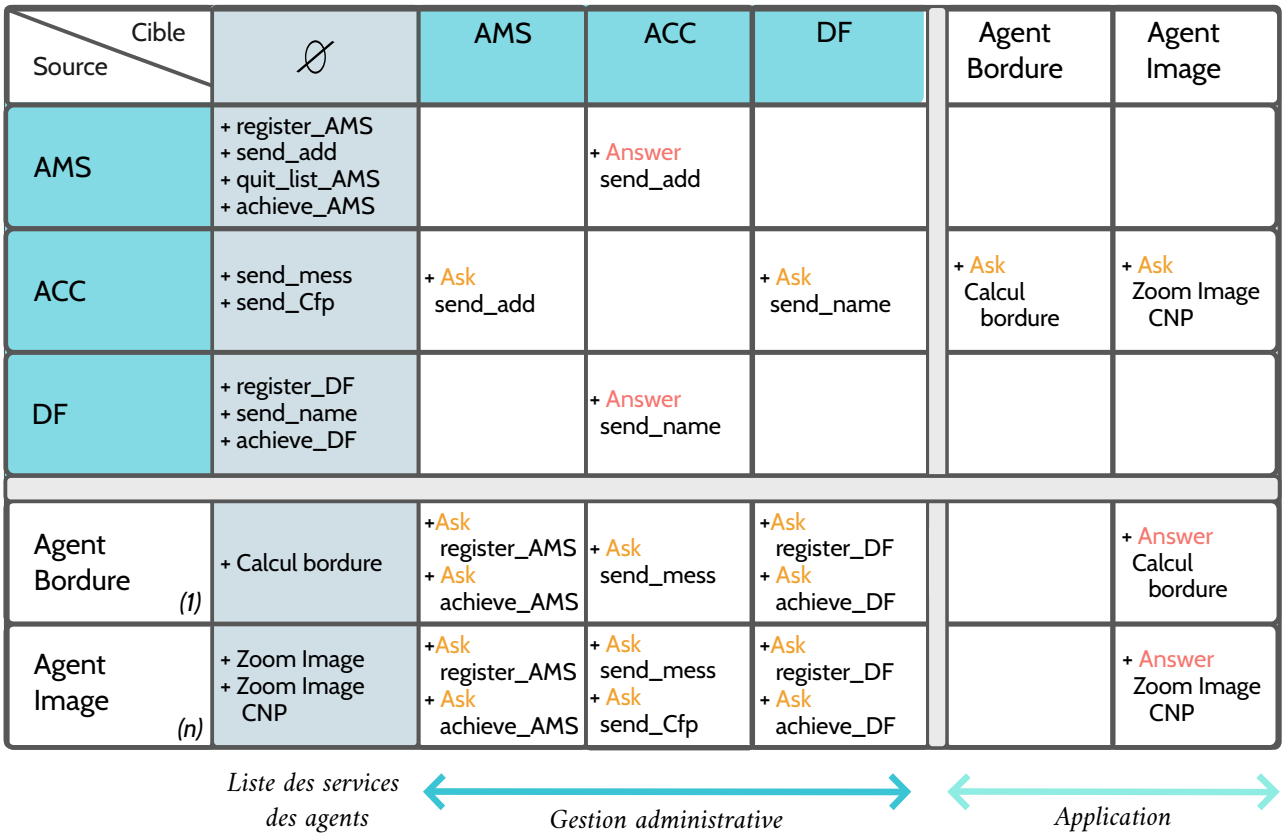


FIGURE 7.4 – *Proposition de la matrice complète des interactions pour le cas de délégation de tâches.*

choisi de nous concentrer sur la notion d'état de nos agents. Ainsi, les agents contactés par le Cfp vérifient leur état tel que présenté dans leur cycle de vie en section 4.2. Sont-ils déjà occupés à réaliser une tâche (état Actif du service) ? Si la réponse est oui, ils renvoient à l'agent initiateur une proposition négative indiquant leur incapacité à répondre au besoin de ce dernier. Si la réponse est non, ils sont donc disponibles (état Prêt) et renvoient une proposition positive. Notre agent initiateur considère la première proposition qu'il reçoit comme étant la meilleure. Il renvoie donc un message de confirmation à l'agent lui ayant répondu le plus rapidement. Pour tous les autres agents ayant renvoyés une proposition positive, l'initiateur la refuse.

Enfin, l'agent dont la proposition a été acceptée vérifie qu'il n'a pas changé d'état au cours de la négociation. S'il est toujours disponible, il confirme sa proposition. L'agent initiateur clôt le protocole et lui délègue la tâche.

7.2.2 Protocole d'expérimentation

Notre objectif est d'évaluer spécifiquement le surplus d'activités induit par les interactions issues du protocole de négociation. Un élément de mesure consiste à prendre en compte le temps de réponse de notre protocole dans le cas le plus gourmand : tous les agents générés sont disponibles et émettent chacun une proposition. Cependant une seule de ces propositions sera acceptée par l'agent initiateur, et donc un seul agent contacté ira jusqu'au bout du protocole de négociation. Nous établissons notre expérimentation pour un nombre d'agents, proposant le même service de zoom sur une image, variant de 2 à 100, soit de 6 à 300 messages générés.

Le temps de réponse induit par la négociation relative à ce nombre d'agents est relevé. Nous considérons les résultats avec la fenêtre de temps du protocole appliqué à la mesure du temps de réponse de notre expérimentation sur la flexibilité [Section 6.6]. Nous retrouvons ainsi la limite de réponse dynamique d'affichage d'une image de 25 ms pour la perception humaine et la limite de cadence d'une vidéo de 55 ms. Pour déterminer le temps de réponse du programme, nous déclenchons un décompte lors de la réception par un service CNP d'un agent d'une tâche jusqu'à l'obtention d'une solution avec un autre agent.

Pour déclencher le protocole de négociation, nous lançons notre expérimentation en contactant continuellement le même agent pour lui demander d'effectuer le service d'agrandissement sur une image donnée. La première fois, l'agent étant disponible, il accepte la demande et commence son travail. Ce service se termine à l'initiative de l'utilisateur. Nous laissons volontairement le service en attente pour provoquer une situation continue d'indisponibilité chez notre agent. Nous lui envoyons alors une nouvelle demande de traitement pour un agrandissement d'une autre image. Cette fois l'agent est occupé à traiter notre première demande, il déclenche alors le protocole de négociation pour déléguer notre service.

7.2.3 Résultats

Pour observer le fonctionnement de la délégation, ce dernier ne relevant d'aucune donnée quantitative, nous avons mis à jour l'affichage de l'image traitée pour indiquer l'identifiant de l'agent Image ayant pris en charge le service. Nous avons ainsi pu constater que si la première demande était bien traitée par l'agent contacté alors disponible, la demande suivante était traitée par un autre agent que l'utilisateur n'avait pas lui-même contacté. Ce changement d'agent responsable de la tâche permet d'attester du bon fonctionnement du principe de délégation. Par ailleurs, la reproductibilité de notre protocole pour chaque expérience nous permet de constater que l'agent héritant de la tâche à traiter n'est pas toujours le même, attestant ainsi d'une solution, issue des interactions entre les agents, non déterministe. Cette observation démontre l'indépendance de nos agents.

En termes de résultats quantitatifs, la Figure 7.5 présente les temps de réponse relevés au cours des négociations. Ces valeurs représentent une moyenne des résultats obtenus lors des 5 expérimentations effectuées pour chaque nombre d'agents Image impliqués.

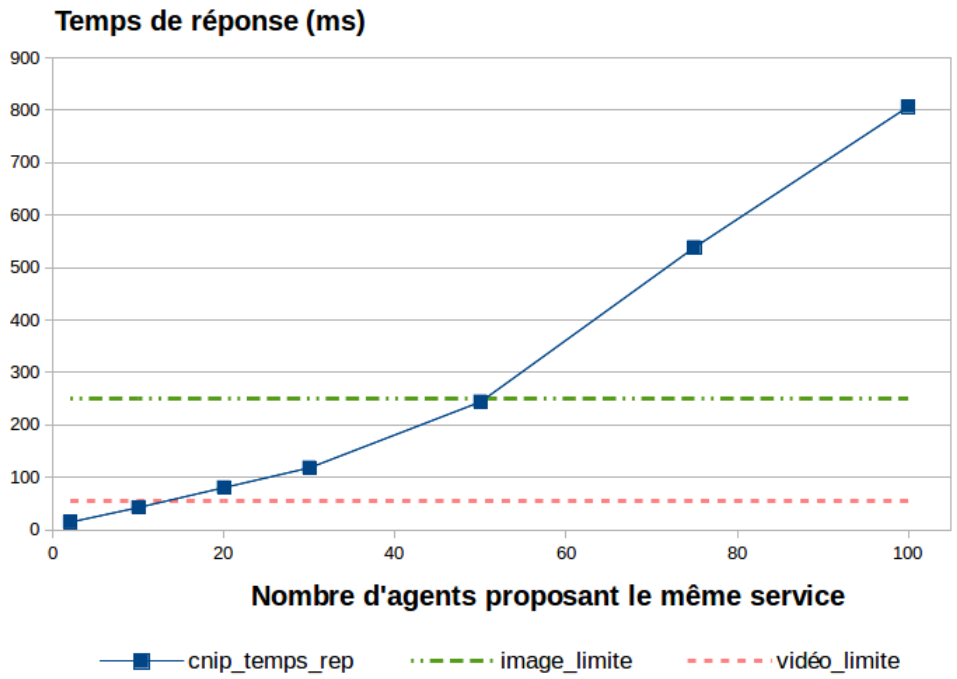


Figure 7.5 – *Temps de réponse de la résolution du protocole de négociation CNP en fonction du nombre d'agents impliqués.*

Nous relevons également les pourcentages du CPU et de la mémoire consommés durant la résolution de ces négociations. De la même façon que pour le temps de réponse, ces mesures se font en fonction du nombre d'agents déployés. Les résultats pour le CPU augmentent linéairement de 0,15 % de sa capacité pour 2 agents à 5,4 % pour 100 agents. Nous observons le même accroissement pour la mémoire, montant elle jusqu'à une consommation maximale de 0,3 %. Par ces résultats, nous constatons que les coûts administratifs en termes de consommation mémoire et CPU sont très faibles par rapport aux capacités maximales des composants. Il n'est donc pas nécessaire de chercher

à réduire davantage ses coûts.

Nous rappelons que dans nos cas d'expérimentations, le nombre d'agents proposant le même service correspond à autant de traitements pouvant être effectués en parallèle. Dans le cadre d'un traitement non temps réel, doté d'une attente passive, la consommation CPU est ponctuelle pour la négociation. Dans le cadre d'un traitement temps-réel, comme par exemple la lecture d'une vidéo, les traitements parallèles accumulent la consommation CPU à laquelle le protocole de négociation s'ajoute encore, ce qui peut engorger considérablement l'ensemble du système. Nous devons donc considérer cette accumulation pour le choix du nombre d'agents à déployer.

Nous associons cette première analyse au temps de réponse des négociations. Il est important de noter que ce temps de réponse n'inclut pas le traitement. Il faut donc également considérer ce dernier pour estimer le nombre d'agents permettant le respect des limites du domaine. Pour garder une marge vis-à-vis de ce temps de traitement, nous diminuerons notre nombre d'agents considérés de 2.

Nous constatons ainsi que nous pouvons tout-à-fait tenir la limite de 250 ms de traitement pour une réactivité fluide à l'œil en déployant jusqu'à 48 agents sur notre plateforme. Notre perspective de délégation de tâches peut donc entraîner un gain énorme en termes de nombre de traitements effectués simultanément. Au niveau du traitement temps-réel, nous restons conformes au seuil minimal de 18 images par seconde nécessaire à la fluidité d'un rendu vidéo en déployant jusqu'à 8 agents.

Dans tous les cas comme nous l'avions évoqué lors de l'analyse des résultats de notre précédente expérimentation en 6.6.1, le coût administratif en termes de temps de réponse représente un impact pour le choix d'une approche multi-agents. Il faut donc déterminer le nombre d'agents à générer en fonction de la rentabilité de ces derniers, c'est-à-dire l'estimation de leur contribution par rapport à leur coût.

L'expérimentation du protocole de négociation CNP permet une délégation de tâches flexible et dynamique qui amène bien un gain en nombre de services traités, tout en respectant un coût administratif acceptable pour notre SE.

Nous souhaitons à présent améliorer notre premier cas d'expérimentation en ajoutant cette capacité de délégation de tâches à notre cas d'étude relié à la flexibilité du SMA.

7.3 | Temps de réponse pour plusieurs images avec délégation des tâches.

Avec cette seconde expérimentation, nous évaluons le temps de réponse du traitement complet du SMA embarqué. Nous avons vu en section 6.6.1 que la performance du système pouvait représenter un coup administratif important. Par conséquent, nous avions conclu que l'application de solutions multi-agents devait chercher un équilibre entre un résultat avantageux et ce coût administratif. Avec les résultats de performance du protocole CNP, nous avons établi une limite haute de 10 agents déployés au delà de laquelle le temps de réponse de la négociation seule représente un coût trop élevé. Cette limite nous permet de rester dans un cadre d'expérimentation pertinent pour évaluer le temps de réponse du traitement global.

Par ailleurs, nous souhaitons cette fois faire ressortir l'intérêt de paralléliser les traitements grâce à notre délégation des tâches. Nous ajouterons donc la variable du nombre d'images traitées à notre protocole d'expérimentation.

Le SMA embarqué appliqué pour cette expérimentation correspond à celui présenté en section 6.4 avec l'ajout de la capacité de l'agent Image à déléguer des tâches présentée en section 7.2.

Nous déployons de 2 à 10 agents Image sur notre système et mesurons le temps de traitement de la totalité des images. Nous expérimentons d'abord notre SMA pour 2 images, puis 5 et enfin 10 afin d'atteindre le ratio d'une image à traiter par agent. Les images à traiter correspondent toutes à la même : l'image standard Lena, de dimension 512x512 et d'encodage PNG, utilisée pour nos autres expérimentations. Les requêtes comportant le traitement des images sont toutes envoyées au même agent. Ce dernier délègue son travail en cas d'indisponibilité. Si tous les agents Image sont indisponibles, la tâche est mise en attente. La Figure 7.6 présente les résultats obtenus.

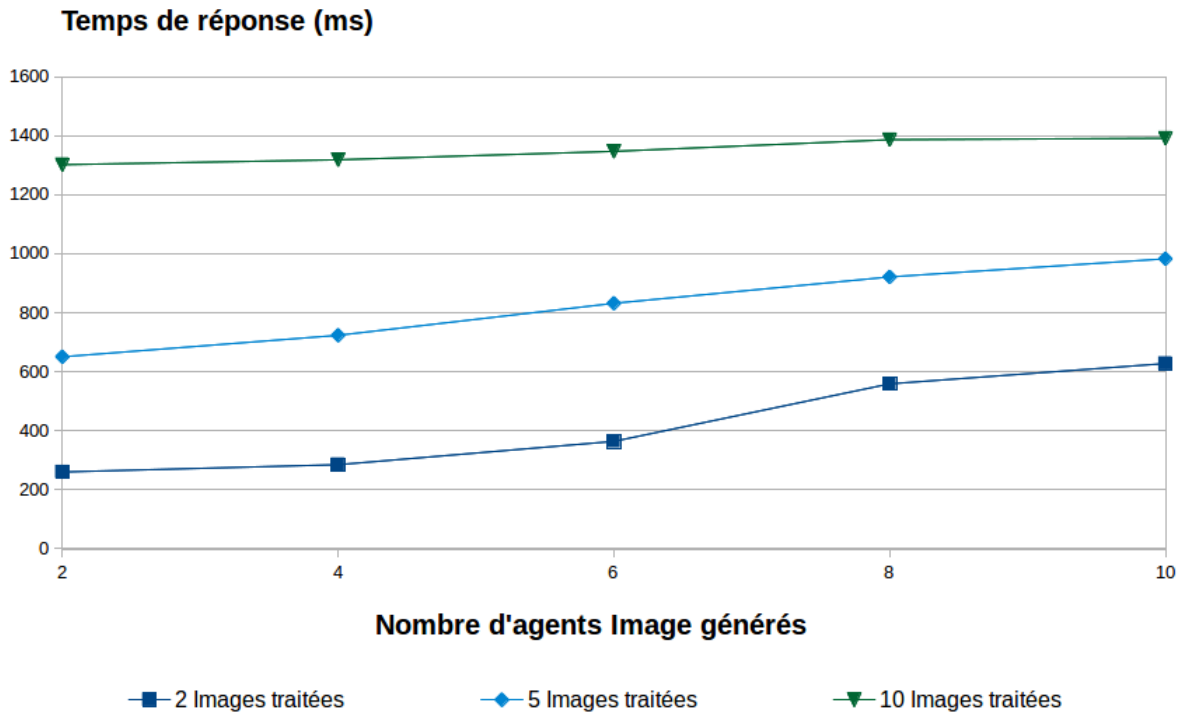


Figure 7.6 – *Mesure du temps de réponse pour la réalisation d'un zoom sur une image selon une approche multi-agents comparée au temps de réponse du même travail selon une approche séquentielle.*

Pour nos deux images à traiter, le nombre d'agents disponibles augmente le temps de réponse avec un impact du nombre de messages pour le protocole de négociation. En augmentant le nombre d'images à 5, nous constatons un adoucissement de notre courbe avec l'augmentation du nombre d'agents. En effet, là où un faible nombre d'agents se retrouve en surcharge de travail, l'augmentation de leur nombre permet de paralléliser le traitement. Nous ressentons cependant toujours l'impact du temps de négociation dans notre temps de réponse total. Avec 10 images, nous obtenons une courbe faiblement croissante. L'impact du temps de négociation est atténué par la répartition du travail entre les agents.

Nous constatons malgré tout que le nombre d'agents déployés ne permet pas de diminuer le temps de réponse du traitement global. Nous expliquons ce résultat par notre contexte de travail : tous nos agents évoluent actuellement sur la même ressource matérielle. Bien qu'ils permettent de paralléliser les traitements, le système d'exploitation ne fait pas de différence au niveau du nombre de cycles nécessaires pour réaliser les tâches. Cette limite de nos travaux ouvre des perspectives intéressantes concernant la liaison des agents à différentes ressources matérielles. Cette réflexion sera détaillée en chapitre 8.2. Nous nous intéressons à présent à l'amélioration, grâce à la délégation de tâches, de notre expérimentation de la flexibilité apportée par les systèmes multi-agents.

7.4 | Analyse de la délégation associée à la flexibilité

Dans le chapitre 6, nous avons présenté un cas d'application de traitement d'images correspondant à une action d'agrandissement sur une image. Nous avons présenté ce cas avec une approche multi-agents permettant de partager cette tâche entre plusieurs agents. Cette répartition du travail permettait ainsi un apport de flexibilité en rendant possible l'application de plusieurs méthodes d'interpolation sur une même image. Elle prenait également en compte le contexte d'exécution pour choisir la méthode à appliquer.

À travers les résultats qualitatifs obtenus, détaillés en sous-section 6.6.2, nous avons ainsi pu observer que moyennant une modification de leur seuil de décision, les agents en charge de l'interpolation pouvaient réaliser leur travail malgré un contexte matériel peu favorable correspondant à un CPU surchargé.

Dans la section 7.1, nous avons présenté la possibilité de déléguer des tâches entre agents proposant le même service. Dans cette section, nous appliquons à présent cette notion de délégation de tâches avec nos agents gérant l'interpolation. Dans le cadre de cette expérimentation, nous reprenons le protocole d'expérimentation appliqué pour l'évaluation qualitative de la flexibilité en sous-section 6.6.2. En cas de surcharge du CPU, les agents doivent pouvoir initier une négociation pour déléguer leur travail.

En plus d'optimiser notre répartition du travail, notre hypothèse est de renforcer la fiabilité de notre système embarqué, telle que décrite en sous-section 2.1.1, en considérant nos agents comme des unités redondantes. En effet, ils sont capables de transférer un travail en cas de dysfonctionnement matériel afin que le service soit achevé malgré tout.

7.4.1 Formalisation des agents embarqués et application à MERMAID

Pour cette expérience, nous améliorons le SMA de l'expérimentation présentée en section 7.3 en y intégrant les services de négociation CNP au niveau des agents de la famille Pixel.

Au niveau des étapes de réalisation, nous reformons une nouvelle table des besoins en Figure 7.7. Nous y retrouvons notre nouveau besoin d'interaction pour les agents Pixel.

Familles d'agents	Besoins	Moyens
Image	Calcul des bordures du zoom	Bordure
	Remplissage de la matrice zoomée	Zoom
	Disponibilité des autres agents Image	**Image**
Bordure	*Agent indépendant*	
Zoom	Remplissage de la sous-matrice zoomée	Pixel
Pixel	Niveau de disponibilité du CPU	Contexte matériel
	Obtention de la valeur d'un pixel voisin	Pixel
	Disponibilité des autres agents Image	**Pixel**

Figure 7.7 – *Table des besoins pour un zoom réalisé sur une image par plusieurs agents Pixel capables de négocier une délégation de tâches.*

La matrice des interactions est mise à jour avec ce nouveau service pour l'agent Pixel déjà traité dans le cadre de la délégation entre les agents Image. Aussi, nous représentons directement la matrice complète des interactions en Figure 7.8.

Dans le cadre d'une délégation de tâches entre agents Pixel, nous établissons cette fois le déclenchement d'une négociation par protocole CNP sur la capacité du CPU sur lequel l'agent évolue. A l'instar de notre étude sur la flexibilité, nous considérons des états de CPU surchargés (occupés à plus de 60 %). En introduisant la possibilité de déléguer des tâches dans ce cas d'étude, nous proposons de ne plus diminuer la qualité du zoom en cas de CPU surchargé, mais plutôt de déléguer la tâche à un autre agent Pixel. L'état considéré pour le déclenchement de la délégation n'est donc plus "indisponible" mais "surchargé". Le principe de déroulement du protocole CNP reste le même : les agents contactés par le Cfp ayant l'état "surchargé" feront une proposition négative. Les autres répondront par une proposition favorable et l'agent initiateur sélectionnera la première proposition positive reçue.

Pour cette expérience, il est nécessaire de transférer un nombre important de données lors de la délégation de la tâche : la matrice de pixels assignée à un agent. Pour ne pas surcharger les copies mémoires, le transfert entre les services "Remplir matrice" et "Remplir matrice CNP" d'un même agent se font par des pointeurs.

Source \ Cible	∅	AMS	ACC	DF	Agent Image	Agent Bordure	Agent Zoom	Agent Pixel
AMS	+ register_AMS + send_add + quit_list_AMS + achieve_AMS		+ Answer send_add					
ACC	+ send_mess + send_Cfp	+ Ask send_add		+ Ask send_name	+ Ask Zoom Image CNP	+ Ask Calcul bordure	+ Ask Zoom matrice	+ Ask Remplir matrice + Ask Remplir matrice CNP
DF	+ register_DF + send_name + achieve_DF		+ Answer send_name					
Agent Image (n)	+ Zoom Image + Zoom Image CNP	+Ask register_AMS + Ask achieve_AMS	+ Ask send_mess + Ask send_Cfp	+Ask register_DF + Ask achieve_DF	+ Answer Zoom Image CNP			
Agent Bordure (1)	+ Calcul bordure	+Ask register_AMS + Ask achieve_AMS		+Ask register_DF + Ask achieve_DF	+ Answer Calcul bordure			
Agent Zoom (1)	+Zoom matrice	+Ask register_AMS + Ask achieve_AMS	+ Ask send_mess	+Ask register_DF + Ask achieve_DF	+ Answer Zoom matrice			
Agent Pixel (n)	+ Remplir matrice + Remplir matrice CNP + Pixel	+Ask register_AMS + Ask achieve_AMS	+ Ask send_mess + Ask send_Cfp	+Ask register_DF + Ask achieve_DF			+ Answer Remplir matrice	+ Ask Pixel + Answer Pixel + Answer Remplir matrice CNP

Liste des services des agents — *Gestion administrative* — *Application*

FIGURE 7.8 – *Matrice des interactions complète pour un zoom avec possibilité de déléguer la tâche d'interpolation gérée par les agents Pixel.*

Nous souhaitons à terme pouvoir faire communiquer nos agents Pixel sur différentes cartes embarquées et appliquer la délégation entre différents systèmes. Pour cela, nous avons choisi de transférer les matrices de données par socket.

7.4.2 Analyse de la délégation

7.4.2.1 Protocole expérimental

Nous reprenons le protocole de la sous-section 6.6.2. Lors d'un contexte de CPU indiqué comme utilisé à plus de 60 %, nous activons le protocole CNP de délégation de la tâche.

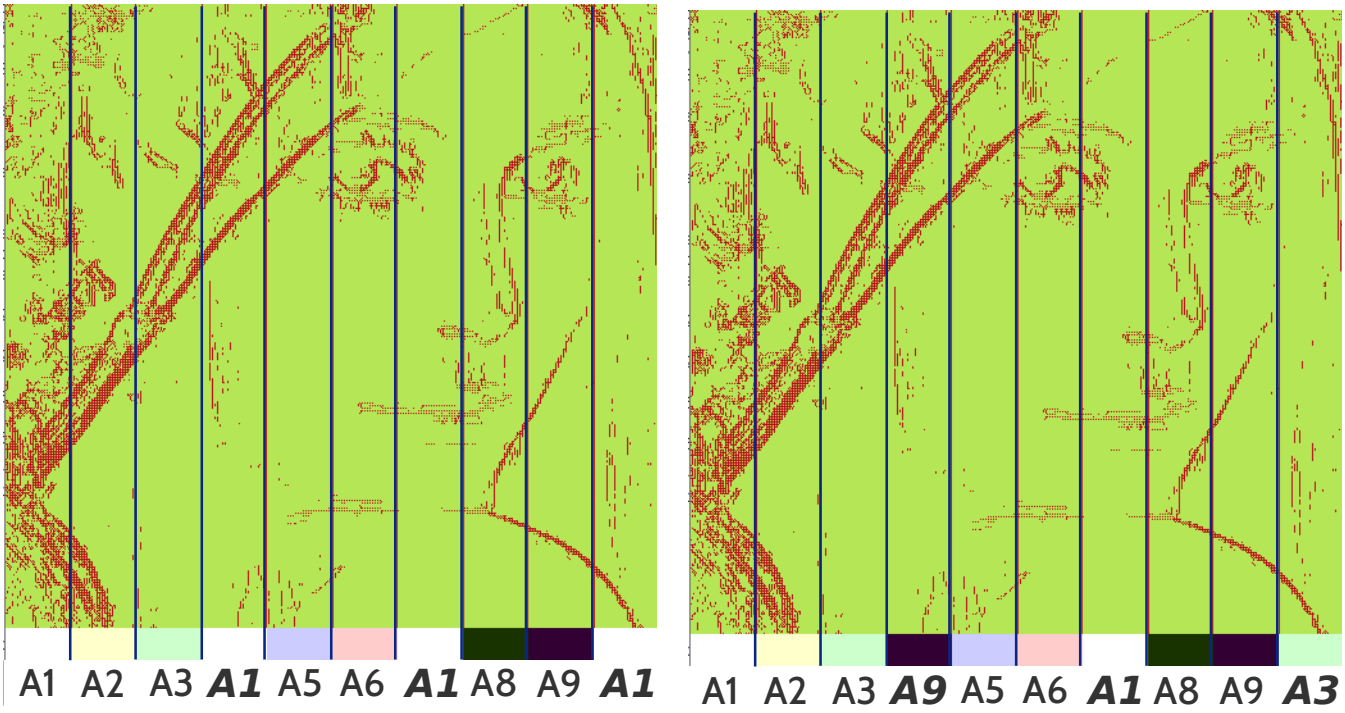
Pour déterminer visuellement si la tâche a été traitée par un autre agent que celui de son affectation d'origine, nous mettons en place une "signature". Elle correspond à une bande de pixels de couleur. Cette bande remplace le résultat de l'interpolation sur une surface correspondant à 10 % de la hauteur de la matrice et 100 % de sa largeur. Elle est placée tout en bas de l'image.

Nous créons un code couleur et associons une couleur à un numéro. Ainsi, la première instanciation d'un agent Pixel se verra attribuer la signature de couleur n°1, le second agent Pixel aura la signature de couleur n°2 et ainsi de suite. De cette manière, lorsqu'un agent Pixel en vient à déléguer une tâche, nous pouvons à la fois vérifier que la signature correspond à un autre agent Pixel et identifier lequel. En effet, nous ne contrôlons pas l'issue du protocole de négociation. Il est donc intéressant de pouvoir observer visuellement quel agent a obtenu le traitement de la tâche.

Nous effectuons deux évaluations avec dix agents Pixel dont trois possédant un contexte surchargé pour observer le phénomène de délégation de tâches au sein de notre zoom. L'image utilisée pour notre expérimentation est le standard "Lena" et est présentée en sous-section 6.6.2 pour l'analyse de la flexibilité.

7.4.2.2 Résultats

Nous effectuons la réalisation d'un agrandissement partagé entre dix agents Pixel. Nous indiquons aux agents Pixel n°4, 7 et 10 que leur CPU est surchargé à plus de 60 % afin de déclencher un protocole de négociation. La Figure 7.9 présente nos résultats pour cette expérience.



zoom par 10 agents Pixel
(Agent n°4, 7 et 10 avec CPU à plus de 60 %)
délégation à l'agent n°1 à chaque fois

zoom par 10 agents Pixel
(Agents n°4, 7 et 10 avec CPU à plus de 60 %)
délégation respectivement aux agents n°9, 1 et 3

FIGURE 7.9 – *Plusieurs agents Pixel partagent le calcul d'interpolation d'un zoom sur l'image standard "Lena". Certains possèdent un contexte d'exécution défavorable et délèguent donc leur travail à d'autres agents Pixel.*

Dans les deux cas, nous pouvons observer la délégation grâce aux signatures. Par exemple pour le premier résultat, au niveau de l'agent Pixel n°4, la signature qui apparaît est celle de l'agent Pixel n°1. Nous pouvons ainsi déduire que face à son contexte défavorable, l'agent Pixel n°4 a bien démarré un protocole de négociation CNP. Il a abouti à un accord avec l'agent Pixel n°1 et c'est ce dernier qui s'est chargé du calcul d'interpolation.

Nous pouvons observer le même résultat pour les agents en contexte surchargé n°7 et 10. Sur l'image de gauche, toutes les négociations ont abouti avec un accord avec l'agent Pixel n°1. Sur l'image de droite, les résultats sont plus variés : le travail alloué à l'agent Pixel n°4 est récupéré par l'agent Pixel n°9, le travail donné à l'agent Pixel n°7 est délégué à l'agent Pixel n°1 et la tâche attribuée à l'agent Pixel n°10 est confiée à l'agent Pixel n°3.

Lors de nos évaluations, nous obtenions chaque fois un résultat différent. En effet, nous avons établi la rapidité comme critère de décision pour notre protocole de négociation. Cependant, le système d'exploitation décide à quel processus ou thread il alloue la ressource processeur à chaque cycle d'horloge du SE. Ce choix influe sur la rapidité de nos agents : ils reçoivent leur requête à différents cycles d'horloge. Par ailleurs, cette décision du système d'exploitation n'est pas reproductible car elle dépend également des autres processus tournant sur le système en tâche de fond, c'est-à-dire sans que l'on puisse influer sur leur résolution.

Nous validons donc l'apport de délégation au sein d'un même système embarqué. Il est à présent intéressant d'expérimenter ce principe entre différentes cartes embarquées ; d'une part pour étendre la fiabilité et l'optimisation de notre système à un ensemble de système embarqués, d'autre part pour observer la délégation entre différentes cartes embarquées et vérifier si un contexte hétérogène apporte l'émergence de nouveaux comportements.

7.5 | Délégation entre différentes cartes embarquées

Cette expérimentation représente une extension de la précédente à un contexte de plusieurs cartes embarquées. Au niveau de l'application de MERMAID, cela signifie un changement de système de communication dans le cadre des interactions entre les différentes cartes : elles passeront par le réseau de socket au lieu de D-Bus. La sous-section 5.2 a présenté l'organisation des plateformes MERMAID sur différentes cartes embarquées. Pour rappel, nous y montrons qu'une seule plateforme embarquée doit intégrer le bloc d'administration de MERMAID, les autres SE supportant simplement son noyau pour permettre aux agents d'interagir entre eux.

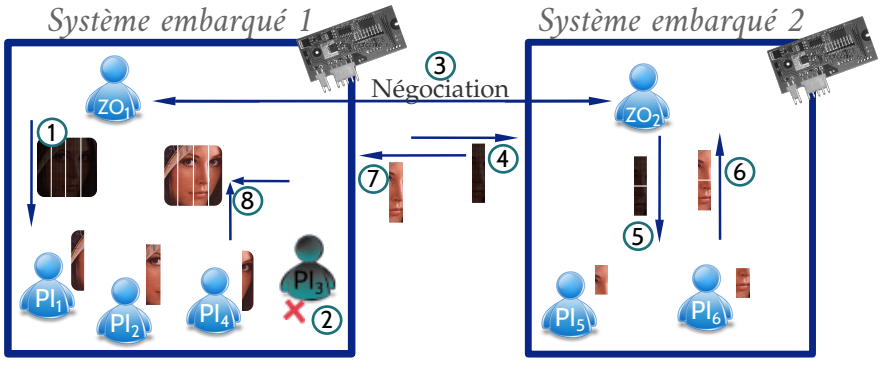
Le contexte applicatif utilisé pour cette expérimentation est le même que pour l'étude précédente en section 7.4, c'est-à-dire le SMA gérant des agrandissements d'image. Nous ré-expérimentons donc les familles d'agents Image, Bordure et Zoom qui gèrent l'image traitée et la sélection de la matrice de pixels à agrandir, ainsi que la famille Pixel regroupant les agent capables d'effectuer les calculs d'interpolation.

Pour distribuer les agents applicatifs de ces différentes familles sur les diverses cartes embarquées nous disposions de plusieurs choix de répartition possibles.

7.5.1 Choix d'organisation des agents au sein du SMA

L'extension de notre délégation de tâches à plusieurs systèmes embarqués nous a amenés à plusieurs choix de répartition de nos agents.

Le premier choix correspond à un agent de la famille Zoom sur chaque SE. Chacun de ces agents Zoom interagit avec les agents de la famille Pixel évoluant dans le même environnement immédiat. Nos agents Pixel traitent les sous-parties d'une matrice gérée par l'agent Zoom et chaque carte embarquée gère donc une image spécifique. La délégation d'une tâche a alors lieu entre les agents Zoom. La Figure 7.10 illustre cette première solution en détaillant un exemple d'application.

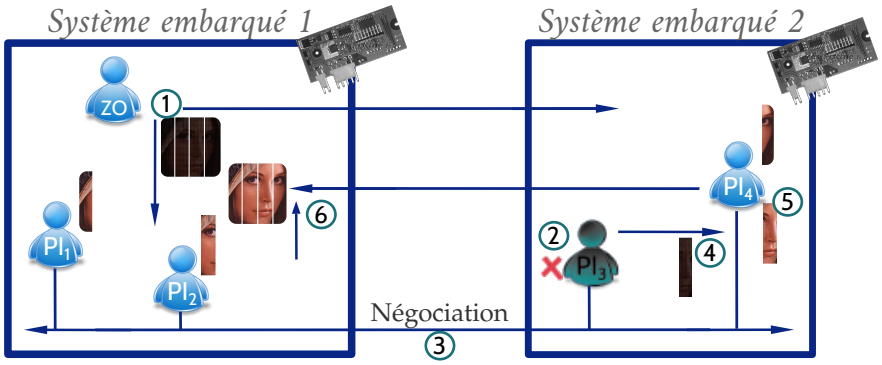


***(1)** : l'agent Zoom ZO_1 du SE n°1 découpe une matrice de pixels en quatre sous-matrices pour les quatre agents Pixel de son environnement immédiat. En **(2)**, l'agent Pixel PI_3 constate que son contexte d'exécution est défavorable à l'application de son service et indique donc à l'agent ZO_1 son incapacité à satisfaire sa requête. L'agent ZO_1 déclenche alors le protocole de négociation CNP pour déléguer le travail relatif à la sous-matrice précédemment confiée à l'agent PI_3. En **(3)** il négocie ainsi avec l'agent ZO_2 évoluant sur le système embarqué n°2 dans l'environnement hétérogène de ZO_1. Les deux agents parviennent à un accord permettant à l'agent ZO_1 en étape **(4)** de déléguer la sous-matrice à traiter à ZO_2. Ce dernier la répartit entre les deux agents Pixel présents dans son environnement immédiat : PI_5 et PI_6 en étape **(5)**. Une fois le travail achevé, ces deux agents renvoient leur sous-matrice complétée à l'agent ZO_2 en **(6)**. Ce dernier, après avoir réuni les deux parties, renvoie à l'agent ZO_1 la matrice complétée en **(7)**. L'agent Zoom ZO_1 peut alors en étape **(8)** réunir l'ensemble des sous-matrices pour reformer une matrice zoomée complète.*

Figure 7.10 – *Première possibilité d'organisation pour une délégation de tâches entre différentes cartes embarquées dans le cadre d'un agrandissement d'images : Négociation entre les agents Zoom.*

L'avantage de cette première solution est de limiter les communications entre systèmes embarqués. Elle présente cependant des limites en forçant la délégation à remonter par nos agents Zoom quel que soit le nombre d'agents Pixel souhaitant déléguer. Nous notons ainsi un risque de goulot d'étranglement. Par ailleurs, cette solution implique de disposer des agents Zoom et Pixel sur chaque système embarqué. Or, nous avons vu à travers les résultats de nos expérimentations quantitatives sur les temps de réponse en sous-section 6.6.1 et 7.2 qu'un trop grand nombre d'agents induit un coût administratif important réduisant l'apport du SMA. Il peut donc être plus pertinent de limiter le nombre d'agents utilisés en fonction des besoins applicatifs directs. Ces limites nous ont conduit à considérer une deuxième option d'organisation.

Ce deuxième choix implique de continuer de considérer le SMA dans son ensemble et de laisser la couche MTPS, présentée en sous-section 3.2.4 et appliquée à MERMAID en sous-section 5.1.1, assurer le routage en fonction de la localisation des agents Pixel. A ce titre, nous avons un seul agent Zoom pour tout le SMA. Ce dernier peut interagir avec les agents Pixel avec un routage assuré de façon transparente pour lui par ACC et la couche MTPS. La délégation de tâches se déroule ici directement entre les agents Pixel dans le cadre de leur environnement hétérogène. Cette solution est illustrée par un exemple en Figure 7.11.



*ZO répartit en étape **(1)** sa matrice de pixels incomplète en quatre sous-matrices pour chaque agent Pixel répertorié sur l'ensemble du SMA. En **(2)** l'agent Pixel PI_3 constate que son contexte d'exécution est défavorable à l'application de son service d'interpolation. Il déclenche alors un protocole de négociation avec tous les autres agents Pixel répertoriés en étape **(3)**. Il parvient à un accord avec l'agent PI_4 et lui délègue donc sa sous-matrice à compléter en **(4)**. PI_4 se charge ainsi de la complétion de la sous-matrice assignée par l'agent ZO et de la sous-matrice déléguée par l'agent PI_3 en **(5)**. Enfin lors de l'étape **(6)**, l'agent ZO récupère l'ensemble des sous-matrices achevant ainsi la recomposition d'une matrice zoomée complète.*

FIGURE 7.11 – *Deuxième solution d'organisation : Délégation de tâches directement entre les agents Pixel.*

L'avantage de cette solution est de valider le fonctionnement de notre couche MTPS. De plus, elle supprime le rôle d'intermédiaire des agents Zoom en permettant une délégation directe entre les agents Pixel. Cependant, elle soulève également des limites avec un plus grand nombre d'agents impliqués lors de la résolution du protocole CNP et donc un plus grand nombre de messages entre les systèmes embarqués.

Lors de l'étude du RTT de notre plateforme en section 5.3.1, nous avons étudié le temps de transfert d'un message en interne et entre deux systèmes. Nous avions pu constater que les transfert en externe ne représentaient qu'une faible augmentation du RTT. Nous pouvons alors considérer la limite de cette deuxième solution comme acceptable. Nous choisissons donc cette option.

Ainsi, dans le cadre de cette étude, nous plaçons l'ensemble de nos agents sur les différentes cartes tout en prenant soin de répartir les instances de la famille Pixel. Lors de l'initialisation d'un agent, nous lui spécifions l'adresse IP de la carte embarquée supportant les agents administratifs de MERMAID afin que chacun puisse s'enregistrer auprès des agents AMS et DF.

7.5.2 Protocole d'analyse de la délégation dans un contexte hétérogène

Ce protocole poursuit l'expérimentation précédente détaillée en sous-section 7.4.2 : certains agents Pixel sont informés d'un contexte peu favorable pour l'exécution de leur service : un CPU déjà utilisé à plus de 60 % de sa capacité maximale. Cette indication prise en compte, la réception d'une requête pour leur service déclenche le protocole de négociation CNP. A l'issue de ce protocole, ils peuvent déléguer leur travail à l'agent Pixel auprès duquel ils auront conclu un accord. Nous utilisons toujours le principe de signature pour déterminer quel agent Pixel a réalisé le travail délégué.

Pour le premier système embarqué enregistré par le système multi-agents, c'est-à-dire celui supportant la version administrative de MERMAID, les couleurs restent le vert pour la méthode d'interpolation PPV et le rouge pour la méthode BL. Pour le suivant, la méthode PPV est représentée par une couleur bleue et la méthode BL par une couleur rose. Pour le troisième système, une interpolation par plus proche voisin correspond à une couleur gris foncé et une interpolation bi-linéaire est représentée par du blanc.

Pour nos expérimentations, nous avons accès à deux cartes embarquées de la famille Cannes, décrites en section 6.1.2. Nous utilisons également un ordinateur Core i7 pour éprouver l'hétérogénéité de notre plateforme.

7.5.3 Expérimentation du contexte multi-cartes

Dans un premier temps, nous éprouvons l'hypothèse relative à MERMAID présentée en section 1.5 et rappelée ci-après. En effet, en répartissant notre SMA sur différentes cartes embarquées nous validons son hétérogénéité apportée par le respect de la norme FIPA et notamment de l'intégration de la couche MTPS.

Rappel hypothèse #2

Le suivi des standards d'administration et d'architecture multi-agents nous permettra de proposer de nouveaux modèles de communication pour une plateforme embarquée.

Nous effectuons ainsi plusieurs expériences avec différentes répartitions de nos agents et de notre bloc d'administration de MERMAID. Nous présentons en Figure 7.12 l'une de ces expériences impliquant trois systèmes et un total de onze agents.

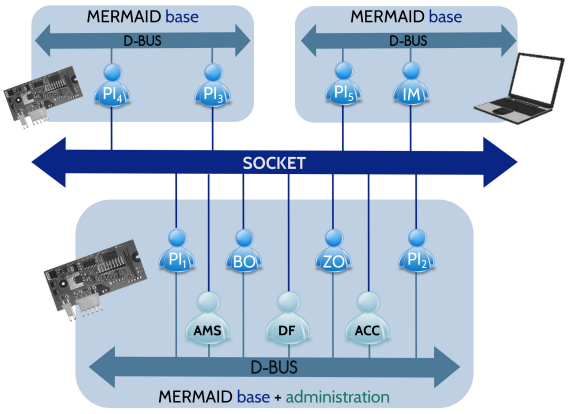
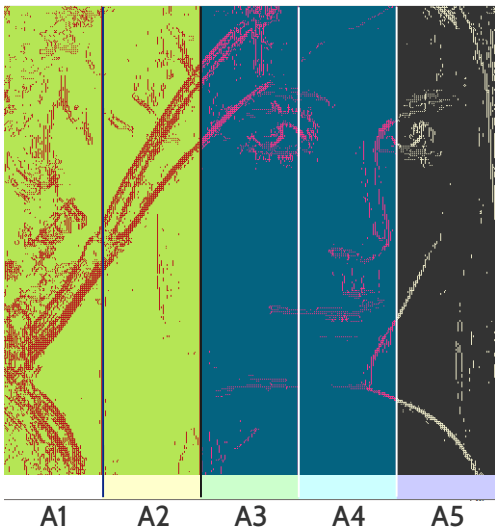


Illustration des choix de répartition du SMA



Visualisation de l'agrandissement effectué sur "Lena"

Figure 7.12 – *Expérimentation d'un agrandissement effectué sur l'image standard "Lena" et partagé entre plusieurs agents Pixel évoluant sur différents systèmes embarqués.*

Les deux premiers agents Pixel évoluent sur une première carte embarquée de la famille Cannes. Les deux suivants s'exécutent sur un deuxième système embarqué de la même famille. Enfin le cinquième agent évolue seul sur le PC portable. Le bloc d'administration de MERMAID est situé sur une carte embarquée.

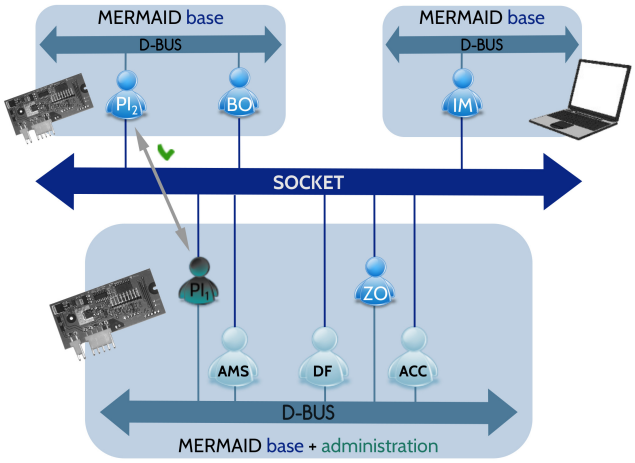
Nous expérimentons différentes répartitions des autres familles d'agents : Image, Bordure et Zoom. L'emploi de ces différents agents sur des contextes matériels distincts n'a pas entraîné de dysfonctionnement ou de modification du résultat. Nous constatons ainsi que notre plateforme MERMAID permet bien d'affranchir les agents applicatifs du contexte matériel sur lequel ils évoluent. Par ailleurs, ces derniers ont échangé des messages sans impact par rapport à un SMA évoluant sur un seul SE. Cette observation nous permet de valider le fonctionnement de notre couche

MTPS qui se charge de la sélection du protocole de communication de façon transparente pour les agents applicatifs.

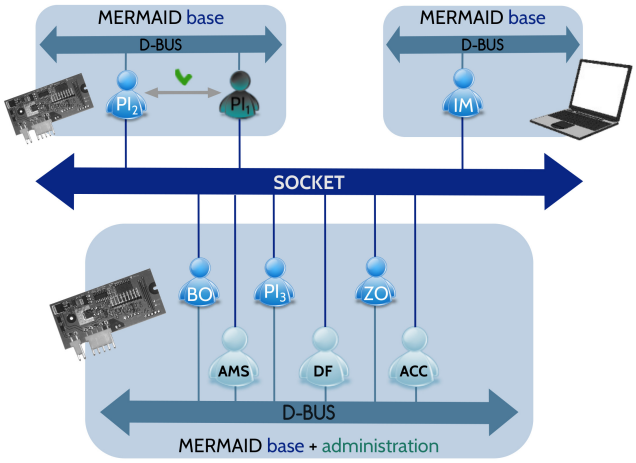
Nous validons ainsi notre hypothèse #2. Le passage à un environnement multi-cartes potentiellement hétérogène ouvre de nombreuses perspectives en permettant de mettre en contact différents systèmes embarqués par le biais des SMA. La possibilité de déléguer une tâche à un autre système embarqué en fait partie en augmentant le nombre de ressources à disposition d'un SE embarqué ainsi relié à d'autres cartes.

7.5.4 *Optimisation entre systèmes embarqués par délégation*

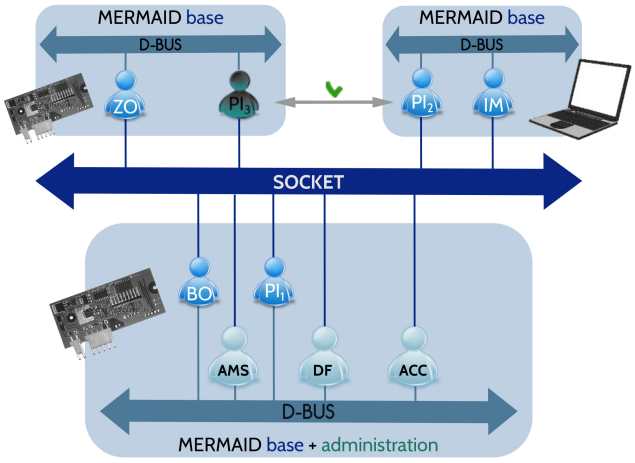
Nous éprouvons à présent notre hypothèse #3 rappelée en début de chapitre concernant l'optimisation des SE par la délégation de tâches entre nos différentes cartes embarquées. La Figure 7.13 présente les différentes répartitions retenues pour les expériences. Nous y soulignons également quel agent Pixel s'est vu attribuer un contexte d'exécution défavorable et avec quel autre agent Pixel il a trouvé un accord pour déléguer son travail. En Figure 7.14 nous présentons les résultats visuels de ces nouvelles expériences.



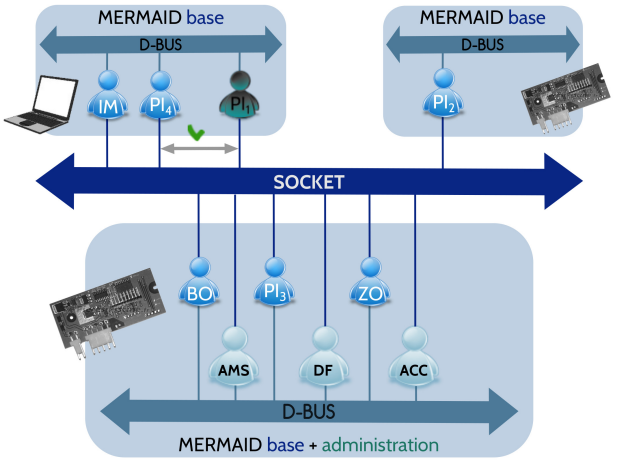
(a) Expérimentation avec 2 agents Pixel répartis sur deux cartes Cannes distinctes. La partie administrative est placée sur une carte Cannes



(b) Expérimentation avec 3 agents Pixel répartis sur deux cartes Cannes distinctes. La partie administrative est placée sur une carte Cannes



(c) Expérimentation avec 3 agents Pixel répartis sur deux cartes Cannes et un PC portable. La partie administrative est placée sur une carte Cannes



(d) Expérimentation avec 4 agents Pixel répartis sur deux cartes Cannes et un PC portable. La partie administrative est placée sur une carte Cannes

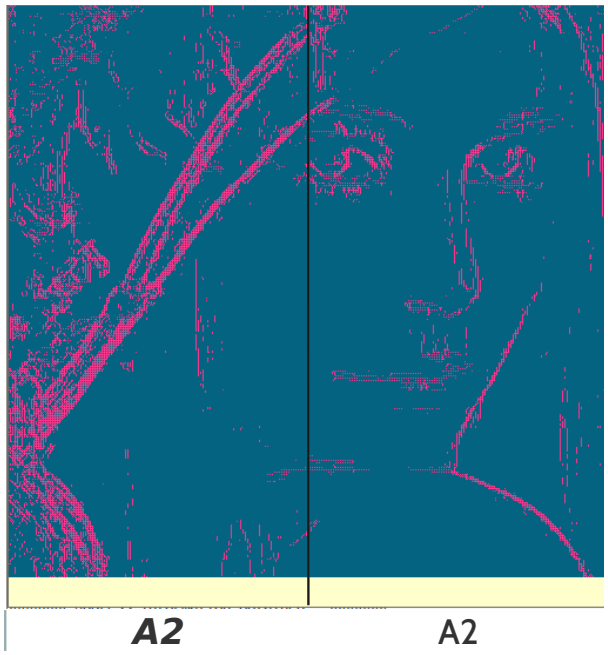
FIGURE 7.13 – *Représentation des différents choix d'architecture appliqués dans le cadre d'expérimentations impliquant de la délégation de tâches.*

En haut à gauche (a), nous avons initié deux agents Pixel sur deux systèmes embarqués différents. Le premier agent a pris en compte un contexte défavorable et a délégué son travail au deuxième agent. La délégation s'est donc réalisée dans l'environnement hétérogène. En haut à droite (b), nous avons trois agents Pixel, deux sur un premier système embarqué et un troisième sur une autre carte. Nous avons indiqué au premier agent que son contexte n'était pas favorable à une exécution. La délégation est effectuée au sein du même système embarqué, dans l'environnement immédiat de l'agent initiateur de la délégation. En bas à gauche (c), nous avons cette fois trois agents Pixel sur trois systèmes différents : le premier sur une carte Cannes, le deuxième sur notre ordinateur Core i7 et le troisième sur la deuxième carte Cannes. Nous avons indiqué au troisième agent Pixel que son CPU était déjà utilisé à plus de 60 %. Nous pouvons observer que ce dernier a alors délégué son travail à l'agent n°2 évoluant sur l'ordinateur. En bas à droite (d), nous répétons l'expérience précédente avec un agent sur ordinateur supplémentaire. Nous surchargeons le CPU du premier agent Pixel évoluant sur l'ordinateur. La négociation parvient à un accord avec l'autre agent évoluant sur ordinateur pour déléguer le travail.

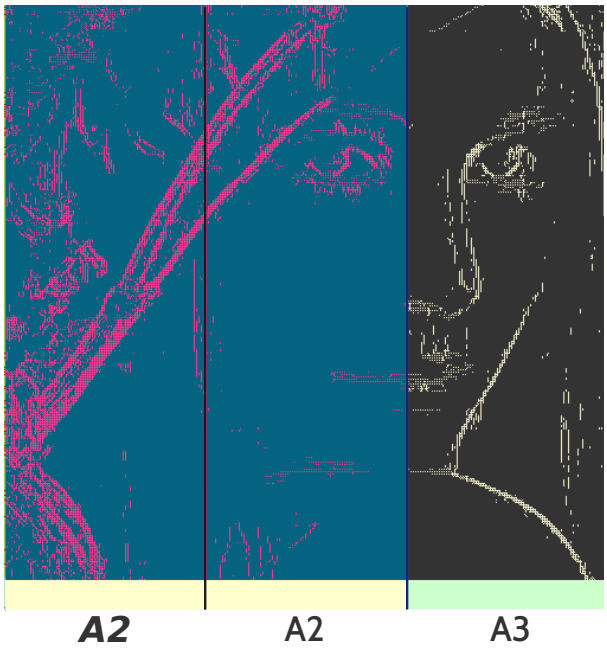
A partir de ces résultats, nous pouvons constater que la délégation est d'abord réalisée en interne lorsque c'est possible et ensuite en externe. Ceci n'est pas dû à une notion de choix spécifique de nos agents pour lesquels la couche MTPS fait abstraction du système sur lequel évoluent les autres agents Pixel. En effet, le travail du routage est effectué par l'agent ACC.

Cependant notre critère de sélection est la rapidité de proposition, et à ce titre la mesure du RTT réalisée en interne et entre cartes embarquées a démontré qu'une communication interne était plus rapide [Section 5.3.1]. Lorsque nous avons ajouté notre ordinateur aux expérimentations pour fonctionner avec différents systèmes, nous avons pu constater que nos agents Pixel évoluant sur ce support étaient systématiquement choisis à l'issue du protocole de négociation. Le seul facteur différent étant le système déployé, nous pouvons en déduire que la différence de capacité des machines a influé sur le temps de communication et donc la rapidité d'un agent à émettre une proposition.

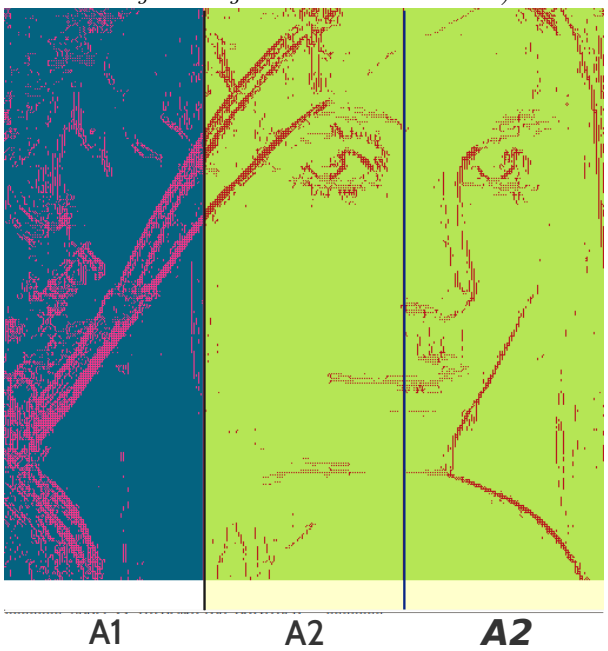
Nous constatons dans tous les cas que l'agent ACC contacte bien tous nos agents Pixel lors de la première étape du protocole de négociation. De plus, nous validons la possibilité pour le protocole de s'achever sur un accord entre deux agents Pixel issus de deux systèmes différents. Cette capacité de déléguer notre travail entre cartes embarquées renforce leur fiabilité. En effet, en cas de dysfonctionnement d'une ressource matérielle d'une carte, les agents peuvent déléguer la tâche à un autre SE possédant la même ressource. Cet aspect est également valable dans le cas d'une carte surchargée ou possédant des ressources trop limitées. Nous validons donc l'optimisation des différentes cartes embarquées par cette délégation effectuée par les systèmes multi-agents.



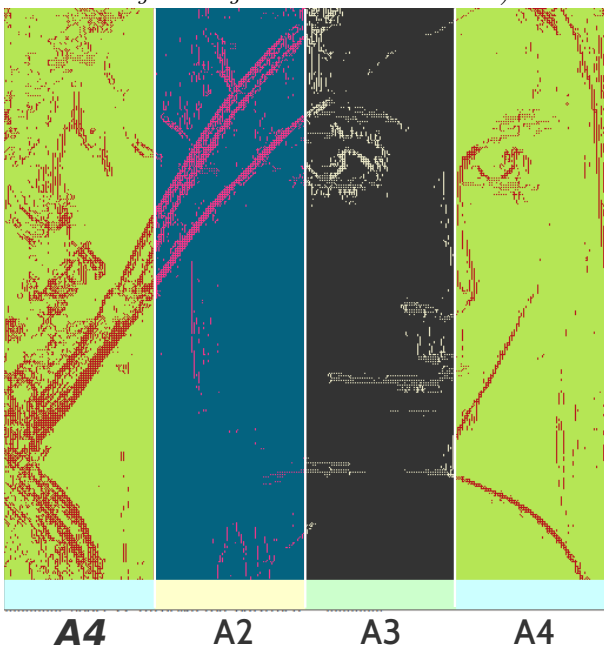
(a) Zoom par 2 agents Pixel (Agent n° 1 avec CPU à plus de 60 % délègue à l'agent n° 2 sur un autre SE)



(b) Zoom par 3 agents Pixel (Agent n° 1 avec CPU à plus de 60 % délègue à l'agent n° 2 sur le même SE)



(c) Zoom par 3 agents Pixel (Agent n° 3 avec CPU à plus de 60 % délègue à l'agent n° 2 sur un autre SE)



(d) Zoom par 4 agents Pixel (Agent n° 1 avec CPU à plus de 60 % délègue à l'agent n° 4 sur le même SE)

FIGURE 7.14 – *Différents exemples d'agrandissement de l'image standard "Lena". Les agents Pixel délèguent leur travail en cas de CPU utilisé à plus de 60 %.*

Synthèse

Au cours de cette partie, nous avons présenté nos expérimentations. Dans un premier chapitre, nous avons éprouvé notre hypothèse d'apport de flexibilité aux systèmes embarqués par les SMA à travers le cas d'étude d'un zoom sur une image.

Rappel hypothèse #4

La réalisation d'une tâche partagée entre plusieurs processus agents nous apportera de la flexibilité via différents choix d'exécution et l'adaptation au contexte d'exécution.

Nous y avons proposé un SMA embarqué selon les formalisations développées en Section 4.3. Composé notamment de plusieurs agents traitant le calcul d'interpolation du zoom, ce SMA a permis d'appliquer deux méthodes de calcul distinctes au sein d'une même image : la méthode du plus proche voisin et l'interpolation bi-linéaire. Ces deux traitements impliquent un coût différent en termes de nombre de cycles nécessaires à leur réalisation. Entre les deux, nous avons ainsi distingué la méthode bi-linéaire comme étant la plus coûteuse. Chaque agent considérant les informations liées aux pixels à traiter pouvait ainsi choisir d'employer la méthode la plus adaptée au cas par cas. Par ailleurs, nous avons souligné la flexibilité de nos agents en modifiant le contexte d'expérimentation pour certains d'entre eux. Nous avons ainsi indiqué pour quelques agents que la ressource matérielle sur laquelle ils évoluaient était utilisée à plus de 60 % de sa capacité maximale. Nous avons alors pu constater l'adaptation des agents concernés par ce changement de contexte : ils ont réduit l'utilisation de la méthode bi-linéaire considérée comme la plus coûteuse en pourcentage de consommation du CPU.

Nous avons ainsi pu valider notre hypothèse d'apport en flexibilité de nos agents embarqués.

Au cours de ce chapitre, nous avons également mesuré la performance de rapidité de traitement de notre solution multi-agents en la comparant avec un algorithme séquentiel réalisant également un zoom selon le même protocole. Si les SMA ont permis plus de flexibilité de résultat et de dynamisme concernant leur contexte, ils possèdent également un temps de réponse beaucoup plus élevé pour leurs services. Nous avons donc émis une réserve quant à leur utilisation dans le cadre de traitements temps-réel.

Avec notre second chapitre, nous avons expérimenté le principe de délégation de tâches au sein de nos systèmes embarqués pour éprouver notre hypothèse d'optimisation de ces derniers grâce aux algorithmes de négociation multi-agents.

Rappel hypothèse #3

Les processus pouvant négocier une délégation des tâches allouées, par le biais d'interactions multi-agents entre différents systèmes embarqués, optimiseront la gestion des tâches de chaque SE de façon réactive.

Nous avons tout d'abord analysé les performances du protocole de négociation choisi : CNP. Nous avons observé le temps de réponse impliqué par la résolution de ce protocole pour un nombre d'agents déployés variant de 2 à 100. Nous avons pu constater la forte croissance du temps de réponse et en avons conclu que le nombre d'agents déployés devait être évalué en fonction des objectifs de traitement visés.

Il était alors important de prendre en considération la capacité de parallélisation du traitement apportée par

notre délégation. Nous avons donc effectué une deuxième expérimentation considérant un nombre croissant d'images à traiter et relevant le temps de réponse pour l'ensemble de ces images. Nous avons pu observer qu'en déployant un nombre pertinent d'agents, en considération du nombre total d'images à traiter, le coût de communication du protocole était rentabilisé par l'évolution parallèle des services. Le niveau de performance restait limité par notre contexte d'expérimentation impliquant l'évolution de nos agents sur la même ressource matérielle : le CPU. Nos propositions relatives à cette limite sont détaillées dans les perspectives.

Nous souhaitions ensuite éprouver l'apport de notre délégation en améliorant notre expérimentation relative à la flexibilité. Nous avons donc repris le cas d'étude lié au traitement d'un zoom par un SMA embarqué ainsi que le protocole relatif à l'indication d'un CPU utilisé à plus de 60 % pour certains agents. Ces derniers déclenchaient alors une délégation de la tâche leur ayant été assignée. Nous avons ainsi pu observer d'autres agents n'expérimentant pas de contexte défavorable à recevoir les tâches déléguées. De ce fait, nous avons validé l'optimisation théorique de notre SE.

Pour valider concrètement notre hypothèse dans le cadre du déploiement de nos agents embarqués sur CPU, nous avons étendu notre expérimentation à une délégation entre plusieurs SE.

Nous avons tout d'abord appliqué notre traitement de zoom flexible, sans délégation, en répartissant notre SMA sur les différentes cartes. Cette étape nous a permis de valider le fonctionnement de notre couche MTPS. Cette dernière assure le transfert des messages à travers les différents systèmes de communication en toute transparence pour nos agents. Nous avons ainsi pu valider notre hypothèse relative à l'hétérogénéité de notre plateforme.

Rappel hypothèse #2

Le suivi des standards d'administration et d'architecture multi-agents nous permettra de proposer de nouveaux modèles de communication pour une plateforme embarquée.

Ensuite, nous avons activé la délégation des tâches en reprenant le protocole de contexte du CPU défavorable à un traitement. Nous avons expérimenté différentes répartitions de nos agents sur les différents systèmes. Pour cette expérience, les systèmes considérés étaient deux systèmes embarqués de la famille Cannes et un ordinateur Core i7. Nous avons pu constater une délégation effective entre les différents systèmes, validant ainsi une optimisation de ces derniers.

Bibliographie

[Amri et al., 2009] Amri, H., Khalfallah, A., and Bouhlel, M. S. (2009). Evaluation de techniques d'agrandissement des images. In *International Conference : Sciences of Electronic, Technologies of Information and Telecommunications (SETIT)*, pages 1–6. IEEE Computer Society Press.

[Chen and Thropp, 2007] Chen, J. Y. and Thropp, J. E. (2007). Review of low frame rate effects on human performance. *IEEE Transactions on systems, man and cybernetics—PART A : Systems and humans*, 37(6) :1063–1076.

Chapitre 8

Conclusions et Perspectives

"Réfléchissez constamment à la manière dont vous pourriez améliorer les choses et vous remettre en question."

Elon Musk
Entrepreneur, ingénieur, inventeur (1971 -)

Dans cette thèse, nous avons étudié l'application d'algorithmes multi-agents au sein des systèmes embarqués. Effectués dans le cadre d'un partenariat entre le laboratoire CREN et l'entreprise STMicroelectronics, nos travaux s'inscrivent dans l'action de recherche (AR) Tifaifai pour la conception de nouveaux espaces d'interactions adaptatifs, mobiles et connectés. Nous nous sommes ici focalisés sur l'expérimentation de systèmes multi-agents centrés interactions pour optimiser la gestion des ressources et des tâches au sein des cartes embarquées.

Nous concluons à présent nos travaux en proposant dans un premier temps un bilan de nos démarches avec nos hypothèses, les propositions formalisées et les expérimentations menées. Nous analysons nos résultats et détaillons les apports réalisés. Dans un deuxième temps, nous présentons les perspectives ouvertes par cette thèse.

8.1 | Apports de la thèse

8.1.1 *Synthèse globale des travaux*

A travers notre introduction, nous avons problématisé nos travaux concernant l'intégration d'algorithmes multi-agents au cœur des systèmes embarqués pour améliorer l'adaptation dynamique de ces derniers à leur contexte d'exécution. Nous avons alors présenté quatre hypothèses de recherche pour guider nos travaux [Section 1.5]. Nos deux premières hypothèses sont liées à la proposition de solutions pour l'intégration des SMA dans les systèmes embarqués. Nous focalisons ainsi notre approche sur les systèmes multi-agents centrés interaction pour favoriser la généricité de nos travaux. De plus, nous recherchons la formalisation d'une plateforme multi-agents embarquée normalisée pour assurer sa compatibilité avec d'autres projets multi-agents. Nous précisons également nos hypothèses liées aux apports pour les SE. Nous prévoyons ainsi l'optimisation de ces derniers par le biais d'une délégation des tâches réalisée grâce aux algorithmes de négociation multi-agents. Enfin, nous estimons que le partage d'une tâche entre plusieurs agents apportera plus de flexibilité à notre SE. Notre hypothèse vise également l'apport d'une capacité à réagir dynamiquement aux variations du contexte d'exécution.

Pour re-situer nos hypothèses, nous avons développé un état de l'art des deux domaines de notre thèse. En Chapitre 2, nous avons présenté les quatre défis principaux régissant les systèmes embarqués : leur fiabilité, leur sécurité, leur réactivité et leur optimisation. Nous avons particulièrement détaillé ce dernier point au centre de notre problématique. Nous avons ainsi précisé les spécificités des composants matériels des SE et de leur gestion pour soulever le problème à l'origine de ce défi : des ressources matérielles limitées. Nous avons alors résumé les mécanismes d'ordonnancement appliqués au cœur du matériel. Nous avons précisé une solution d'optimisation de nos ressources en agissant sur les tâches leur étant allouées : le principe de délégation des tâches. Nous avons ensuite exprimé le besoin

d'établir des interactions sociales entre les différentes ressources pour rendre cette délégation possible. Cette limite nous a amenés à présenter les différents systèmes de communication existant au sein des SE. Nous avons détaillé les notions de processus et de threads pour introduire les différentes méthodes de synchronisation nécessaires aux échanges d'information. Nous avons achevé cette partie par la présentation des protocoles de communication inter processus dont les socket et D-Bus que nous avons choisi d'expérimenter par la suite. Nous avons achevé ce chapitre par un rappel des différentes méthodes d'évaluation des travaux du domaine, notamment de la mesure du temps de réponse d'un traitement comme repère de performance.

Dans le Chapitre 3 nous avons introduit les notions de systèmes multi-agents par des définitions des principaux termes. Nous avons ainsi présenté les différents types d'agents, leur comportement, leurs modèles et leurs plateformes. Nous avons particulièrement détaillé deux modèles agents centrés interaction : DIAMOND et IODA. Le premier propose une approche complémentaire de nos travaux avec une méthode de co-design de systèmes embarqués et de SMA. Nos travaux se focalisent sur l'intégration d'un SMA à un SE existant. Le deuxième présente un modèle agent basant sa réalisation sur la spécification de ses interactions. Il établit notamment une matrice des interactions dont nous nous inspirons pour la formalisation de nos systèmes multi-agents embarqués.
Lors de notre présentation des différentes plateformes multi-agents existantes, nous avons constaté leur focalisation sur d'autres contextes de recherche. Nous avons profité de l'étude de ces plateformes existantes pour spécifier nos critères. Nous avons ainsi souligné l'importance de supporter un langage de programmation bas niveau pour optimiser notamment notre gestion de la mémoire. Nous avons également spécifié le besoin d'appliquer des modèles agents centrés interaction, de supporter des domaines d'application liés aux SE et de suivre des normes pour favoriser l'hétérogénéité de la plateforme.

Nous avons poursuivi avec la présentation des normes existantes : KQML et FIPA. La deuxième propose en plus de son standard de communication, FIPA-ACL, une normalisation de l'administration de la plateforme. Nous avons choisi d'employer cette option pour nos propositions et donc particulièrement détaillé les spécifications de cette norme. Nous avons notamment précisé les paramètres d'encapsulation des messages, le fonctionnement de la couche MTPS assurant leur transfert et les propriétés des agents administratifs AMS, DF et ACC. Nous avons ensuite rappelé les principes des algorithmes sociaux multi-agents et en particulier celui de négociation applicable à notre contexte de travail. Nous avons en particulier présenté le protocole de négociation CNP utilisé par la suite pour expérimenter notre délégation des tâches. Enfin nous avons présenté les méthodes d'évaluation pour ce domaine, notamment la mesure du temps de transfert moyen d'un message sur une plateforme ainsi que l'évaluation des temps de réponse des services d'un agent administratif.

Après cette présentation de l'existant, nous nous sommes intéressés à la mise en place de nos propositions. Dans le Chapitre 4, nous avons proposé la spécification de nos agents embarqués à travers leur cadre d'évolution et leur cycle de vie. Nous avons présenté les deux environnements regroupant les entités cibles pour les interactions de nos agents : l'environnement immédiat pour les entités partageant le même SE et l'environnement hétérogène pour l'ensemble des entités reliées au même réseau. Pour la caractérisation du cycle de vie de nos agents embarqués, nous avons différencié le processus agent de ses threads de services et d'écoute assurant son évolution. Nous avons ensuite proposé une méthodologie de formalisation pour les systèmes multi-agents embarqués. Cette méthode conduit à l'aboutissement d'une table des besoins listant les entités moyens pouvant aider un agent à réaliser ses services. Elle établit également une matrice des interactions regroupant l'ensemble des échanges entre les agents du SMA.

Dans le Chapitre 5, nous avons détaillé notre proposition de MERMAID, une plateforme multi-agents embarquée répondant aux critères présentés lors de l'état de l'art. Nous avons spécifié son architecture séparée en deux blocs : un noyau essentiel à l'application d'un agent embarqué sur SE et un bloc d'administration unitaire par SMA. Ce dernier regroupe notre propositions des trois agents administratifs AMS, DF et ACC. Le noyau apporte une API de communication correspondant à la couche MTPS de la norme FIPA et des bibliothèques de fonctions permettant la gestion du cycle de vie et des interactions de nos agents.

En troisième partie de cette thèse, nous avons proposé plusieurs expérimentations pour éprouver nos propositions et nos hypothèses. Lors du Chapitre 5.3, nous avons évalué notre proposition MERMAID en mesurant son RTT ainsi que les temps de réponse des services principaux de l'agent administratif AMS. Dans le Chapitre 6, nous avons éprouvé notre hypothèse concernant l'apport de flexibilité des SMA aux SE en étudiant le cas du partage d'un zoom. Nous avons évalué nos résultats par une étude performance en relevant le temps de réponse de notre solution et en la comparant à une alternative en programmation séquentielle. Nous avons également analysé la possibilité de réagir dynamiquement à un contexte de traitement défavorable. Nous avons ensuite éprouvé notre hypothèse liée

à la délégation des tâches en Chapitre 7 toujours dans le cadre d'étude du traitement d'images. Dans un premier temps, nous avons analysé le temps de réponse induit par le protocole de négociation CNP. Puis, nous avons évalué l'apport en parallélisation du travail en relevant le temps de réponse d'un traitement groupé de plusieurs images. Nous avons enfin amélioré notre expérimentation relative au contexte défavorable en permettant à nos agents de déléguer dynamiquement les tâches leur ayant été allouées. Nous avons étendu notre protocole d'expérimentation à plusieurs cartes embarquées. A ce stade, nous avons tout d'abord effectué une première expérience sans délégation pour éprouver notre hypothèse concernant la normalisation de notre plateforme. Nous avons finalement appliqué la délégation pour optimiser la répartition des tâches entres les différents systèmes.

8.1.2 Résultats obtenus

A l'issue des Chapitres 4 et 5, nous avons apporté deux propositions théoriques. Nous avons éprouvé notre formalisation d'agents embarqués et notre méthodologie de spécification de SMA embarqués durant nos expérimentations au cours des Chapitres 5.3, 6 et 7. Pour chacune de nos expériences nous avons pu établir des SMA embarqués adaptés à nos cas d'étude et les faire évoluer au fur et à mesure de nos avancées. Nous validons ainsi notre hypothèse #1 concernant la généricité de nos propositions par notre modélisation focalisée en premier lieu sur les interactions de nos agents pour cette expérimentation.

Par ailleurs, l'évaluation de MERMAID en section 5.3 nous a permis d'obtenir un RTT inférieur à 11 ms pour diverses tailles de messages par le biais de communications locales et inférieur à 13 ms pour des communications externes. Lors du passage en multi-cartes, nous avons également eu l'occasion d'éprouver la normalisation de notre plateforme notamment par la validation du fonctionnement de la couche MTPS. Nous avons ainsi pu vérifier notre hypothèse #2 relatives à l'emploi des normes multi-agents pour une plateforme appliquée à l'embarquée.

Le Chapitre 6 était dédié à la mise à l'épreuve de notre hypothèse #3 annonçant un apport de flexibilité pour les SE par l'application d'algorithmes multi-agents. En observant la possibilité d'appliquer deux méthodes d'interpolation distinctes au sein d'un même image, et en faisant varier le choix d'application en fonction du contexte d'exécution de nos agents, nous avons validé notre hypothèse pour ce cas d'étude. Nous avons cependant relevé une faible performance concernant le temps de réponse de notre traitement. Nous avons ainsi pu observer une potentielle limite à l'utilisation des SMA pour des applications temps-réel. En effet, les temps de communication induits par les nombreuses interactions impactent fortement le temps de traitement. Ce coût administratif doit être considéré avec l'apport de flexibilité pour appliquer un équilibre satisfaisant selon les objectifs d'expérimentation.

Enfin à travers le Chapitre 7 nous avons éprouvé notre hypothèse directement liée à l'optimisation des SE par le biais de la délégation des tâches. Ici encore, le temps de réponse est impacté par les nombreuses interactions du SMA. Cependant l'apport de la délégation représente une augmentation de la fiabilité et de l'optimisation de notre SE, soit une plus-value majeure pour ce domaine. Nous avons ainsi pu également vérifier cette hypothèse.

Comme notre thèse représente une phase expérimentale du projet de recherche Tifaifai, nous avons dû créer l'ensemble de nos protocoles d'expérimentation. Pour le domaine d'étude du traitement d'images considéré, nous avons ressenti les limites de la solution multi-agents du fait des temps de communication importants. De plus, l'avancement du projet nous a limité à un support de nos agents uniquement sur l'unité processeur principale de chaque carte. Ce contexte ne nous a pas permis d'exploiter pleinement les apports de la délégation des tâches au sein d'un même SE. Par ailleurs, nous avons effleuré les défis de fiabilité et de réactivité des SE, et n'avons pas traité les aspects de sécurité.

L'expérimentation de nos hypothèses, les succès et les limites constatés nous ont amenés à la formulation de plusieurs perspectives.

8.2 | Extensions et perspectives

Dans ce mémoire, nous avons présenté nos propositions théoriques avec la formalisation des SMA embarqués et la plateforme multi-agents embarquée MERMAID.

Ces propositions nous ont permis d'établir de premières expérimentations sources de vérification de nos hypothèses. Nous avons relevé certaines limites et des possibilités d'améliorations. Nous proposons donc dans cette partie des pistes à explorer sur trois points principaux :

- **l'amélioration des propositions théoriques**.

Notre travail relevant de la phase expérimentale, nous avons établit MERMAID sans l'amener au maximum de son potentiel : de nombreux points d'amélioration restent en suspens. Nous proposons dans une première section des directions de recherche pour étoffer la formalisation de notre plateforme. Nous parlons notamment de la communication de nos SMA jugée coûteuse lors des expérimentations relatives à la flexibilité et à la délégation des tâches. Nous détaillons nos premières réflexions sur ce sujet pour ouvrir la reprise de cette axe de recherche.

- **la poursuite des travaux présentés** concernant la flexibilité et la délégation des tâches.

L'expérimentation de nos hypothèses sur ces points a démontré l'intérêt d'employer les SMA au sein des SE. Nous proposons dans cette section des prémices pour augmenter leur apport vis-à-vis de ces études de cas. Nous présentons des possibilités d'expérimentation pour de nouvelles hypothèses à étudier.

- **l'exploration de nouveaux domaines d'application**.

L'application des algorithmes multi-agents au cœur des systèmes embarqués soulève un intérêt ne se limitant pas aux cas d'application du traitement d'images, pour des notions de gestion des ressources ou de flexibilité. Dans cette section, nous listons d'autres domaines pour lesquels l'expérimentation des SMA pourrait ouvrir des hypothèses de recherches intéressantes.

8.2.1 Perfectionnement de la plateforme MERMAID

8.2.1.1 Optimisation des communications

A travers nos analyses des coûts de communication de nos SMA embarqués, nous avons pu constater l'impact des nombreuses interactions entre nos agents sur le temps de réponse de notre traitement. Nous renouvelons ici notre intérêt pour les possibles thématiques de recherches liées à ces limites relevées.

Actuellement, la plateforme MERMAID intègre la norme FIPA-ACL avec une encapsulation sous format de chaîne de caractères. Ce choix propose une base d'adéquation à la norme. Nous pensons que cette base doit pouvoir évoluer pour une meilleure adéquation au milieu des systèmes embarqués. Notamment, l'utilisation de formats d'information plus compactes comme le bit-length doit permettre d'amoindrir l'empreinte mémoire d'un message et d'améliorer le RTT de notre plateforme. Étant donné le nombre important de messages circulant au sein du SMA, réduire le temps de transfert moyen d'un message doit nous permettre de réduire le temps de réponse globale de nos applications. Cette estimation est proposée ici comme une hypothèse ouverte à de plus amples études.

Nous posons également l'hypothèse que toutes les options implémentées dans la norme FIPA-ACL ne sont pas forcément pertinentes pour le domaine de l'embarqué. Nous pensons qu'une recherche plus poussée sur cette hypothèse devrait permettre de déboucher sur une proposition d'adaptation de la norme pour une utilisation spécialisée à l'embarqué.

L'emploi de cette norme en elle-même reste pertinent en considération des perspectives ouvertes par les objets connectés. En effet, nous nous dirigeons vers des environnements d'évolution très hétérogènes et le respect d'une norme de communication est essentiel pour assurer la compatibilité de l'ensemble.

8.2.1.2 Intégration du multi-niveaux

Un autre facteur influant sur le temps de réponse globale d'une solution multi-agents est l'agent ACC. Toutes les interactions de type requête passent par cet agent, ce qui créé une surcharge de son service. Cet accumulation de requêtes va en effet impacter le temps de réponse relatif à chacune.

Face à cette problématique, nous proposons d'expérimenter le potentiel des systèmes multi-agents multi-niveaux. Cette thématique de recherche a permis d'établir des structures multi-niveaux génériques [Maudet et al., 2015]. Le modèle IODA dont nous nous sommes inspirés pour la formalisation de nos SMA embarqués a également été approfondie sous cette perspective pour devenir PADAWAN [Picault et al., 2010]. Les cas d'études de ces travaux parlent de changement d'échelle, soit de multi-niveaux verticaux.

Pour notre cas, nous sommes intéressés par une possibilité d'extension en multi-niveaux horizontaux. Notre proposition théorique est qu'en cas de surcharge de requêtes, déterminée par un dépassement d'une valeur repère, un agent puisse dupliquer son service. Le déclenchement de cette manœuvre activerait un nouveau thread de traitement permettant une répartition des demandes entre plusieurs services identiques. Ce changement serait transparent pour le reste du système multi-agents qui continuerait de contacter la même interface.

Cette proposition représente une perspective intéressante pour nos travaux de recherche. Elle permet de poser de nouvelles hypothèses d'adaptation et d'optimisation en terme de réactivité de notre SMA. En effet, nous supposons un gain en terme de temps de réponse pour l'ensemble du système multi-agents, ainsi qu'une meilleure capacité d'adéquation à un contexte donné.

8.2.2 Continuité de la gestion des ressources dans les SE

8.2.2.1 Agents-capteurs pour les ressources

Dans le cadre de la gestion des ressources, nous avons présenté nos travaux concernant la délégation des tâches. Ce mécanisme nous permet d'intervenir en correction de l'ordonnanceur en fonction des paramètres du contexte matériel et de l'environnement utilisateur.

Actuellement, nos études se sont concentrées sur des paramètres fixés pour le processus d'expérimentation. La suite logique de ces expériences est de prendre en considération en temps-réel la capacité des ressources impactées par le traitement de la tâche considérée. Pour mesurer les niveaux de capacité d'une ressource, nous avons deux choix :

- la ressource envoie sa capacité aux agents à une certaine fréquence par le biais du protocole D-Bus ;
- un agent dédié récupère la capacité de la ressource à une certaine fréquence par le biais d'un appel système.

La deuxième solution offre la possibilité de n'interroger la ressource qu'en cas de besoin. Elle évite donc l'accumulation de messages transitant sur la fréquence D-Bus. Par ailleurs, pour assurer la réception de la valeur de la capacité dans le cadre de la première solution, il aurait certainement été nécessaire de dédier malgré tout un agent au traitement de cette information.

Nous pouvons associer cette nouvelle spécificité à notre délégation des tâches. En effet, chaque fois que l'unité se verra assigner une tâche, l'agent en sera notifié, vérifiera la disponibilité de l'unité matérielle et déclenchera le processus de délégation si nécessaire.

Nous proposons donc d'étendre nos expérimentations actuelles en intégrant des agents dédiés à la mesure des capacités des ressources matérielles nécessaires au traitement des tâches gérées par le SMA.

8.2.2.2 Intégration des agents sur d'autres ressources matérielles

Pour nos travaux de recherche, nous sommes restés dans le cadre d'une implémentation de nos agents sur CPU. Cette ressource matérielle présente sur une majorité de systèmes embarqués permet l'utilisation de langages de programmation courant pour le développement de nos SMA.

Nous sommes cependant désormais confrontés à une limite de délégation de tâche et de modularité en restant sur cette ressource. Bien qu'ayant implémenté la possibilité d'étendre notre système multi-agents à plusieurs cartes embarquées, nous devons évoluer vers l'intégration de nos agents sur d'autres ressources matérielles pour exploiter pleinement le potentiel d'optimisation de la gestion des ressources par SMA.

Dans un premier temps, nous proposons d'explorer la possibilité d'intégrer des agents à un GPU pour traiter des tâches graphiques ou à un DSP pour traiter des tâches mathématiques complexes. Notre SMA pourra alors proposer une modularité plus importante en exploitant les avantages des différentes ressources matérielles pour un même type de travail. On comparera ainsi le rendu GPU et CPU pour un traitement graphique, ou le rendu DSP et CPU pour un traitement mathématique, pour déterminer les paramètres du contexte matériel et de l'environnement utilisateur qui seront pris en compte par le SMA. Ce dernier pourra ensuite offrir une qualité de service plus adaptée en renvoyant la tâche vers la ressource matérielle la plus adaptée à un contexte donné. Par exemple le décodage d'une image H264 pourrait aussi bien être traitée par le CPU principal, par un CPU secondaire, par un accélérateur matériel ou par le GPU, chacun offrant des spécificités de traitement.

Cette perspective de travail nécessite de reprendre la base du développement d'un agent avec un nouveau langage de programmation adapté à la ressource matérielle considérée. La compatibilité de l'agent doit être assurée par le respect des normes de communication supportées par la plateforme MERMAID afin que le fonctionnement interne de l'agent soit ensuite transparent pour le reste du SMA.

Dans cette optique, nous serons également plus à même d'offrir une réelle optimisation du système embarqué en ouvrant la possibilité de déléguer les tâches entre différentes ressources matérielles.

8.2.3 Domaines d'application à explorer

Pour ce mémoire, nous avons concentré nos expérimentations au cadre du traitement d'images. Ce domaine d'application offre un rendu visuel des résultats et nous a permis d'observer concrètement les choix réactifs de nos agents à des contextes donnés. Il fut également choisi en lien avec le système embarqué considéré pour ce mémoire et fourni par la société partenaire de cette thèse STMicroelectronics : une Set-Top box. Ces SE incorporent de nombreux algorithmes de traitement multimédia ainsi que les ressources matérielles adéquates (encodeur, décodeur, etc.).

Pour autant, la thématique de base du projet de recherche dans lequel cette thèse s'inscrit et qui propose d'utiliser des SMA au cœur des SE s'étend à bien d'autres domaines d'application. Ici nous présentons deux de ces multiples domaines d'application possibles.

8.2.3.1 Sécurité

La sécurité des systèmes embarqués représente l'une des quatre catégories de défis majeurs pour le domaine des SE tel que nous l'avions évoqué dans la partie 2.1 de ce manuscrit. Que ce soit pour la confidentialité des données, ou pour l'assurance de l'intégrité du système, l'emploi d'un SMA offre des perspectives de recherche intéressantes. Une deuxième thèse CIFRE a débuté en octobre 2017 en partenariat avec l'entreprise STMicroelectronics. Basée sur les travaux présentés dans ce manuscrit, elle se focalise sur la sécurité au sein des SE par la mise en œuvre de SMA.

8.2.3.2 Éducation

Si les propositions théoriques de ce travail de recherche sont effectivement indépendantes des domaines d'application, il n'en reste pas moins que le Centre de Recherche en Éducation de Nantes (CREN) est un laboratoire tourné vers des contextes éducatifs. Développer notre recherche vers une application dans le domaine de l'éducation représente donc une perspective déterminante.

Notre thématique plutôt éloignée nous a permis de proposer des apports théoriques ré-utilisables sur n'importe quel support informatique. Dans le cadre d'un environnement utilisateur dédié à l'éducation, nous pourrions tout à fait utiliser la plateforme MERMAID pour intégrer des agents aux SE utilisés par les élèves et le professeur. La délégation des tâches pourrait alors se faire selon des paramètres relatifs au cours (travail terminé, réception d'une correction, affichage du travail d'un élève pour toute la classe, etc.).

Les possibilités d'interactions à développer dans ce cadre sont ouvertes avec la possibilité de reprendre notre travail et nos propositions théoriques pour se concentrer sur l'utilisation du SMA dans ce cadre particulier.

Pour chacun de ces nouveaux domaines d'application, notre méthodologie de formalisation d'un SMA embarqué et notre plateforme MERMAID peuvent être repris et adaptés. Nous espérons ainsi avoir ouvert une porte vers un nouveaux domaine de recherche à part entière.

Bibliographie

[Maudet et al., 2015] Maudet, A., Touya, G., Duchêne, C., and Picault, S. (2015). Patterns multi-niveaux pour les sma. In *Journées francophones sur les systèmes multi-agents (JFSMA)*, pages 139–148. Cépaduès Édition.

[Picault et al., 2010] Picault, S., Mathieu, P., and Kubera, Y. (2010). Padawan, un modèle multi-échelles pour la simulation orientée interactions. In *Journées francophones sur les systèmes multi-agents (JFSMA)*, pages 193–202. Cépaduès Édition.

Bibliographie complète

[Abras, 2009] Abras, S. (2009). *Système domotique multi-agents pour la gestion de l'énergie dans l'habitat*. PhD thesis, Institut Polytechnique de Grenoble, Laboratoire d'Informatique de Grenoble, IMAG.

[Abras et al., 2013] Abras, S., Kieny, C., Ploix, S., and Wurtz, F. (2013). Mas architecture for energy management : Developing smart networks with jade platform. In *International Conference on Smart Instrumentation, Measurement and Applications (ICSIMA)*, Kuala Lumpur, Malaysia. IEEE Computer Society Press.

[Abras et al., 2009] Abras, S., Ploix, S., Pesty, S., and Jacomino, M. (2009). Apport d'une approche multi-agents pour la résolution d'un probleme de gestion de l'energie dans l'habitat. In *17èmes Journées francophones sur les systèmes multi-agents (JFSMA)*, pages 110–116, Lyon, France. Cépaduès Édition.

[Adam et al., 2017] Adam, C., Dugdale, J., and Garbay, C. (2017). Modélisation multi-agent de la cohésion sociale en situation de crise. In *Journées francophones sur les systèmes multi-agents (JFSMA)*, pages 1–4. Cépaduès Éditions.

[AlEnawy and Aydin, 2005] AlEnawy, T. A. and Aydin, H. (2005). Energy-aware task allocation for rate monotonic scheduling. In *11th IEEE Real Time on Embedded Technology and Applications Symposium (RTAS)*, pages 213–223. IEEE Computer Society Press.

[Amouroux et al., 2013] Amouroux, E., Huraux, T., Sempé, F., Sabouret, N., and Haradji, Y. (2013). Simulating human activities to investigate household energy consumption. In *5th International Conference on Agents and Artificial Intelligence (ICAART)*, pages 71–80. Springer.

[Amri et al., 2009] Amri, H., Khalfallah, A., and Bouhlel, M. S. (2009). Evaluation de techniques d'agrandissement des images. In *International Conference : Sciences of Electronic, Technologies of Information and Telecommunications (SETIT)*, pages 1–6. IEEE Computer Society Press.

[Aza-Vallina et al., 2011] Aza-Vallina, D., Denis, B., and Faure, J. (2011). *Communications reliability analysis in networked embedded systems*, pages 2594–2602. CRC Press.

[Balaji et al., 2017] Balaji, M., Aarthi, E., Kalpana, K., Nivetha, B., and Suganya, D. (2017). Adaptable and reliable industrial security system using pic controller. *International Journal for Innovative Research in Science & Technology (IJIRST)*, 3(12) :56–60.

[Beaumont et al., 2006] Beaumont, O., Marchal, L., Rehn, V., and Robert, Y. (2006). Fifo scheduling of divisible loads with return messages under the one-port model. In *International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1–25. IEEE Computer Society Press.

[Beisel et al., 2011] Beisel, T., Wiersema, T., Plessl, C., and Brinkmann, A. (2011). Cooperative multitasking for heterogeneous accelerators in the linux completely fair scheduler. *Application-Specific Systems, Architectures, and Processors (ASAP)*, pages 223–226.

[Bellifemine et al., 2005] Bellifemine, F., Bergenti, F., Caire, G., and Poggi, A. (2005). *Jade — A Java Agent Development Framework*, pages 125–147. Springer US.

[Benhani et al., 2017] Benhani, E. M., Marchand, C., Aubert, A., and Bossuet, L. (2017). On the security evaluation of the arm trustzone extension in a heterogeneous soc. In *30th IEEE International System-on-chip Conference (SOCC)*, pages 1–6. IEEE Computer Society Press.

[Bergenti and Ricci, 2002] Bergenti, F. and Ricci, A. (2002). Three approaches to the coordination of multiagent systems. In *2002 ACM Symposium on Applied Computing (SAC)*, pages 367–372, New York, NY, USA. ACM.

[Berrandjia et al., 2015] Berrandjia, M. L., Ourari, S., and Chalal, R. (2015). Un protocole de négociation pour l'ordonnancement distribué multi-agents d'un atelier job-shop. In *Séminaire National en informatique à Biskra (SNIB)*, page 1–8.

[Bonnet and Tessier, 2008] Bonnet, G. and Tessier, C. (2008). Evaluer un système multiagent physique - retour sur expérience. In *Systèmes Multi-Agents, Communautés virtuelles et naturelles - Journées francophones sur les systèmes multi-agents (JFSMA)*, pages 13–22. Cépaduès Éditions.

[Bordini et al., 2006] Bordini, R. H., Braubach, L., Dastani, M., El, A., Seghrouchni, F., Gomez-sanz, J. J., Leite, J., Pokahr, A., and Ricci, A. (2006). A survey of programming languages and platforms for multi-agent systems. *Informatica*, 30(1) :33–44.

[Boussard et al., 2007] Boussard, M., Bouzid, M., and Mouaddib, A.-I. (2007). La décision multi-critère pour la coordination locale dans les systèmes multi-agents. *Annales du Lamsade*, 8.

[Brazier et al., 1997] Brazier, F. M. T., Dunin-keplicz, B. M., and Jennings, N. R. (1997). Desire : Modeling multi-agent systems in a compositional formal framework. 6(1) :67–94.

[Brazier et al., 2002] Brazier, F. M. T., Jonker, C. M., and Treur, J. (2002). Principles of component-based design of intelligent agents. *Data and Knowledge Engineering*, 41(1) :1–27.

[Bridges et al., 2016] Bridges, R. A., Imam, N., and Mintz, T. M. (2016). Understanding gpu power : A survey of profiling, modeling, and simulation methods. *ACM Computing Surveys*, 49(3) :1–27.

[Calvaresi et al., 2018] Calvaresi, D., Appoggetti, K., Lustrissimini, L., Marinoni, M., Sernani, P., Dragoni, A. F., and Schumacher, M. (2018). Multi-agent systems' negotiation protocols for cyber-physical systems : Results from a systematic literature review. In *10th International Conference on Agents and Artificial Intelligence (ICAART 2018)*, pages 224–235.

[Carbo et al., 2016] Carbo, J., Sanchez-Pi, N., and Molina, J. M. (2016). Agent-based simulation with netlogo to evaluate ambient intelligence scenarios. *Journal of Simulation*, pages 1–10.

[Chao and Fang, 2012] Chao, L. and Fang, H. (2012). Research on reliability design of data storage for embedded system. *Physics Procedia, International Conference on Solid State Devices and Materials Science*, 25 :1405 – 1408.

[Chen and Thropp, 2007] Chen, J. Y. and Thropp, J. E. (2007). Review of low frame rate effects on human performance. *IEEE Transactions on systems, man and cybernetics—PART A : Systems and humans*, 37(6) :1063–1076.

[Chin et al., 2014] Chin, K. O., Gan, K. S., Alfred, R., Anthony, P., and Lukose, D. (2014). Agent architecture : An overview. *Transactions on Science and Technology*, 1(1) :18–35.

[Coates et al., 2000] Coates, G., Whitfield, R. I., Duffy, A. H. B., and Hills, B. (2000). Coordination approaches and systems – part ii : An operational perspective. *Research in Engineering Design*, 12(2) :73 – 89.

[consortium, 2003] consortium, F. (2003). *FIPA Communicative Act Library, Specification and FIPA ACL Message Structure Specification*. Technical report.

[Cossentino et al., 2003] Cossentino, M., Sabatucci, L., and Chella, A. (2003). A possible approach to the development of robotic multi-agent systems. In *IEEE/WIC International Conference on Intelligent Agent Technology (IAT)*, pages 539–548. IEEE Computer Society Press.

[Demazeau, 1995] Demazeau, Y. (1995). From interactions to collective behaviour in agent-based systems. In *1st European Conference on Cognitive Science*, pages 117–132.

[Demazeau and Costa, 1996] Demazeau, Y. and Costa, A. C. R. (1996). Populations and organizations in open multi-agent systems. In *1st National Symposium on Parallel and Distributed AI*, pages 1–13.

[Et-Tolba et al., 2015] Et-Tolba, E. H., Ouassaid, M., and Maaroufiand, M. (2015). An implementation of fipa contract net interaction protocol adapted for smart home agents simulation. In *3rd International Renewable and Sustainable Energy Conference (IRSEC)*, pages 1083–1087. IEEE Computer Society Press.

[Etiemble, 2016] Etiemble, D. (2016). *Introduction aux systèmes embarqués, enfouis et mobiles*. Editions Techniques de l'Ingénieur.

[Ferber, 1995] Ferber, J. (1995). *Les Systèmes multi-agents : vers une intelligence collective*. InterEditions.

[Ferber, 1997] Ferber, J. (1997). Les systèmes multi-agents : un aperçu général. *Technique et Science Informatiques*, pages 979–1012.

[Ferber and Gutknecht, 1998] Ferber, J. and Gutknecht, O. (1998). A meta-model for the analysis and design of organizations in multi-agent systems. In *International Conference on Multi Agent Systems (ICMAS)*, pages 128–135, Paris, France. IEEE Computer Society Press.

[FIPA, 2002] FIPA (2002). Fipa standard status specifications. http://www.fipa.org/repository/standardspecs.html. Accessed : 2018-04-16.

[García et al., 2012] García, C., Cárdenas, P. F., Puglisi, L. J., and Saltaren, R. (2012). Design and modeling of the multi-agent robotic system : Smart. *Robotics and Autonomous Systems*, 60(2) :143–153.

[Georgeff et al., 1999] Georgeff, M. P., Pell, B., Pollack, M. E., Tambe, M., and Wooldridge, M. (1999). The belief-desire-intention model of agency. In *5th International Workshop on Intelligent Agents V, Agent Theories, Architectures, and Languages*, pages 1–10, London, UK, UK. Springer.

[Gherbi et al., 2013] Gherbi, T., Borne, I., and Meslati, D. (2013). An mde approach to develop mobile-agents applications. *Communications in Computer and Information Science*, 417 :64–80.

[Gil-Quijano and Sabouret, 2010] Gil-Quijano, J. and Sabouret, N. (2010). Prediction of humans' activity for learning the behaviors of electrical appliances in an intelligent ambiant environment. In *IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT)*, page 283–286. IEEE Computer Society Press.

[Goel and Garg, 2012] Goel, N. and Garg, R. (2012). A comparative study of cpu scheduling algorithms. *International Journal of Graphics & Image Processing (IJGIP)*, 2(4) :245–251.

[Grignard et al., 2013] Grignard, A., Taillandier, P., Gaudou, B., Vo, D. A., Huynh, N. Q., and Drogoul, A. (2013). Gama 1.6 : Advancing the art of complex agent-based modeling and simulation. In Springer, editor, *Pacific Rim international Conference on Multi-Agents (PRIMA)*, volume 8291, pages 117–131, Dunedin, New Zealand.

[Gupta and Kansal, 2011] Gupta, R. and Kansal, G. (2011). A survey on comparative study of mobile agent platforms. *International Journal of Engineering Science and Technology (IJEST)*, 3(3) :1943–1948.

[Gutknecht and Ferber, 2000] Gutknecht, O. and Ferber, J. (2000). The madkit agent platform architecture. In *In Agents Workshop on Infrastructure for Multi-Agent Systems*, pages 48–55. Springer.

[Habiba et al., 2014] Habiba, U., Khan, S. A., and Javed, Y. (2014). Gap analysis in software engineering process adoption in implementing high end embedded system design. In *4th International Conference on Circuit, System and Simulation (ICCSS)*, pages 495–503, Department of Computer Engineering, College of EME, National University of Sciences and Technology (NUST), Islamabad, Pakistan. TextRoad Publication.

[Hammarlund et al., 2014] Hammarlund, P., Martinez, A. J., Bajwa, A. A., Hill, D. L., Hallnor, E., Jiang, H., Dixon, M., Derr, M., Hunsaker, M., Kumar, R., Osborne, R. B., Rajwar, R., Singhal, R., D'Sa, R., Chappell, R., Kaushik, S., Chennupaty, S., Jourdan, S., Gunther, S., Piazza, T., and Burton, T. (2014). Haswell : The fourth-generation intel core processor. *IEEE Micro*, 34(2) :6–20.

[Hauck and DeHon, 2007] Hauck, S. and DeHon, A. (2007). *Reconfigurable Computing : The Theory and Practice of FPGA-Based Computation*. Systems on Silicon. Morgan Kaufmann.

[Herrmann et al., 2014] Herrmann, B., Lang, C., and Rousset, A. (2014). Étude comparative des plateformes parallèles pour systèmes multi-agents. In *Conférence en parallélisme, architecture et systèmes (ComPAS)*, pages 1–12, Neuchâtel, Suisse.

[Huang et al., 2011] Huang, J., Blech, J. O., Raabe, A., Buckl, C., and Knoll, A. (2011). Analysis and optimization of fault-tolerant task scheduling on multiprocessor embedded systems. In *7th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, pages 247–256. ACM.

[Hung, 2017] Hung, M. (2017). Leading the iot, gartner insights on how to lead in a connected world. Technical report, Gartner.

[Iñigo-Blasco et al., 2012] Iñigo-Blasco, P., del Rio, F., Romero-Ternero, M., DanielCagigas-Muñiz, and SaturninoVicente-Diaz (2012). Robotics software frameworks for multi-agent robotic systems development. *Robotics and Autonomous Systems*, 60(6) :803–821.

[Jagga et al., 2015] Jagga, A., Juneja, D., and Singh, A. (2015). Updates in knowledge query manipulation language for complex multiagent systems. *International Journal of Computing Academic Research (IJCAR)*, 4(1) :19 – 26.

[Jamont, 2005] Jamont, J.-P. (2005). *DIAMOND : Une approche pour la conception de systèmes multi-agents embarqués*. PhD thesis, Institut National Polytechnique de Grenoble - INPG.

[Jamont and Occello, 2007] Jamont, J.-P. and Occello, M. (2007). Designing embedded collective systems : The diamond multiagent method. In *19th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 91–94, Washington, DC, USA. IEEE Computer Society Press.

[Jurasovic et al., 2006] Jurasovic, K., Jezic, G., and Kusek, M. (2006). A performance analysis of multi-agent systems. *International Transactions on Systems Science and Applications (ITTSA)*, 1(4) :335–342.

[Kaur et al., 2013] Kaur, S., Kaur, H., and Sehra, S. K. (2013). Modification of contract net protocol(cnp) : A rule-updation approach. *International Journal of Advanced Computer Science and Applications (IJACSA)*, 4(11) :40–46.

[Kinny et al., 1996] Kinny, D., Georgeff, M., and Rao, A. (1996). A methodology and modelling technique for systems of bdi agents. In *European Workshop on Modelling Autonomous Agents in a Multi-agent World : Agents Breaking Away : Agents Breaking Away (MAAMAW)*, pages 56–71, Secaucus, NJ, USA. Springer.

[Kleiner et al., 2013] Kleiner, A., Farinelli, A., Ramchurn, S., Shi, B., Mafioletti, F., and Refatto, R. (2013). Rmasbench : benchmarking dynamic multi-agent coordination in urban search and rescue (extended abstract). In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1195–1196. International Foundation for Autonomous Agents and Multiagent Systems.

[Kocher et al., 2004] Kocher, P., Lee, R., McGraw, G., Raghunathan, A., and Ravi, S. (2004). Security as a new dimension in embedded system design. In *41st Annual Design Automation Conference (DAC)*, pages 753–760. ACM.

[Kravari and Bassiliades, 2015] Kravari, K. and Bassiliades, N. (2015). A survey of agent platforms. *Journal of Artificial Societies and Social Simulation (JASSS)*, 18(1) :11–28.

[Kubera et al., 2010] Kubera, Y., Mathieu, P., and Picault, S. (2010). Everything can be agent ! In van der Hoek, W., Kaminka, G., Lespérance, Y., Luck, M., and Sen, S., editors, *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1547–1548. International Foundation for Autonomous Agents and Multiagent Systems.

[Kubera et al., 2011] Kubera, Y., Mathieu, P., and Picault, S. (2011). Ioda : An interaction-oriented approach for multi-agent based simulations. *Journal of Autonomous Agents and Multiagent Systems (AAMAS)*, 23(3) :303–343.

[Kumar and Indira, 2017] Kumar, G. S. and Indira, S. (2017). Implementation of pic microcontroller based boost converter for solar panel without battery backup. *International Journal of Scientific Research in Science, Engineering and Technology (IJSRSET)*, 3(6) :52–55.

[Kwak et al., 2012] Kwak, J., Varakantham, P., Maheswaran, R., Tambe, M., Jazizadeh, F., Kavulya, G., Klein, L., Becerik-Gerber, B., Hayes, T., and Wood, W. (2012). Saves : A sustainable multiagent application to conserve building energy considering occupants. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, volume 1, pages 21–28. International Foundation for Autonomous Agents and Multiagent Systems.

[Laurgeau et al., 2002] Laurgeau, C., Steux, B., and Metman, G. (2002). Outils pour le prototypage des systèmes embarqués temps réel. *Revue Jautomatise*, 21(21) :58–62.

[Lavendelis and Grundspenkis, 2014] Lavendelis, E. and Grundspenkis, J. (2014). Multi-agent auction based simulation tool for an insurance policy market. *Applied Computer Systems*, 15(1) :5–13.

[Lazarin and Pantoja, 2015] Lazarin, N. M. and Pantoja, C. E. (2015). A robotic-agent platform for embedding software agents using raspberry pi and arduino boards. In *Software Agents, Environments and Applications School (WESAAC)*, pages 13–20. UFAL, UFF, IME.

[Liekna et al., 2012] Liekna, A., Lavendelis, E., and Grabovskis, A. (2012). Experimental analysis of contract net protocol in multi-robot task allocation. *Applied Computer Systems*, 13(1) :6–13.

[LIFL, 2011] LIFL (2011). Ioda ants. http://www.lifl.fr/SMAC/projects/ioda/ioda/models/iodaants/index.html. Accessed : 2018-04-16.

[Liu and Layland, 1973] Liu, C. L. and Layland, J. W. (1973). Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1) :46–61.

[Lupetti and Zagorodnov, 2006] Lupetti, S. and Zagorodnov, D. (2006). Data popularity and shortest-job-first scheduling of network transfers. In *International Conference on Digital Telecommunications (ICDT)*, pages 26–32. IEEE Computer Society Press.

[Mamidi et al., 2012] Mamidi, S., Chang, Y.-H., and Maheswaran, R. (2012). Improving building energy efficiency with a network of sensing, learning and prediction agents. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, volume 1, pages 45–52, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

[Mariani et al., 2013] Mariani, G., Sima, V.-M., Palermo, G., Zaccaria, V., Marchiori, G., Silvano, C., and Bertels, K. (2013). Run-time optimization of a dynamically reconfigurable embedded system through performance prediction. *International Journal for Innovative Research in Science & Technology (IJIRST)*, pages 1–8.

[Maudet et al., 2015] Maudet, A., Touya, G., Duchêne, C., and Picault, S. (2015). Patterns multi-niveaux pour les sma. In *Journées francophones sur les systèmes multi-agents (JFSMA)*, pages 139–148. Cépaduès Édition.

[Mayowa et al., 2016] Mayowa, O., M, F. T., Olabiyisi, S. O., Omidiora, E., and Fawole, A. (2016). A survey on migration process of mobile agent. *World Congress on Engineering and Computer Science (WCECS)*, 1 :415–419.

[Meedeniya et al., 2010] Meedeniya, I., Buhnova, B., Aleti, A., and Grunske, L. (2010). Architecture-driven reliability and energy optimization for complex embedded systems. In *6th International Conference on Quality of Software Architectures : Research into Practice - Reality and Gaps (QoSA)*, pages 52–67. Springer.

[Michel, 2013] Michel, F. (2013). Intégration du calcul sur gpu dans la plate-forme de simulation multi-agent générique turtlekit 3. In *Journées francophones sur les systèmes multi-agents (JFSMA)*, pages 189–198. Cépaduès Édition.

[Midonnet et al., 2010] Midonnet, S., Masson, D., and Lassalle, R. (2010). Slack-time computation for temporal robustness in embedded systems. *IEEE Embedded Systems Letters*, 2(4) :119–122.

[Minar et al., 1996] Minar, N., Burkhart, R., Langton, C., and Askenazi, M. (1996). The swarm simulation system : A toolkit for building multi-agent simulations. Technical report, Santa Fe Institute.

[Miraoui et al., 2016] Miraoui, M., El-etriby, S., Abid, A. Z., and Tadj, C. (2016). Agent-based context-aware architecture for a smart living room. *International Journal of Smart Home*, 10(5) :39–54.

[Mittal, 2014] Mittal, S. (2014). A survey of techniques for improving energy efficiency in embedded computing systems. *International Journal of Computer Aided Engineering and Technology (IJCAET)*, 6(4) :440–459.

[Mulet et al., 2006] Mulet, L., Such, J. M., and Alberola, J. M. (2006). Performance evaluation of open-source multiagent platforms. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1107–1109, New York, NY, USA. ACM.

[Narayanan and Xie, 2006] Narayanan, V. and Xie, Y. (2006). Reliability concerns in embedded system designs. *Computer*, 39(1) :118–120.

[Nikolai and Madey, 2009] Nikolai, C. and Madey, G. (2009). Tools of trade : A survey of various agent based modeling platforms. *Journal of Artificial Societies and Social Simulation (JASSS)*, 12(2) :2–64.

[Paletta and Herrero, 2010] Paletta, M. and Herrero, P. (2010). A mas-based negotiation mechanism to deal with saturated conditions in distributed environments. *International Conference on Agents and Artificial Intelligence (ICAART)*, 2 :1–7.

[Pantoja et al., 2016] Pantoja, C. E., Junior, M. F. S., Lazarin, N. M., and Sichman, J. S. (2016). Argo : A customized jason architecture for programming embedded robotic agents. In *4th International Workshop on Engineering Multi Agent Systems (EMAS 2016)*, pages 133–148. Springer.

[Pappalardo et al., 2017] Pappalardo, S., Alessandro, A., Santagati, C., Ardizzone, C., Ribellino, C., Mirabella, I., Principato, O., and Tonicello, F. (2017). Miniaturised integrated circuit core element for point of load (pol) conversion. *European Space Power Conference (ESPC)*, 16 :1–8.

[Pawlowski et al., 2014] Pawlowski, K., Kurach, K., Svensson, K., Ramchurn, S., Michalak, T. P., and Rahwan, T. (2014). Coalition structure generation with the graphics processing unit. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 293–300, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

[Pereau, 2009] Pereau, J. C. (2009). Négociation et théorie des jeux : les « dessous » d'un accord acceptable. *Négociations*, 12(2) :37–51.

[Perera et al., 2015] Perera, C., Liu, C. H., and Jayawardena, S. (2015). The emerging internet of things marketplace from an industrial perspective : A survey. *IEEE Transactions on Emerging Topics in Computing*, 3(4) :585–598.

[Picault, 2013] Picault, S. (2013). *De la simulation multi-agents à la simulation multi-niveaux. Pour une réification des interactions.* Habilitation à diriger des recherches, Université des Sciences et Technologie de Lille - Lille I.

[Picault et al., 2010] Picault, S., Mathieu, P., and Kubera, Y. (2010). Padawan, un modèle multi-échelles pour la simulation orientée interactions. In *Journées francophones sur les systèmes multi-agents (JFSMA)*, pages 193–202. Cépaduès Édition.

[Poggi, 2008] Poggi, A. (2008). Developing multi-user online games with agents. *Journal WSEAS Transactions on Computers*, 7(8) :1240–1249.

[Poslad, 2007] Poslad, S. (2007). Specifying protocols for multi-agent systems interaction. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 2(4) :51–75.

[Rahwan and Dignum, 2009] Rahwan, I. and Dignum, F. (2009). Formal analysis of interest-based negotiation. *Annals of Mathematics and Artificial Intelligence*, 55(3) :253–276.

[Ranganadh et al., 2011] Ranganadh, N., Patel, P., and Grigoryan, A. (2011). Performances of texas instruments dsp and xilinx fpgas for cooley-tukey and grigoryan fft algorithms. *Journal of Engineering and Technology*, 1(2) :83–87.

[Rasmussen and Trick, 2008] Rasmussen, R. V. and Trick, M. A. (2008). Round robin scheduling - a survey. *European Journal of Operational Research*, 188(3) :617–636.

[Russell and Norvig, 2009] Russell, S. J. and Norvig, P. (2009). *Artificial Intelligence : A Modern Approach (3rd Edition)*, chapter number 2 : Intelligent agents, pages 34–59. Prentice Hall.

[Rutagangibwa and Krishnamurthy, 2014] Rutagangibwa, V. and Krishnamurthy, B. (2014). A survey on the implementation of real time systems for industrial automation applications. *International Journal for Innovative Research in Science & Technology (IJIRST)*, 1(7) :174–177.

[Sabouret, 2009] Sabouret, N. (2009). *Interactions about Actions in Open and Heterogenous Multi-Agent Systems*. Habilitation à diriger des recherches, Université Pierre et Marie Curie - Paris VI.

[Sangiovanni-Vincentelli, 2007] Sangiovanni-Vincentelli, A. (2007). Quo vadis, sld ? reasoning about the trends and challenges of system level design. *IEEE*, 95(3) :467–506.

[Scharf et al., 2002] Scharf, E., Hamer, P., Smparounis, K., Payer, W., Ronan, J., and Crotty, M. (2002). An intelligent integrated approach to multi-service residential access networks. In *41st FITCE European Telecommunications Congress*, pages 1–6. FITCE.

[Sifakis, 2011] Sifakis, J. (2011). A vision for computer science — the system perspective. *Central European Journal of Computer Science*, 1 :108–116.

[Sifakis and Henzinger, 2007] Sifakis, J. and Henzinger, T. A. (2007). The discipline of embedded systems design. *Computer*, 40 :32–40.

[Smarsly et al., 2012] Smarsly, K., Law, K. H., and Hartmann, D. (2012). A multi-agent-based collaborative framework for a self-managing structural health monitoring system. *Journal of Computing in Civil Engineering*, 26(1) :76–89.

[Smith, 1980] Smith, R. G. (1980). The contract net protocol : High-level communication and control in a distributed problem solver. *Transactions on Computers*, C-29(12) :1104–1113.

[Stavropoulos et al., 2014] Stavropoulos, T. G., Rigas, E. S., Kontopoulos, E., Bassiliades, N., and Vlahavas, I. (2014). A multi-agent coordination framework for smart building energy management. In *25th International Workshop on Database and Expert Systems Applications (DEXA)*, pages 126–130. IEEE Computer Society Press.

[Stilkerich et al., 2014] Stilkerich, S., Siemers, C., and Ristig, C. (2014). Appropriate multi-core architecture for safety-critical aerospace applications - certifiable real-time switching network. In *International Conference on Pervasive and Embedded Computing and Communication Systems (PECCS)*, pages 180–185. SciTePress.

[Subagdja and Tan, 2014] Subagdja, B. and Tan, A.-H. (2014). On coordinating pervasive persuasive agents. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1467–1468, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

[Sun et al., 2013] Sun, Q., Yu, W., Kochurov, N., Hao, Q., and Hu, F. (2013). A multi-agent-based intelligent sensor and actuator network design for smart house and home automation. *Journal of Sensor and Actuator Networks*, 2(3) :557–588.

[Vulgarakis and Seceleanu, 2008] Vulgarakis, A. and Seceleanu, C. (2008). Embedded systems resources : Views on modeling and analysis. In *38th Annual Computer Software and Applications Conference (COMPSAC)*, pages 1321–1328, Los Alamitos, CA, USA. IEEE Computer Society Press.

[Waghmare and Mohite-Patil, 2012] Waghmare, N. and Mohite-Patil, P. T. (2012). Comparison of embedded system and fpga application for universal multi-resource bus. *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)*, 1 :23–29.

[Wang et al., 2014] Wang, J., Zhong, S., Yan, L., and Cao, Z. (2014). An embedded system-on-chip architecture for real-time visual detection and matching. *IEEE Trans. Cir. and Sys. for Video Technol.*, 24(3) :525–538.

[Want et al., 2002] Want, R., Pering, T., Borriello, G., and Farkas, K. I. (2002). Disappearing hardware. *IEEE Pervasive Computing*, 1(1) :36–47.

[Wood and Deloach, 2000] Wood, M. and Deloach, S. A. (2000). An overview of the multiagent systems engineering methodology. In *The First International Workshop on Agent-Oriented software Engineering (AOSE)*, pages 207–222. Springer.

[Wooldridge et al., 2000] Wooldridge, M., Jennings, N. R., and Kinny, D. (2000). The gaia methodology for agent-oriented analysis and design. *Journal of Autonomous Agents and Multiagent Systems (AAMAS)*, 3 :285–312.

[Xie et al., 2017] Xie, G., Chen, Y., Liu, Y., Wei, Y., Renfa, L., and Li, K. (2017). Resource consumption cost minimization of reliable parallel applications on heterogeneous embedded systems. *IEEE Transactions on Industrial Informatics*, 13(4) :1629–1640.

[Zheng et al., 2014] Zheng, R., Xu, Y., Chakraborty, N., and Sycara, K. (2014). Multiagent coordination for demand management with energy generation and storage. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1587–1588. International Foundation for Autonomous Agents and Multiagent Systems.

Publications réalisées au cours des travaux de la thèse

[Inguère et al., 2016] Inguère, T., Renault, V., Carlier, F., and Leroux, P. (2016). Systèmes multi-agents pour l'ordonnancement de ressources dans les systèmes embarqués. In *Rencontres Jeunes Chercheurs en Intelligence Artificielle (RJCIA)*, pages 25–32, Clermont-ferrand, France.

[Inguère et al., 2016] Inguère, T., Renault, V. and Carlier, F. (2016). Modèle multi-agents centré interactions adapté aux systèmes embarqués (Poster). In *Journées Francophones sur les Systèmes Multi-Agents (JFSMA'2016)*, Rouen, France.

[Inguère et al., 2016] Inguère, T., Carlier, F., and Renault, V. (2016). Multi-agents system for modular image processing (Poster). In *Design & Architectures for Signal & Image Processing (DASIP)*, Rennes, France.

[Inguère et al., 2016] Inguère, T., Carlier, F., and Renault, V. (2016). Flexible image processing in Embedded Systems using Multi-agents Systems. In *14th IFAC Conference on Programmable Devices and Embedded Systems (PDES)*, 49(25) :164–169, Brno, Czech Republic.

[Inguère et al., 2017] Inguère, T., Renault, V. and Carlier, F. (2017). Délégation de tâches sur la plateforme multi-agents embarquée MERMAID (Poster). In *Journées Francophones sur les Systèmes Multi-Agents (JFSMA'2017)*, Caen, France.

[Inguère et al., 2017] Inguère, T., Carlier, F., and Renault, V. (2017). Task Delegation through Multi-Agent Negotiation in Embedded Systems by the Platform MERMAID. In *IEEE International Conference on Vehicular Technology Conference (VTC-Fall)*, pages 1–5, Toronto, ON, Canada. IEEE Computer Society Press.

Table des acronymes

A

- **ACC** : *Agent Communication Channel*, canal de communication entre agents, agent administratif
- **AID** : *Agent IDentifier*, identifiant pour les agents
- **AMS** : *Agent Management System*, système de gestion d'agents, agent administratif
- **API** : *Application Programming Interface*, interface de programmation applicative
- **ASIC** : *Application-Specific Integrated Circuit*, circuits intégrés dédiés à une application

B

- **BDI** : *Belief Desire Intention*, modèle agent basé sur les croyances, les désirs et les engagements
- **BL** : "Bi-Linéaire", méthode d'interpolation

C

- **Cfp** : *Call for proposals*, appel à propositions, étape du protocole de négociation Contract Net
- **CNP** : *Contract Net Protocol*, protocole d'interaction de "réseau contractuel", utilisé en négociation
- **CPU** : *Central Processing Unit*, unité processeur centrale, composant matériel

D

- **DF** : *Directory Facilitator*, facilitateur d'annuaire, agent administratif
- **DIAMOND** : *Decentralized Iterative Multiagent Open Networks Design*, méthode de co-design dédiée aux systèmes multi-agents
- **DMIPS** : *Dhrystone Million Instructions Per Second*, unité de mesure de la puissance d'un processeur
- **DSP** : *Digital Signal Processor*, microprocesseurs optimisés pour des applications liées au traitement numérique du signal

F

- **FA** : Files d'Attente
- **FIFO** : *First In First Out*, algorithme d'ordonnancement
- **FIPA** : *Foundation for Intelligent Physical Agents*, norme multi-agents
- **FIPA-ACL** : *Foundation for Intelligent Physical Agents - Agent Communication Language*, norme multi-agents pour les communications
- **FPGA** : *Field-Programmable Gate Array*, unités de circuit logique programmable

G

- **GPU** : *Graphical Processing Unit*, unité processeur graphique, composant matériel

I

- **IODA** : *Interaction-Oriented Design of Agent simulations*, modèle agent centré interactions
- **IoT** : *Internet of Things*, objets connectés
- **IP** : *Internet Protocol*, protocole Internet, protocole de communication réseau
- **IPC** : *Inter-Process Communication*, mécanisme de communication inter-processus

M

- **MERMAID** : *Managed Embedded Resources by Multi-Agents applied to Integrated and Distributed systems*, plateforme multi-agents embarquée présentée dans cette thèse
- **MTPS** : *Message Transport Protocol Service*, couche d'abstraction du routage des messages spécifiée par FIPA-ACL

P

- **PPV** : Plus Proche Voisin, méthode d'interpolation

R

- **RTT** : *Round-Trip Time*, temps moyen de transfert d'un échange de messages entre deux agents d'une plateforme, unité de mesure de la performance d'une plateforme multi-agents

S

- **SE** : Système Embarqué
- **SMA** : Système Multi-Agents
- **SoC** : *System-on-Chip*, système intégré sur une puce

T

- **TR** : Temps de Réponse, unité de mesure de la performance d'un programme

UNIVERSITE BRETAGNE LOIRE / EDUCATION LANGAGES, INTERACTIONS COGNITION, CLINIQUE

Le Mans Université

Titre : Intégration des systèmes multi-agents aux systèmes embarqués pour la délégation de tâches

Mots clés : Systèmes embarqués, Systèmes multi-agents, Délégation de tâches, Optimisation, Flexibilité, Plateforme multi-agents embarquée

Résumé : Cette thèse présente comment l'intégration de systèmes multi-agents au sein de systèmes embarqués peut permettre d'optimiser la gestion des tâches. Nous relevons un manque de flexibilité pour les systèmes embarqués et posons l'hypothèse d'une solution multi-agents permettant la prise en compte dynamique du contexte d'évolution du système. Les systèmes embarqués, intégrés à l'environnement utilisateur, sont contraints en termes d'espace physique et donc de ressources matérielles. Ces limites impliquent un besoin d'optimisation des ressources. Nous proposons d'expérimenter des algorithmes de négociation multi-agents pour déléguer des tâches entre les différentes ressources. Pour valider nos hypothèses, nous détaillons les caractéristiques des systèmes multi-agents, leurs comportements, leurs modèles, les plateformes sur lesquelles ils évoluent, leurs standards de communication et leurs algorithmes sociaux. Nous avons constaté que la majorité des travaux du domaine agent se concentraient sur d'autres problématiques. Par conséquent, nous avons proposé la formalisation de systèmes multi-agents embarqués et d'une plateforme multi-agents adaptée. Nous avons ensuite expérimenté cette plateforme au sein de systèmes embarqués avec le cas d'étude du traitement d'images, notamment avec le calcul d'une interpolation de pixels. Nous avons mené des études de performances pour estimer le coût administratif d'une solution multi-agents, puis avons considéré ces résultats au regard des gains de capacité de nos systèmes embarqués. Nos dernières expérimentations mettent à l'épreuve notre solution de délégation de tâches entre plusieurs cartes embarquées dans un contexte hétérogène.

Title : Multi-agents systems integration within embedded systems for tasks delegation

Keywords : Embedded systems, Multi-agents systems, Tasks delegation, Optimization, Flexibility, Embedded multi-agents platform

Abstract : This thesis shows how the integration of multi-agents systems within embedded systems can optimize tasks management. We notice a lack of flexibility for embedded systems and hypothesize that a multi-agents solution will allow the dynamic consideration of the system context of evolution. Embedded systems, being integrated into the user environment, are limited in terms of physical space and thus hardware resources. These limits involve the necessity to optimize the resources. We suggest experimenting multi-agents negotiation algorithms to delegate tasks between several resources. To validate our hypotheses, we detail the characteristics of multi-agents systems, their behavior, their models, the platforms on which they evolve, their communication standards and their social algorithms. We observed that the majority of the works of the multi-agents domain concentrated on other problems. Therefore, we proposed the formalization of embedded multi-agents systems and of an adapted multi-agents platform. We then experimented this platform within embedded systems with the case study of image processing, especially the calculation of a pixels interpolation. We led performance studies to estimate the administrative cost of a multi-agents solution, then considered these results in relation to the capacity earnings of our embedded systems. Our last experiments put to the test our solution of tasks delegation between several embedded cards within a heterogeneous context.