Modèle conceptuel uniforme pour l'adaptation des agents logiciels en environnement ambiant

Irène Velontrasina

UNIVERSITÉ DE LA RÉUNION

THÈSE DE DOCTORAT

Spécialité

INFORMATIQUE

présentée par

Irène VELONTRASINA

pour l'obtention du titre de

DOCTEURE DE L'UNIVERSITÉ DE LA RÉUNION

Modèle conceptuel uniforme pour l'adaptation des agents logiciels en environnement ambiant

---

Soutenance le 26 avril 2019 devant le jury composé de

Pr. Salima HASSAS	Rapporteur
Pr. Frédéric AMBLARD	Rapporteur
Pr. Lalaonirina RAKOTOMANANA	Examinateur
Dr. HDR. Zahia GUESSOUM	Examinateur
Dr. HDR. Jean-Chrystophe SOULIE	Examinateur
Dr. HDR. Noël CONRUYT	Examinateur
Pr. Rémy COURDIER	Directeur
Dr. Denis PAYET	Co-encadrant

Cette thèse a reçu le soutien financier de la Région Réunion et de l'Union Européenne – Fonds Européen de Développement Régional (FEDER), dans le cadre du Programme Opérationnel de Coopération Territoriale

#RÉGIONRÉUNION

Remerciements

Ma thèse est l'aboutissement d'un long parcours, riche en expériences scientifiques, intellectuelles, et même humaines. J'arrive au bout grâce à plusieurs catalyseurs à qui j'aimerais témoigner au début de ce manuscrit toute ma reconnaissance.

Merci à toi Rémy Courdier, de m'avoir dirigée dans cette thèse. Tu m'as accompagnée dans mon cursus universitaire, bien avant la thèse, depuis le jour où tu as participé à mon jury de sélection pour entrer en École d'Ingénieur, jusqu'au jour où tu participes à mon jury de thèse. Et pour bien d'autres choses encore, merci pour ton implication. Merci à toi Denis Payet, de m'avoir co-dirigée dans cette thèse. Tu as réussi à me convaincre de me lancer sur ce sujet. Tu m'as fortement soutenue dans la constitution du dossier pour la demande de financement. Et pour bien d'autres choses encore, merci pour ton implication

Mes remerciements vont vers vous, Mme Salima Hassas et M. Frédéric Amblard. Vous avez accepté de participer à mon jury de thèse en qualité de rapporteur. Je suis honorée par la qualité de votre rapport qui m'a conduit à la soutenance. Mes remerciements vont également vers vous mes examinateurs : M. Lalaonirina Rakotomanana, Mme Zahia Guessoum, M. Jean-Chrystophe Soulié, et M. Noël Conruyt. Je vous remercie pour ce regard critique que vous apportez à mes travaux de recherche.

Je remercie mon laboratoire LIM qui m'a accueillie depuis mon master. J'ai bénéficié de bonnes conditions de travail et une bonne ambiance de convivialité. Merci à mes collègues doctorants pour toutes ces échanges et longues discussions. Je remercie les membres de mon comité de suivi de thèse qui ont participé durant ces années à l'identification des grands axes de ma thèse. Merci d'avoir été favorables chaque année à ma réinscription.

Merci à l'ensemble de tous mes Amis qui m'ont soutenue chacun à sa manière. Une mention spéciale à ceux qui ont relu ce manuscrit, à ceux qui m'ont ouvert la porte de leur maison, particulièrement à celle chez qui j'ai eu une adresse, à ceux qui m'ont accompagnée dans les moments difficiles... Mes sincères remerciements à chacune, à chacun, de mes "*Amis, Vrais et Précieux*". J'espère qu'à travers ces mots vous vous reconnaissiez. À vos côtés, on dirait que tout m'est acquis : l'attention, le soutien, l'amitié.

En dernier, mais pas le moindre, merci à vous ma chère famille pour cet indéfectible soutien. Au milieu de vous était née mon envie de faire une thèse. Mes sincères reconnaissances à *mama* & *papa*, mes chers et tendres parents. Merci d'avoir toujours cru en moi, malgré le début de scolarité peu promettant de l'enfant turbulente que j'étais... que je suis.

Résumé

Ce travail s'intéresse à l'adaptation des systèmes informatiques opérant dans un environnement ambiant. Ce type de système est caractérisé par un ensemble de composants hétérogènes, distribués et connectés en réseaux. Ce cadre applicatif ambiant est fortement dynamique. A tout moment, des événements imprévisibles représentent une perturbation potentielle. Par conséquent, d'un côté, la cohésion de l'ensemble des activités collectives n'est pas toujours garantie. D'un autre côté, le fonctionnement normal n'est pas toujours maintenu lorsque les perturbations se présentent.

Face à ces problèmes, nous proposons un modèle conceptuel d'architecture pour l'adaptation de ces systèmes informatiques. Notre objectif est d'optimiser l'activité collective. Notre contribution principale est un pattern de conception d'architecture uniforme et générique appelé GMAS. Ce modèle est basé sur une approche orientée comportement. Sa particularité est la vérification de l'applicabilité d'un comportement avant son exécution. Les critères de vérification sont principalement l'état courant du cadre applicatif et celui des interfaces d'action. L'adaptation vient ensuite par ajustement des comportements futurs en fonction des retours de la vérification.

Le modèle d'architecture GMAS gère de manière indépendante les événements exceptionnels. Ce qui fait que les objectifs propres à l'agent ou au collectif n'ont pas à tenir compte de ces événements exceptionnels dans leur code source. La gestion des problèmes d'adaptation au niveau individuel et au niveau collectif est uniforme. Nous estimons qu'une adaptation collective passe par la prise en considération de l'individualité. Le but est d'avoir des agents flexibles une fois au niveau collectif. Pour cela, au niveau des composants individuels, l'instance de ce modèle constitue un modèle d'architecture interne. Au niveau de l'ensemble des composants, qui est le niveau global, GMAS offre un modèle de coordination de comportements collectifs basé sur la distribution des modalités opératoires. Avec ces trois propositions de modèle conceptuel : générique, pour le niveau local et pour le niveau global, nous évitons les situations néfastes lors de l'exécution. Nous implémentons chaque modèle sous forme de librairie Java pour montrer son efficacité.

Mots-clés : Flexibilité individuelle, Adaptation collective, Environnement Ambiant, Coordination de comportements, Modèle d'architecture

Abstract

This work deals with adaptation issues of computer systems that operate in an ambient environment. Such system is characterized by a set of heterogeneous components, which are distributed and connected to a network. The ambient environment is highly dynamic. Unpredictable events may result in a potential disturbance at any time. Therefore, on the one hand, the cohesion of their collective activities is not always guaranteed. On the other hand, the normal functioning is not always maintained when disturbances occur.

To cope with these issues, we propose a conceptual model of architecture that provides adaptation. The point is to optimize their collective activity. Our main contribution is a uniform and generic pattern. We call it GMAS. It relies on a behaviour-based approach. The special feature of GMAS is the checking of the applicability of a behaviour before its execution. Checking criteria are mainly the current state of both the external surroundings and the action interfaces. Future behaviour is adjusted according to checking outcome. This mechanism leads to adaptation.

The GMAS architecture model enables independent management of behaviours and exceptional events. As a result, the objectives of the agent or those of the collective do not have to take into account these exceptional events in their source code. This model uniformly manages adaptation issues of both the individual level and the collective level. We argue that collective adaptation comprises the consideration of the individuality. We aim to make agent flexible once it is at a collective level. To this end, we instantiate GMAS to design an agent internal architecture for the individual level. For the collective level, we instantiate GMAS as a coordination model for collective behaviours. This instantiated model is based on the distribution of the operating modalities. With these three conceptual models : generic, for individual level and for the collective level, we prevent harmful situations during the execution. We implement each model as a Java library to validate its effectiveness.

Keywords : Individual Flexibility, Applicability Checking, Collective Adaptation, Ambient Environment, Uniformal Pattern

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction générale

1.1 Contexte

Jusqu'à très récemment, à quelques exceptions près, la seule vocation de nos ordinateurs était restreinte aux traitements de flux d'informations fournies pour l'essentiel par l'homme. Cette restriction est en train de voler en éclat. Les ordinateurs sont aujourd'hui reliés à de plus en plus de dispositifs miniaturisés. Ces dispositifs permettent de capter des informations de façon plus autonome. Ils dotent également les ordinateurs de capacité d'actions physiques dans l'environnement. Les capacités des ordinateurs sont ainsi étendues et leurs formes changent également. En effet, ils se disséminent sur chacun des maillons de la chaîne qui les relient aux dispositifs de captation et d'action. Avec cette tendance, les technologies d'information et de communication intègrent de plus en plus l'environnement quotidien, que nous appelons "environnement ambiant". La frontière entre la robotique et l'informatique s'estompe. Les ordinateurs avec leurs nouvelles formes variées ont ainsi vocation à être omniprésents et accessibles à tout moment. Ils deviennent invisibles en laissant la place à une interface physique unifiée [Fer09]. L'environnement ambiant devient instrumenté. De nouvelles formes de système informatique émergent de cette tendance : l'informatique diffuse, l'informatique ubiquitaire ou l'internet des objets. Ces systèmes informatiques pilotent un ensemble d'objets connectés, distribués. Les composants de cet ensemble interagissent entre eux pour réaliser une activité collective. Toutefois, l'environnement ambiant est hétérogène et dynamique. Cela implique que l'activité collective est sujette à des perturbations pouvant conduire à des situations néfastes tant pour le système que pour son utilisateur. Dans ce contexte, notre objectif est d'optimiser l'activité collective en évitant ces situations néfastes. Pour cela, nous proposons un modèle conceptuel d'architecture logicielle qui tient compte des propriétés du cadre applicatif. Nous analysons plus en détail ces propriétés dans les sous-sections 1.1.1 et 1.1.2 suivantes.

1.1.1 Environnement hétérogène

D'un point de vue interne, nous venons de voir que le système informatique, dont l'environnement est instrumenté, est constitué par des composants miniaturisés, connectés en réseau. L'infrastructure résultante est décentralisée. La prise de décision dans un tel système est par conséquent distribuée entre les unités qui le composent. L'interaction entre les unités joue un rôle essentiel pour mettre en relation chacun des éléments [CH15].

Les composants sont très hétérogènes. Ils sont de natures diverses : capteur, périphérique mobile, unité de calcul. Ils utilisent des technologies différentes comme le wifi ou le bluetooth. Ils proviennent également de constructeurs différents. Leurs manières de communiquer sont par conséquent différentes. La diversité au niveau des interactions se traduit par la différence de réaction par rapport à un même stimulus. L'hétérogénéité des composants complexifie les inter-liaisons au point de limiter leurs capacités d'interaction. Cela implique qu'à tout moment, et pour une durée plus ou moins importante, tout ce qui est produit par un composant (données ou services) reste inaccessible et inutilisable par les autres. Cependant, l'une des propriétés recherchées dans ces systèmes est que l'interaction entre les composants puisse conduire à la réalisation d'une activité collective. L'objectif étant que le potentiel du collectif soit supérieur à la somme de chacun des comportements individualisés des composants [SM09] [ABE+13].

1.1.2 Environnement dynamique

D'un point de vue externe, l'environnement ambiant est dynamique et ouvert. Les perturbations apparaissent de manière imprévisible. Nous pouvons citer par exemple, l'insuffisance de ressources, la perte d'information, les mauvaises manipulations de la part de l'utilisateur, l'interruption de connexion. Ces changements impactent sur l'interaction entre les composants. Ces événements imprévisibles occasionnent facilement des situations néfastes comme l'endommagement des équipements ou le blocage de l'ensemble du système. Ce qui limite encore une fois leur efficience collective. Le risque est d'autant plus élevé si le système inclut des humains comme les systèmes d'assistance de personnes à mobilité réduite. Enfin, les utilisateurs constituent un autre facteur de cette dynamicité de l'environnement ambiant. En effet, ils ont souvent besoin de services personnalisés. Le système est donc amené à évoluer en fonction des exigences des utilisateurs.

Ces systèmes sont aussi "situés" [ZOA+15], [CH15]. Cela signifie qu'ils possèdent une référence spatiale et/ou une référence temporelle. La référence spatiale se traduit par l'interaction avec un environnement ambiant, ou dans certains cas par l'interaction avec des structures sociales (système de surveillance, ou d'assistance). La référence temporelle renvoie

à l'évolution dans le temps. En effet, à tout moment de nouveaux composants peuvent être ajoutés ou supprimés en fonction des besoins.

1.2 Problématiques

Les deux propriétés de l'environnement influent fortement sur le bon déroulement de l'activité collective. Toutefois, il est difficile d'anticiper les éventuels changements du cadre opératoire au moment de la conception du logiciel. D'abord, parce qu'il n'est pas toujours possible d'identifier de manière exhaustive les différents cas d'événements inattendus. Ensuite, même dans les cas où c'est possible, le système devient rapidement complexe en tenant compte de toutes ces contraintes. Ce problème conduit à deux principaux verrous qu'il faut lever pour concevoir une architecture logicielle adéquate.

— Le premier verrou résulte du caractère hétérogène de l'environnement. Il s'agit de l'incapacité du système à garantir la cohésion de l'ensemble des activités collectives. La solution consiste à favoriser l'interaction et l'interopérabilité entre chacun des composants. Nous avons donc ici une première exigence à gérer qui est l'aspect collectif.

— Le deuxième verrou résulte du caractère dynamique de l'environnement. Il s'agit de l'incapacité du système à maintenir un fonctionnement normal quel que soit le changement du contexte opératoire. La solution consiste à ajuster les réponses du système pour qu'elles correspondent aux conditions extérieures.

En analysant ces deux verrous, nous remarquons qu'ils constituent finalement un seul problème qui est l'adaptation. En effet, pour le premier nous remarquons que l'interaction entre les composants hétérogènes est une forme de dynamique interne à laquelle le système doit s'adapter. Pour le deuxième, les événements imprévisibles présentent une dynamique externe à laquelle le système doit également s'adapter. L'adaptation constitue de cette manière le point essentiel de cette thèse. C'est une propriété primordiale pour être efficace dans un environnement hors de contrôle. Notre objectif dans cette thèse est donc de proposer un modèle conceptuel d'architecture permettant de lever ces verrous.

1.3 Contributions

1.3.1 Choix de l'approche

Nous identifions deux manières différentes de considérer l'adaptation dans la littérature. Dans la première, l'adaptation est vue comme un processus individuel (micro) propre à chaque composant. Dans la deuxième, elle est vue comme un processus collectif résultant de

l'organisation de l'ensemble des composants (macro).

Lorsque l'adaptation est vue comme un processus individuel, cela revient à doter chaque composant d'une capacité d'adaptation propre. Certains Systèmes Auto-adaptatifs [KRV+15] utilisent par exemple des méthodes de rétroaction (*Feedback control loop*) ou de prise de décision (*Decision-making*) qui requièrent généralement un raisonnement. D'autres approches comme celles de l'Autonomic Computing [DDF+06] [KC03] utilisent des techniques d'apprentissage pour détecter et corriger des problèmes : méthodes stochastiques, réseau de neurone, théorie de jeux, inférence ou encore théorie de contrôle. Ces méthodes apportent une forme d'intelligence et demandent un certain niveau de puissance de calcul.

Lorsque l'adaptation est vue comme un processus collectif on s'intéresse à l'organisation [PHBG09]. Les composants n'ont pas nécessairement une capacité d'adaptation individuelle. Ils interagissent et parviennent à un phénomène global comme l'auto-organisation, la réorganisation ou encore l'émergence.

En considérant l'approche micro, nous réalisons qu'elle est peu compatible avec notre cadre applicatif. Ce dernier est constitué de composants miniatures avec des capacités de calcul et de stockage limitées. Par exemple, le microprocesseur embarqué dans un Arduino possède une certaine puissance de calcul. Cependant, il n'est pas possible d'y faire tourner un algorithme d'apprentissage. De même, les autres composants comme les montres, les bracelets connectés ne peuvent pas stocker la quantité d'informations nécessaires pour l'apprentissage. Par conséquent, ces approches micro ne conviennent pas à notre cadre applicatif où il est impossible d'avoir un composant doté d'une capacité cognitive. Nous nous intéressons alors à l'approche macro où l'adaptation globale ne nécessite pas des composants puissants en termes de calcul.

1.3.2 Flexibilité du niveau individuel

Nous avons dit que dans ce travail nous sommes dans une démarche d'optimisation de l'activité collective. Dans ce sens, nous souhaitons tirer profit de tous les potentiels existants pour parvenir à cet objectif. Au niveau individuel, nous avons certes des composants non cognitifs mais qui restent cependant réactifs. Ils réagissent en fonction de leur perception. Nous estimons que cette réactivité des composants individuels est un potentiel exploitable par rapport à l'objectif. Nous partons de l'idée qu'il est nécessaire d'instaurer une équilibre entre l'individuel et le collectif. Nous proposons dans ce sens une flexibilité dans la manière dont le composant individuel agit avec son environnement. Cette flexibilité est une forme d'adaptation dans la mesure où elle permet à l'individu de faire face aux événements imprévus du cadre applicatif. Cependant, comme nous ne pouvons pas utiliser ici les méthodes

d'apprentissage et les raisonnements propres à l'adaptation individuelle, il nous est nécessaire d'utiliser la notion de flexibilité. Ce terme est défini comme la capacité d'un composant individuel à ajuster sa réaction dans l'environnement en fonction de ses perceptions, mais non pas en fonction de son raisonnement interne. Nous consacrons donc une partie de ce travail à cette flexibilité du niveau individuel. C'est une fois que nous avons géré ce niveau que nous traitons ensuite le niveau collectif.

Nous proposons donc une solution qui couple l'ensemble de ces deux approches micro et macro pour tirer le meilleur de chacune d'elle. Le but est de parvenir à une activité collective adaptable. Dans notre approche, la flexibilité de l'individuel face à la dynamique de l'environnement est un levier nécessaire pour une adaptation globale. L'organisation du collectif est basée sur l'interaction entre les composants. Le collectif a donc un fort impact sur les individualités. Il nous faut donc une solution uniforme pour tenir compte de cette corrélation entre les deux niveaux. De cette manière, nous visons à faire émerger une adaptation collective à partir de la flexibilité de chacun des composants. Cela revient à une forme d'intelligence collective issues d'un ensemble de composants réactifs.

1.3.3 Synthèse de la contribution

Les solutions proposées dans la littérature gèrent séparément les deux aspects micro et macro. Dans cette thèse nous proposons une solution uniforme pour ces deux niveaux : flexibilité au niveau micro, et adaptabilité au niveau macro. La contribution consiste en un modèle conceptuel d'architecture pour une gestion uniforme de la flexibilité individuelle et de l'adaptabilité collective. Le but est de retrouver les mêmes principes au niveau local, le micro, qu'au niveau global, le macro. Ce modèle conceptuel est basé sur une approche orientée comportements. Dans un premier temps, nous concevons un modèle générique. Nous l'instancions ensuite de manière différente dans chacun des deux niveaux en fonction de leurs exigences propres.

Le point essentiel de cette architecture réside dans une unité de gestion des événements exceptionnels similaire à la gestion des exceptions dans la Programmation Orientée Objet. L'objectif est d'éviter que ces événements ne perturbent la cohésion de l'activité collective. Cette unité est appelée *Contrôle d'Applicabilité*. Son rôle est de juger en fonction de l'état de l'environnement si une décision interne peut être transformée en une action concrète dans l'environnement ou non. Avec les instances de ce modèle nous avons à la fois une architecture interne de gestion de comportements pour l'individuel et une architecture globale de coordination des comportements pour le collectif. La gestion des comportements au niveau individuel s'intéresse plus à la problématique du maintien d'un fonctionnement normal par rapport aux contraintes locales. Ce qui correspond à la solution au deuxième

verrou dans 1.2. La coordination des comportements pour le collectif résout le problème de cohésion entre les composants. Ce qui correspond à la solution au premier verrou dans 1.2. Les grandes lignes de cette contributions sont décrites comme suit :

— Un pattern générique d'architecture pour la gestion des événements exceptionnels dans les deux niveaux local et global (Chapitre 3).

— Un modèle de flexibilité pour le niveau local pour optimiser l'efficacité au niveau collectif (Chapitre 4).

— Un modèle d'adaptabilité pour la coordination du niveau collectif (Chapitre 5).

— Une proposition d'implémentation de notre modèle sous forme de librairie java (Chapitre 6)

1.4 Plan de la thèse

Nous développons cette contribution dans les chapitres suivants. Ce manuscrit est structuré comme suit :

Le chapitre 2 présente les problématiques principales de cette thèse. Il s'agit des problèmes d'adaptation d'un système informatique évoluant dans un environnement ambiant. Nous présentons un état de l'art des différents travaux traitant ce problème. L'objectif est de positionner notre contribution par rapport aux travaux existants.

Le chapitre 3 introduit notre solution aux problèmes vus dans le chapitre précédent. Il s'agit d'un modèle d'architecture uniforme qui constitue notre contribution principale. Nous présentons dans ce chapitre ses bases théoriques et son modèle formel. A un certain niveau d'abstraction les problèmes d'adaptation aux deux niveaux local et global présentent une similarité. L'objectif est donc de proposer une solution uniforme pour ces deux niveaux. Les deux chapitres qui suivent présentent ensuite l'instanciation de ce modèle dans chacun des deux niveaux. Cette instanciation est particulière pour chaque niveau. Elle prend en compte les contraintes propres à chaque niveau.

Le chapitre 4 décrit l'instanciation de notre modèle conceptuel pour la flexibilité du niveau local. Il s'agit d'un modèle d'architecture interne appelé MECA. Ce modèle est caractérisé par la réutilisation du Modèle Influence/Réaction. Nous présentons dans ce chapitre les répercussions du caractère dynamique et hétérogène de l'environnement au niveau local. Ensuite nous détaillons la spécialisation correspondante du modèle uniforme. L'objectif est de montrer que la flexibilité individuelle est un levier pour l'adaptation globale du système.

Le chapitre 5 décrit l'instanciation de notre modèle conceptuel pour l'adaptation au niveau global. Il s'agit d'un modèle de coordination des comportements dans une activité collective. Ce modèle est caractérisé par des méthodes de suivi et d'ajustement des activités collectives. Nous présentons d'abord dans ce chapitre le passage du niveau local au niveau global. Nous proposons dans ce contexte une modélisation des comportements. Ensuite comme dans le chapitre précédent nous analysons les caractéristiques de l'environnement ambiant vues au niveau global. L'objectif est de montrer l'efficacité du modèle pour la cohésion des activités collectives.

Le chapitre 6 présente les aspects techniques de notre contribution. Ce chapitre commence par l'expérimentation de notre architecture interne d'agent avec les librairies Java correspondantes. Ensuite, pour le niveau global nous donnons une présentation de la plateforme *Ubiquity*. C'est une plateforme d'interaction sociale sur laquelle s'appuie notre proposition. Nous y présentons les expérimentations que nous avons menées sur des activités collectives.

Nous terminons ce manuscrit avec la conclusion dans le chapitre 7. Nous y revenons sur les grandes lignes de la contribution de la thèse ainsi que les différentes perspectives de nos travaux.

Chapitre 2

Adaptation dans un environnement ambiant

Ce chapitre présente les problématiques principales de cette thèse. Il s'agit des problèmes d'adaptation d'un système informatique évoluant dans un environnement ambiant. Nous présentons un état de l'art des différents travaux traitant ces problèmes. L'objectif de ce chapitre est d'identifier les différents verrous relatifs à notre contexte et de définir les méthodes et les outils pour les lever. Ce chapitre est organisé comme suit. La section 2.1 donne une vue générale des solutions d'adaptation à travers les systèmes dits "auto-adaptatifs". Nous analysons ensuite particulièrement l'adaptation dans les Systèmes Multi-Agents dans la section 2.2. Cela nous mène aux méthodes et outils utilisés dans ce domaine et qui sont pertinents pour notre solution. La section 2.3 se focalise sur l'approche orientée comportement résultant de l'analyse précédente. Elle constitue notre approche principale. Nous terminons ce chapitre avec la section 2.4 sur une synthèse qui résume le positionnement de notre contribution par rapport aux autres travaux.

2.1 Les systèmes auto-adaptatifs

Nous analysons dans cette section le problème d'adaptation dans les systèmes auto-adaptatifs. Après une description du problème, nous aborderons les différentes approches pour identifier celles pouvant répondre à notre besoin.

2.1.1 Description du problème d'adaptation

Avec l'avènement des objets connectés, la démocratisation des unités de traitements miniaturisées, et la performance de la technologie sans fil, l'informatique se dilue de plus en plus dans la vie quotidienne des utilisateurs. De ce fait, les systèmes informatiques qui y résultent doivent présenter une forte disponibilité pour être accessibles à tout moment

et à tout endroit. En outre, nous avons vu dans la section 1.2 précédente que l'environnement ambiant est hétérogène et fortement dynamique. L'adaptation est définie dans [Pic14] comme une propriété que l'on fournit aux systèmes afin de répondre à une dynamique endogène et exogène. Dans notre travail, la dynamique endogène correspond à l'interaction entre les composants hétérogènes qui forment le collectif. La dynamique exogène correspond à l'interaction avec l'environnement ambiant. La plupart des travaux sur les systèmes pervasifs s'intéresse plus particulièrement à la notion d'auto-organisation (*self-organisation*). La motivation de cet aspect "auto" résulte de la criticité de l'environnement ambiant. En effet, le système doit être en mesure d'ajuster son comportement de manière autonome. Cela signifie que les interventions de l'utilisateur en matière d'ajustement du système doivent être minimales ou idéalement nulles [MHdTH13].

2.1.2 Les différentes approches

Le cycle MAPE

L'adaptation constitue une problématique commune à différents domaines de recherche. Les travaux menés dans l'*Autonomic Computing* (encore appelé *Informatique autonome*) [KC03] [DDF+06] définissent quatre fonctions principales du processus d'adaptation appelées MAPE qui forment un cycle. Ces fonctions sont utilisées pour la détection et la correction des erreurs rencontrées lors de l'exécution. Le but étant de réduire l'intervention humaine. Les fonctions du cycle MAPE sont reprises en intégralité ou en partie dans la plupart des systèmes auto-adaptatifs [MHdTH13] [KRV+15] [BSG+09]. Ces fonctions sont :

— ***Monitoring*** : pour la collecte des données nécessaires issues de l'environnement, de l'utilisateur ou des capteurs.

— ***Analysing*** : pour le traitement des données dans le but de tirer une information pertinente.

— ***Planning*** : pour la prise de décision par rapport à l'information issue de l'*Analysing*.

— ***Executing*** : pour la concrétisation des actions dans l'environnement.

Chacune de ces fonctions utilise des méthodes et outils plus ou moins complexes en fonction du niveau d'adaptation attendu. Les fonctions les plus critiques sont l'*Analysing*, avec des méthodes comme la théorie des jeux, l'inférence,... et le *Planning* avec des méthodes stochastiques, basés sur les réseaux de neurone.

Catégorisation des approches

Du côté des systèmes auto-adaptatifs les approches relatives à l'adaptation sont classées suivant différents paramètres. Nous avons identifié deux types de catégorisation. La

première est une catégorisation suivant le niveau d'abstraction. La deuxième est une catégorisation suivant les thématiques utilisées.

La première catégorisation est donnée par [MHdTH13]. Elle représente trois niveaux d'abstraction. Du plus abstrait au plus concret, on distingue : les approches principales, ensuite les outils et méthodes globales, et enfin les outils et méthodes spécifiques. Nous citons ici les différents types d'approches, et de méthodes proposées pour chaque niveau. Nous les analysons ensuite avec celles de la deuxième catégorie. Les approches principales présentées dans la première catégorisation sont : *External control mechanisms*, *Component-Based Software Engineering (CSBE)*, *Model-driven*, *Nature-inspired engineering*, *Multiagent systems*, *Feedback systems*. Les outils et méthodes globales proposés sont : *models*, *simulation*, *architecture* et *frameworks*. Les outils et méthodes spécifiques sont le *Feedback control loops* ou boucle de retroaction, le *Decision-making* ou prise de décision, et le *Requirements engineering*.

La deuxième catégorisation assez similaire mais plus enrichie est donnée dans [KRV+15]. Cet article présente également une taxonomie pour l'auto-adaptation. Contrairement au travail dans [MHdTH13] où il y a trois niveaux d'approches, les approches présentées ici ont les mêmes niveaux. Ces approches sont : *Model-based approaches*, *Architecture-based approaches*, *Reflexion approaches*, *Programming paradigms*, *Control theory*, *Service-oriented approaches*, *Agent-based approaches*, *Nature-inspired approaches*, *Formal modeling and Verification approaches*, *Learning approaches*, et le *Requirements-oriented approaches*.

Analysons maintenant ces différentes approches, méthodes et outils pour identifier ceux qui nous conviennent au mieux pour chacune de ces catégories. Les approches *External control mechanisms* et l'*Architecture-based approaches* présentent une forte similarité. En effet, dans ces approches, le contrôle de l'adaptation est externalisé et parfois centralisé. Les approches, *model-based approach* et *model-driven* utilisent des modèles pour représenter l'environnement et le logiciel. Les modèles servent alors de référence pour ajuster le système en fonction des changements. L'approche *CSBE* facilite l'intégration automatique des composants qui s'ajoutent au cours de l'exécution. Les approches *Formal modeling and Verification approaches* se focalisent davantage sur l'intégrité et la sécurité des systèmes critiques. Le contrôle de l'adaptation est externalisé et pour la plupart centralisé. Cette approche fait partie des *Programming paradigms* qui sont des approches non spécifiques aux systèmes auto-adaptatifs.

Vers une approche SMA

Ces approches ont montré leur efficacité dans de nombreux travaux. Leur efficacité réside dans la performance de la couche logique de l'adaptation. Néanmoins, elles ne tiennent pas compte forcément de la dimension collective du système. Par conséquent, celles dont le contrôle de l'adaptation est externalisé ne nous conviennent pas. Dans ce contexte la

prise de décision n'est pas distribuée entre les composants. De même pour les autres approches, même si le contrôle de l'adaptation n'est pas externalisé, l'interaction entre les composants n'intervient pas dans le processus d'adaptation. Ce qui fait que la dimension collective n'est pas exploitée. Dans cette thèse, cette dimension collective constitue une des critères de choix de notre approche. Pour être conforme à ce critère, il ne nous reste donc plus que le *Nature inspired approach* et l'*Agent based-modeling* dont le principe est issu des SMA.

La *Nature inspired approach* s'inspire des phénomènes biologiques, physiques ou chimiques. C'est une approche décentralisée qui favorise l'interaction entre les composants afin de parvenir à une auto-organisation et même à une émergence. Cette propriété nous intéresse parce que cela implique que l'on n'a pas besoin de composants dotés de grande puissance. Les *Nature inspired approach*, en particulier les approches biologiques sont initialement issues des travaux menés sur les comportements collectifs dans les SMA. Beaucoup de recherches en SMA utilisent alors ces approches : [WBH08] utilise un champ de potentiel pour une allocation de tâche et [ZOA+15] utilise une coordination physique et biochimique pour concevoir une architecture appelée SAPERE. Les principes de cette approche correspondent donc à notre attente. Cependant, sa forte similarité avec les principes multi-agents nous conduit directement à l'approche SMA. D'autant plus que les deux travaux [MHdTH13] et [KRV+15] montrent chacun l'efficacité des SMA dans le développement des systèmes auto-adaptatifs.

2.1.3 Approches retenues

Nous choisissons maintenant parmi ces différentes approches celles qui correspondent à notre besoin.

Correction et prévention des erreurs

Le modèle MAPE a pour but de détecter les erreurs et de les corriger ensuite. Ce modèle a montré son efficacité dans l'auto-configuration des équipements industriels. Dans ce contexte, le but n'est pas d'éviter les problèmes ou les erreurs mais de trouver une solution lorsqu'ils se produisent. Toutefois, dans certains cas d'utilisation il est préférable d'éviter les problèmes que de les corriger. Les composants en contact direct avec les utilisateurs font partie de ces cas d'utilisation.

Considérons par exemple les batteries des smartphones. Ce sont des accessoires pouvant occasionner des explosions plus ou moins dangereuses lorsqu'elles se trouvent dans des conditions défavorables. En janvier 2018, deux explosions de batterie de iPhone suite à une surchauffe ont lieu dans les Apple Store à Zurich et à Valence[1]. Ces explosions ont mis

1. Référence ici

directement en danger les utilisateurs. Il en est de même pour les explosions de la batterie du Samsung Galaxy Note 7 en 2016. Explosions qui ont même conduit au retrait de l'appareil du marché [2]. Les utilisateurs sont donc exposés à un risque d'explosion irréparable. Outre les préjudices sur la marque, ces types d'explosions sont très coûteux. Pour Samsung, le coût estimé est au voisinage de 17 milliards de dollars [3].

Ces deux exemples nous montrent que lorsque les éventuelles erreurs présentent un grand risque, il est préférable d'éviter les problèmes plutôt que de les corriger. Nous avons ici deux exemples de mesures de sécurité utilisée dans l'environnement ambiant :

— La mise à jour d'un iPad ne peut avoir lieu que lorsque l'appareil est branché à l'alimentation [4].

— Le chargement de la batterie d'un iPhone est limité automatiquement à 80% lorsque la température de la batterie augmente et atteint un certain seuil. Le chargement reprend de nouveau uniquement quand la température baisse. [5]

Dans le premier exemple, l'objectif est de s'assurer d'avoir suffisamment de batterie pour ne pas perturber la mise à jour. Cette dernière n'aura pas lieu tant que l'appareil n'est pas branché même si le niveau d'énergie est à 100%. Dans le deuxième exemple, on vise plutôt la durée de vie de la batterie et la sécurité des utilisateurs. Une température élevée favorise l'obsolescence de la batterie, mais peut provoquer surtout une explosion.

Notre cadre applicatif est très dynamique et interagit directement avec le monde physique réel. Nous parvenons à la conclusion qu'il est préférable dans notre cas de favoriser la prévention que la correction des erreurs ou des problèmes. De ce fait, l'adaptation que nous souhaitons mettre en place ne priorise pas la résolution d'un problème qui se produit. Son but est plutôt de limiter au maximum les situations néfastes tant pour les équipements que pour l'environnement autour.

Cahier des charges des approches retenues

La catégorisation à trois niveaux dans [MHdTH13] nous intéresse dans la mesure où elle nous donne des repères pour dresser notre proposition de l'approche principale à l'implémentation. Nous allons donc identifier les différentes approches et outils correspondant à chacun des trois niveaux.

Pour le choix de l'approche principale, compte tenu de ce que nous avons vu, nous choisissons donc l'approche SMA. De plus [Wey10] montre que les SMA sont compatibles

2. Référence ici
3. Référence ici
4. https://support.apple.com/fr-fr/HT204204
5. https://support.apple.com/fr-ci/HT201569

Approche Principale	Outil global	Méthodes spécifiques
Approche SMA	Architecture	Mécanisme de contrôle par rétroaction Mécanisme de prise de décision

TABLE 2.1 Synthèse des approches retenues par rapport aux paramètres donnés dans [MHdTH13]

avec les systèmes décentralisés, assez flexibles aux conditions dynamiques et supportent l'ajout et la suppression des composants. Ce qui correspond à notre besoin.

Parmi les méthodes globales nous choisissons un modèle d'architecture. Une architecture donne une vue complète de l'ensemble du système.

Le choix de notre méthode spécifique résulte des caractéristiques de notre approche principale SMA. Si on se réfère aux fonctions principales du cycle MAPE, [KRV+15] montre que l'approche multi-agents est plus focalisée sur la fonction *Planning*. De ce fait, pour compléter cette fonctionnalité, nous choisissons le *feedback control* comme outil spécifique. Toutes les fonctions du MAPE y sont prises en compte. En outre, nous rajoutons également les mécanismes de prise de décision pour assurer la fonction d'analyse. Nous obtenons donc :

— *Le contrôle par rétroaction* (*feedback control loop*) : avec lequel le système analyse les informations venant de l'environnement, contrôle le processus en cours, et ajuste les futures décisions d'action qui vont être exécutées au niveau du code opératoire [BSG+09].

— *La prise de décision* (*decision-making*) [ST09], qui procède à la sélection de l'action la plus appropriée parmi un ensemble d'actions prédéfinies. Les méthodes de *decision-theoritic planning*, et le *Markov Decision Process* sont celles des plus courantes.

Nous avons maintenant les éléments complets pour construire notre proposition. Le tableau 2.1 nous résume le choix que nous avons fait par rapport à chaque niveau.

2.2 Adaptation dans les SMA

Nous analysons dans cette section les techniques utilisées dans les SMA pour concevoir des systèmes adaptatifs. L'adaptation y est aussi utilisée pour faire face aux changements. L'origine de ces changements peut être le système lui-même. Les composants des SMA sont généralement hétérogènes et leurs interactions conduisent à une forte dynamique interne. Les perturbations proviennent aussi de l'environnement et sont plus fréquentes dans un cadre opératoire non simulé. Les travaux de recherche sur l'adaptation considèrent celle-ci soit au niveau macro ; soit au niveau micro ; soit de façon transversale par une analyse globale à partir de laquelle les décisions locales seront déduites. Le niveau macro correspond à l'adaptation collective qui est associée à l'organisation. Le niveau micro concerne l'adaptation

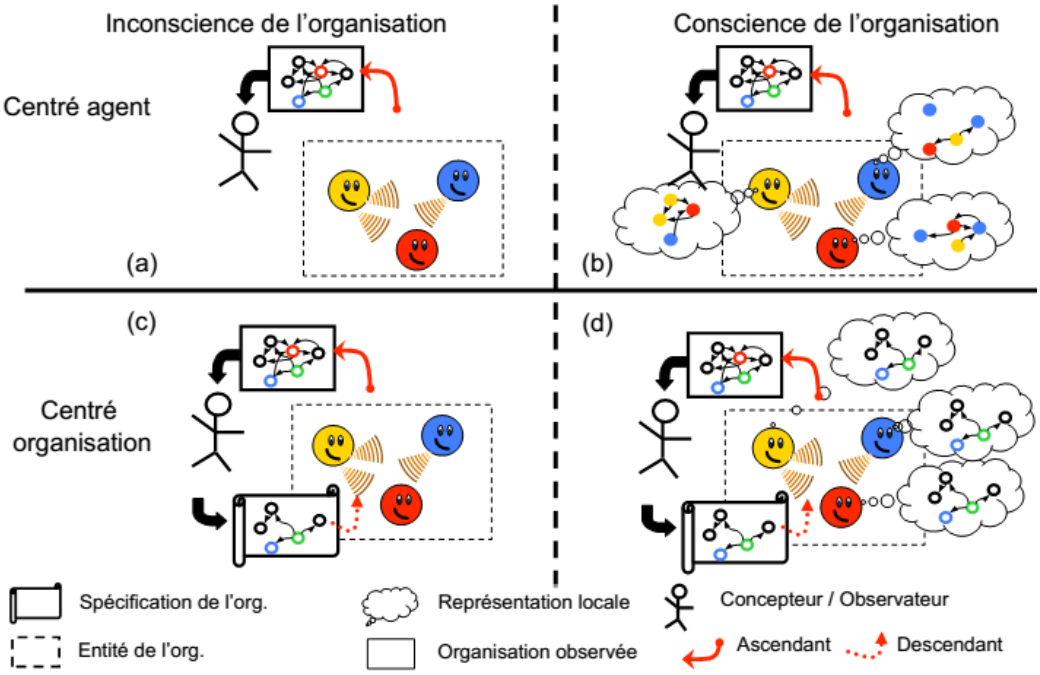


FIGURE 2.1 Vue synthétique de l'organisation [PHBG09] : (a) SMA émergents ; (b) SMA basés sur les coalitions ; (c) Ingénierie orientée agent ; (d) SMA orientés organisation.

au sein d'un agent. Les approches transversales sont celles qui caractérisent les systèmes auto-adaptatifs vues dans la section 2.1.2. Pour la suite, nous examinons d'abord l'aspect macro avec l'adaptation globale ensuite l'aspect micro avec l'adaptation locale.

2.2.1 Adaptation et organisation

Au niveau macro, l'adaptation est vue comme un processus collectif qui s'articule autour de l'organisation. Les agents font partie d'une organisation. Ils interagissent et parviennent à un phénomène global comme l'auto-organisation, la ré-organisation ou l'émergence. C'est le cas par exemple de l'approche *Adaptive Multi-Agent System* (AMAS) proposée dans [BCGP03]. Les travaux dans [PHBG09] définissent 4 types d'organisation dans les SMA présentés par la figure 2.1. Ces types d'organisation sont définis en fonction de l'approche utilisée qui peut être centrée agent (première ligne) ou centrée organisation (deuxième ligne), et du caractère implicite ou explicite de l'organisation : inconscience de l'organisation (première colonne) ou conscience de l'organisation (deuxième colonne).

Les travaux dans [Pic14] donnent plus de description sur ces types d'organisation. L'approche centrée agent encore appelée ACPV (*Agent Centred Point of View*) est une approche ascendante qui met l'agent au centre de l'organisation. L'organisation est alors le résultat de la coopération entre les agents. L'approche centrée organisation encore appelée OCPV (*Organisation Centred Point of View*) est descendante. Elle contraint les agents à suivre une

organisation qui est déjà définie au préalable. Les agents ne connaissent rien de l'organisation quand c'est implicite. Ils ont une représentation locale de leur organisation quand c'est explicite. La combinaison de chacun de ces paramètres nous donne les 4 types de travaux SMA notés (a), (b), (c), (d) dans la figure 2.1.

— ***Les SMA à organisation émergente*** (a) où les agents s'auto-organisent ou communiquent de manière indirecte via l'environnement.

— ***Les SMA basés sur les coalitions*** (b) où les agents peuvent construire des modèles propres des interactions avec le voisinage.

— ***Ingénierie orientée agent*** (c) où les organisations sont spécifiées avant l'implémentation des agents.

— ***SMA orientés organisation*** (d), les agents ont une représentation de l'organisation et peuvent potentiellement la modifier.

Parmi ces approches, nous nous intéressons particulièrement aux *SMA basés sur les coalitions* (b). D'abord, notre cadre applicatif principal présente un fort caractère dynamique. Les approches qui utilisent une définition préalable de l'organisation ne nous conviennent donc pas. Ce qui justifie notre choix pour l'approche ACPV. Ensuite, nous considérons que pour collaborer plus efficacement, les agents ont besoin d'une représentation locale de son voisinage. La conscience de l'organisation met en relation les agents pour qu'ils puissent échanger facilement. Ce qui justifie notre choix pour l'organisation explicite.

Ces approches sur l'organisation ont été utilisées dans de nombreux travaux SMA, en particulier dans la résolution de problème. Toutefois, ces approches ne tiennent pas en compte la capacité d'adaptation au niveau d'un agent. Soit, elles considèrent que cette adaptation individuelle n'est pas pertinente pour l'adaptation de l'ensemble. Soit, elles partent du principe que par définition, les agents sont adaptatifs. Ce qui signifie que la capacité d'adaptation est inhérente à l'agent. Cependant un agent n'est pas systématiquement adaptatif. Nous allons décrire davantage cette adaptation individuelle dans la section 2.2.2 suivante.

2.2.2 Adaptation au niveau de l'agent

Pour l'approche micro, l'adaptation est vue comme un processus individuel intégré dans l'agent. Ce processus est lié à l'architecture interne de l'agent. Les travaux dans [WJ94] donnent trois groupes classiques d'architecture interne d'agent définis suivant le niveau de raisonnement de l'agent : architecture d'agent délibératif, architecture d'agent réactif, et architecture d'agent hybride.

Architecture d'agent délibératif

L'agent est conçu avec des représentations symboliques de son environnement. La prise de décision est basée sur un raisonnement logique. Doté de ces propriétés, l'agent est capable

de planifier ses actions. Dans une telle architecture, l'agent possède des buts bien définis et la planification de ses actions tend vers la réalisation de son objectif. Généralement, les agents utilisent ici des techniques d'apprentissage pour interagir avec leur environnement, et pour ajuster ensuite leurs comportements. Par exemple, l'apprentissage par renforcement dans [WU05], le réseau de neurone dans [Che11]. Plus elles seront confrontées à des variations du contexte opérationnel, plus ces méthodes permettront à l'agent d'affiner ses comportements et donc de s'adapter. L'architecture BDI (Belief-Desired-Intention) est un exemple d'architecture d'agent délibératif très utilisée dans différents domaines comme la robotique [GTC+10], les réseaux de capteurs [CGPM11], ou pour un simulateur de comportement d'agriculteur comme dans [TTG12].

Architecture d'agent réactif

L'agent n'a pas la même capacité cognitive qu'un agent délibératif. Ici l'agent n'a ni représentation symbolique ni raisonnement logique. Le comportement intelligent de l'ensemble est le résultat de l'interaction entre les agents, et cette interaction peut conduire à des phénomènes d'émergence. L'exemple le plus courant de ce type d'architecture est l'architecture de subsomption de Brooks [Pri02]. Dans ce modèle, les comportements de l'agent dépendent uniquement de ses perceptions. Ces comportements sont classés dans des modules hiérarchiques : les modules supérieurs sont dominants et peuvent inhiber les actions du modules inférieurs. Chacun des modules est associé à une tâche particulière. Cette architecture d'agent réactif est utilisée par exemple dans [KJM11] pour modéliser des agents dans la fabrication industrielle. Les calculs dans ce type d'architecture sont généralement simples à cause de l'absence du raisonnement explicite.

Architecture hybride

Ce type d'architecture combine les deux architecture délibérative et réactive afin de tirer profit de chacune de leurs avantages et de limiter leurs inconvénients.

Parmi ces trois type d'architectures, celle qui répond au mieux à notre exigence est l'architecture hybride. Elle présente un bon compromis entre le niveau de raisonnement et la simplicité des calculs. En effet, comme notre cadre applicatif est l'environnement réel, il n'est pas toujours possible d'avoir une représentation symbolique valable de l'environnement. Sauf si on se trouve dans le cas où il est possible de créer un formalisme symbolique lui même adaptatif, en mesure de représenter des contextes applicatifs. Les composants de notre système sont assez simples et miniaturisés, donc ils ne peuvent pas toujours supporter une grande capacité de raisonnement. Nous privilégions une architecture qui n'est pas purement délibérative. Elle ne peut pas non plus être purement réactive. Nous avons besoin de doter l'agent d'un mécanisme d'adaptation incluant la boucle de rétroaction et la prise de décision.

2.2.3 Synthèse des méthodes retenues

La plupart des méthodes SMA pour la sélection d'actions est généralement conçue pour un environnement simulé. Par exemple, [TRGG99] propose un algorithme d'adaptation pour des robots autonomes dans un environnement dynamique simulé. Le modèle adapte le comportement des robots en fonction de leur perception et de leur attitude sociale. Une simulation de déplacement de piétons est présentée dans [BSA12]. Le comportement des piétons dépend de leur interaction et de leur perception. Dans ces deux exemples, toutes les situations auxquelles les agents (robots ou piétons) seront soumis sont connues au préalable. Leurs comportements sont codés en fonction de ces situations. D'autres travaux comme [SF02] proposent aussi d'autres algorithmes de sélection d'actions mais toujours pour un environnement simulé. Cependant si l'environnement est réel, l'efficacité de l'algorithme de comportement n'est pas toujours garantie. L'agent peut faire face à une situation pour laquelle il n'est pas codé explicitement : insuffisance de ressources, ajout d'un nouvel agent dans le système qui ne sera pas reconnu par les autres agents.

Les trois formes d'approche : macro, micro, et transversale, présentent chacune des avancées théoriques pertinentes vers l'élaboration de systèmes applicatifs adaptables. Néanmoins, chacune d'elles impose également des contraintes fortes sur la construction de systèmes opératoires en environnement ambiant :

— pour l'approche macro : l'orchestration du collectif doit être de haute précision et demande une visibilité et une maîtrise totale des entités constitutives.

— l'approche micro impose des agents disposant d'un fort potentiel cognitif (grande puissance de calcul), et une procédure d'apprentissage qui demande souvent un temps d'étalonnage conséquent.

— les approches transversales présentent parfois une centralisation du processus d'analyse qui fragilise la robustesse du système.

Nous souhaitons consolider la capacité d'adaptation de notre système. Au même titre que l'aspect collectif, nous nous intéressons aussi à l'aspect individuel. Nous partons de l'hypothèse qu'inclure une flexibilité dès le niveau unitaire optimise la capacité d'adaptation de l'ensemble. Aussi de façon idéale, il faudrait parvenir à coupler l'ensemble de ces trois formes d'approches en ne conservant que le meilleur de chacune d'elles : décentraliser les approches transversales en distribuant les méthodes de contrôle par rétroaction, et celles de la prise de décision sur les constituants du SMA, tout en relâchant les contraintes respectives des niveaux macro et micro par un juste équilibre entre l'adaptabilité collective et la flexibilité individuelle. Pour assurer cet équilibre entre les deux niveaux nous estimons qu'il sera plus efficace de parvenir à une solution capable de gérer de manière uniforme ces deux aspects. Les solutions existant dans la littérature gèrent toujours ces deux aspects local et global de

manière séparée. Pour la suite, nous examinons l'approche orientée comportement qui est une potentielle approche pour assurer cet équilibre entre les deux niveaux.

2.3 Approche orientée comportement

Nous avons choisi l'approche orientée comportement compte tenu de l'ensemble des contraintes que nous avons citées précédemment. Dans cette section, nous décrivons davantage cette approche et montrons sa pertinence pour notre architecture multi-agents pour un environnement ambiant. Nous abordons ensuite le mécanisme de la sélection d'actions qui constitue la principale problématique de cette approche.

2.3.1 Motivation

Distribution du modèle décisionnel

Les approches utilisées dans les SMA se distinguent par le niveau élémentaire considéré. Il s'agit soit de l'agent, de l'environnement, de l'interaction, de l'organisation ou du comportement. Nous avons vu dans 2.2.1 que l'approche de type ACPV est centrée agent, tandis que celle de type OCPV est centrée organisation. D'autres travaux s'appuient sur l'environnement. L'environnement est défini comme une abstraction de premier ordre pour la conception de systèmes multi-agents [WOO07]. Dans [GGR+14], l'environnement possède à la fois une dimension physique (support des objets) et une dimension sociale (support des interactions sociales). Certains travaux se focalisent sur des granularités plus fines comme l'interaction ou le comportement. Pour les approches orientées interaction, on peut citer par exemple l'IODA (*Interaction Oriented Design of Agent Simulation*) [KMP11]. Cette approche considère que chaque action d'un agent est une partie d'une interaction. L'agent peut alors être impliqué dans plus d'une interaction. L'interaction y est définie comme une séquence sémantique d'actions effectuées simultanément par un nombre fixé d'agents. Elle ne se limite pas uniquement à l'échange de message mais inclut toutes les interactions physiques fondamentales ou toute autre action impliquant simultanément l'environnement avec un ou plusieurs agents. La capacité d'interaction est alors distribuée entre chacun des agents. Elle est indépendante des agents et est ainsi réutilisable. Cette approche est utilisée principalement dans les simulations, particulièrement par les systèmes dont les agents présentent des interactions complexes et variées.

Dans notre travail, nous nous intéressons particulièrement aux comportements. L'approche orientée comportement est apparue dans la robotique avec les travaux de Brooks [Bro86]. Cette approche consiste à définir plusieurs comportements et de distribuer entre eux le modèle décisionnel [HG07]. Elle est encore très utilisée dans la robotique, par exemple pour le contrôle des robots artificiels semi-autonomes dans [BHGT12], pour le développement en temps réel d'un système multi-robots [LLN13]. Les travaux de [AKB11] décrivent

les avantages de cette approche par rapport aux approches classiques des systèmes monolithiques. En effet, les comportements des agents ne sont pas nécessairement complexes mais cela n'empêche pas le système global de présenter une complexité apparente qui émerge de l'interaction entre ces comportements. De plus, elle est particulièrement bien adaptée aux agents réactifs. Par la simplicité de leur caractérisation individuelle ces agents parviennent à résoudre les problèmes complexes.

Comportements

Par rapport à la notion d'interaction, la notion de comportement est plus générale. En effet, toujours dans [KMP11], l'interaction est considérée comme un comportement spécifique. Nous présentons ici deux définitions du comportement :

Définition 2.3.1. Le comportement est défini comme un ensemble déterminé d'unités de perception et d'action [Pir99].

Avec la définition 2.3.1 , on en déduit que si l'interaction implique un ou plusieurs entités (agent ou environnement), le comportement se définit uniquement par rapport à un seul agent et concerne plus les fonctions qui y sont liées. Ces deux fonctions sont assimilables respectivement aux deux fonctions *Monitoring* et *Executing* du cycle MAPE dans la section 2.1.2.

Dans d'autres travaux comme dans hanon2007selection, nous avons la définition suivante

Définition 2.3.2. Le comportement correspond à la notion d'entité plus ou moins indépendante sur l'ensemble duquel est réparti le modèle décisionnel. Chaque comportement gère ainsi un aspect du problème.

Avec la définition 2.3.1 , nous retrouvons la propriété distribuée de l'approche orientée interaction. La décision est distribuée entre les comportements. Le comportement (externe) de l'agent étant issu de la coordination ou compétition entre ces comportements en interne (décision). Si la problématique principale de l'approche orientée interaction réside dans le choix de la représentation des interactions, celle de l'approche orientée comportement concerne la sélection d'action. Cette problématique rejoint notre besoin de mécanisme de prise de décision pour choisir une action, vu dans 2.1.3. Ce mécanisme inclut les deux fonctions du MAPE *Analysing* et *Planning* dans 2.1.2.

Avec ces deux définitions, nous concluons que considérer le comportement répond convenablement aux trois aspects de l'approche pour l'adaptation dans les SMA comme dans 2.2. Pour le niveau collectif, l'approche orientée comportement distribue la décision

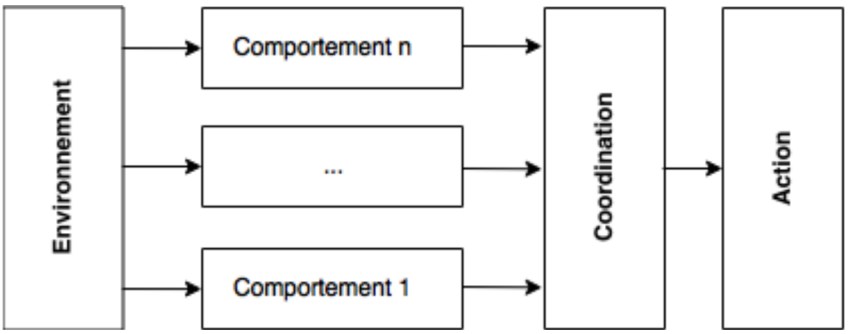


FIGURE 2.2 Architecture orientée comportement [HG07]

entre les comportements des agents. Au niveau individuel, cette approche donne à l'agent le choix d'un comportement parmi d'autres comportements en compétition. Pour les méthodes transversales, cette approche tient compte des 4 fonctions principales du cycle MAPE. De ce fait, nous avons donc choisi d'utiliser cette approche pour doter notre système d'une capacité d'adaptation. Dans la suite de ce travail, le comportement désigne une entité décisionnelle élémentaire comme défini dans la définition 2.3.1.

L'architecture correspondant à l'approche orientée comportement est donnée par la figure 2.2 suivante. Cette représentation est issue des travaux ménes dans [HG07]. La fonction de *Monitoring* s'effectue entre l'*Environnement* et les blocs de *Comportements*. Avec les perceptions reçues, les comportements prennent des décisions qui seront ensuite traitées dans le bloc de *Coordination*. Les fonctions *Analysing* et *Planning* s'effectuent dans ces blocs. Enfin, une fois qu'une décision de comportement a été validée, le bloc *Coordination* envoie cette décision au niveau du bloc *Action*, qui représente la fonction *Executing*.

2.3.2 La sélection d'actions

Description

La sélection d'actions constitue la principale problématique des approches orientées comportements. Dans [Pir99] ce problème appelé *Action Selection Problem* (ASP) est défini comme la manière dont un agent sélectionne l'action la plus appropriée ou la plus pertinente dans un moment particulier, face à une situation particulière. Cette définition rejoint notre problématique d'adaptation. Ce qui implique que la résolution de ce problème nous conduit à une solution pour l'adaptation. Cet article présente aussi les limites qui font que la solution optimale (action la plus appropriée et la plus adaptée) est difficilement atteignable dans un environnement ambiant. Des changements inattendus du contexte opératoire sont assez fréquents (insuffisances de ressources, complexité de l'environnement). Cela entraine des contraintes additionnelles sur le choix des comportements à mobiliser. Pour pallier à ces

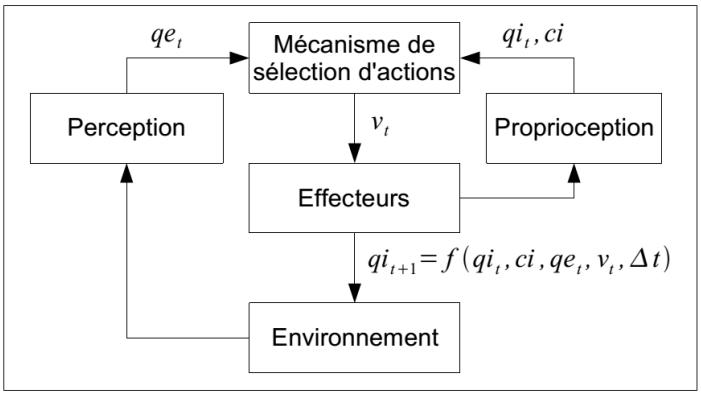


FIGURE 2.3 Sélection d'actions dans le cycle d'un agent [Han06]

problèmes, Pirjanian propose dans cet article [Pir99] des critères qualitatifs pour évaluer l'efficacité d'une action sélectionnée :

- ***Goal-orientedness*** ou l'adéquation par rapport à l'objectif. Ce critère favorise les actions destinées à atteindre un ou plusieurs objectifs.
- ***Situatedness*** ou l'adéquation par rapport au contexte. Ce critère favorise les actions pertinentes par rapport au contexte courant.
- ***Persistence*** ou la persistance. Ce critère favorise les actions destinées à atteindre l'objectif courant.
- ***Planning*** ou la planification pour éviter les situations néfastes.
- ***Robusteness*** ou la robustesse qui implique que le dysfonctionnement d'une partie du système ne doit pas impacter tout le reste.
- ***Reactivity*** ou le niveau de réactivité pour fournir des réponses rapides et opportunes.

L'ASP est généralement utilisé dans le domaine de la robotique mais on le retrouve également dans des travaux SMA. Les travaux dans [Han06], utilisent l'ASP avec la méthode de vote pour la navigation d'agents autonomes en environnement simulé. Ce travail reprend les critères cités ci-dessus et en rajoute trois autres :

- **Opportuniste** : Un agent ne doit pas hésiter à rompre la persistance pour réaliser un objectif secondaire facilement atteignable.
- **Un bon compromis** : Un agent doit favoriser les actions satisfaisant un maximum de ses objectifs en respectant ses contraintes.
- **Réutilisable** : Le modèle doit être adaptable à différents niveaux d'abstraction : choix stratégiques, tactiques et opérationnels.

Ce travail nous donne également une formalisation de la sélection d'actions dans le cycle de l'agent. La figure 2.3 représente la sélection d'actions dans le cycle de l'agent.

— qe_t indique la perception du monde à l'instant t

— **ci** indique les caractéristiques constantes propres à l'agent

— qi_t indique le vecteur de configuration à l'instant t

— v_t indique le vecteur de commande à l'instant t

— **f** indique la fonction d'évolution pour le calcul du vecteur de configuration au pas suivant $qi_{t+1} = \mathbf{f}(qi_t, \boldsymbol{ci}, qe_t, v_t)$

Méthodes existantes

La sélection d'actions a lieu au niveau du bloc *Coordination* que nous avons vu dans la figure 2.2. Les deux méthodes principales utilisées dans le mécanisme de sélection d'actions sont l'arbitrage et la fusion. Elles sont décrites comme suit dans [Pir99] :

— ***L'arbitrage*** est une approche centralisée sur un ou plusieurs comportements privilégiés. L'arbitrage constitue une forme d'internalisation du processus de négociation que l'on retrouve habituellement au niveau macro sous la forme d'interaction entre les agents. Elle nécessite donc des comportements complexes capables de prendre déjà en considération une forme de dimension collective.

— ***La fusion*** est une méthode où le contrôle est distribué entre tous les comportements. Dans ce contexte, les comportements sont coopératifs et les décisions prises tiennent en considération la décision de chacun.

A part ces deux méthodes, il existe également d'autres méthodes comme les techniques d'apprentissage. Ces techniques requièrent une capacité de raisonnement et de calcul. Avec nos composants miniaturisés et à faible capacité de calcul, il n'est pas toujours possible de les doter de cette capacité. D'autant plus que, l'apprentissage demande parfois un temps de calcul. Cela devient une contrainte pour une application en temps réel comme dans notre cadre applicatif.

2.3.3 Synthèse de l'approche orientée comportement

Nous avons vu dans cette section la pertinence de l'approche orientée comportement pour notre architecture multi-agents. En effet, elle s'applique à la fois aux deux niveaux individuel et collectif. Ensuite, la prise de décision est distribuée entre les comportements. Elle répond ainsi à notre besoin de mécanisme d'adaptation par la prise en compte des fonctions principales du cycle MAPE d'adaptation. Nous avons vu également les avantages et les inconvénients des deux méthodes de sélection d'action. Ce processus de sélection d'actions est déterminant pour l'adaptation. La limite de cette approche réside cependant sur l'optimalité de l'action sélectionnée. Les critères d'évaluation ne sont pas forcément rationnels et ne sont pas quantitatifs. Néanmoins, par rapport aux caractéristiques de notre

cadre opératoire, nous estimons que pour éviter les situations néfastes, les critères essentiels sont l'adéquation par rapport au contexte (*Situatedness*) et la planification (*Planning*).

2.4 Positionnement et contribution

2.4.1 Synthèse

Nous avons décrit à travers ce chapitre, le contexte et les bases théoriques relatives à notre problématique. Nous avons d'abord vu dans la section 2.1 que la conception d'un système adaptatif en environnement réel requiert une approche principale, un outil global et des méthodes spécifiques. Pour répondre à ces exigences, nous avons choisi l'approche multi-agents comme approche principale, une architecture comme outil global et les mécanismes de contrôle par rétroaction et de prise de décision pour les méthodes spécifiques. Ensuite, nous avons décrit cette approche principale multi-agents dans la section 2.2. Nous avons vu que la capacité collective passe par le niveau individuel. Enfin dans la section 2.3, nous nous sommes focalisés sur l'architecture qui est notre outil global. Nous avons décrit l'approche orientée comportement qui montre comment cette architecture va être édifiée. Dans cette même section, nous avons vu également comment cette approche répond aux problèmes d'adaptation. Les éléments considérés sont les fonctions du cycle d'adaptation et la sélection d'action. Elles correspondent respectivement à nos méthodes spécifiques : le contrôle par rétroaction et la prise de décision.

Ce chapitre nous a également permis d'identifier deux verrous scientifiques par rapport à l'état de l'art :

— L'approche dichotomique séparant niveau micro et niveau macro ne gère pas de manière uniforme les deux aspects individuels et collectifs qui sont pourtant fortement corrélés (section 2.2).

— Les méthodes utilisées dans la sélection d'actions ne répondent pas de manière optimale au besoin d'un système informatique ubiquitaire en environnement ambiant (section 2.3) en terme de prise en considération en temps réel de l'état du cadre opératoire.

Toutefois, nous avons également identifié d'autres problèmes relatifs à notre cadre applicatif. Nous pouvons citer par exemple, le problème de sécurité dans le réseau, l'interaction avec les utilisateurs [SM09], ou encore la gestion de données, la sécurité et la confidentialité [CSM+15]. Ces problèmes sont aussi pertinents pour la performance d'un système informatique dans un environnement ambiant, mais ils ne sont pas traités dans cette thèse.

2.4.2 Contribution

L'objectif de cette thèse est de lever les verrous identifiés en proposant de nouvelle méthode de conception d'architecture. Pour le premier verrou, nous proposons un modèle conceptuel générique pour la gestion uniforme des deux niveaux individuels et collectifs. Notre objectif est d'assurer une cohésion entre ces deux niveaux sur les questions d'ubiquité, d'adaptation et d'interaction. Pour le deuxième verrou, nous proposons une méthode de sélection d'actions qui prend en compte en temps réel l'état courant du contexte opératoire. Pour cela, nous mettons en place une unité appelée *Contrôle d'Applicabilité*. Ces deux propositions constituent notre principale contribution qui est le modèle conceptuel générique d'architecture basé sur le contrôle en temps réel des décisions à appliquer. Ce même modèle est ensuite instancié dans les deux niveaux en fonction des particularités propres à chacun. Nous pouvons maintenant enrichir avec leur originalité les éléments constitutifs de notre contribution cités dans la section 1.3.3 :

— Un pattern générique d'architecture pour la gestion des événements exceptionnels dans les deux niveaux local et global (Chapitre 3). La particularité de cette contribution réside dans l'uniformité et la généricité du modèle qui prend en compte les deux aspects micro et macro.

— Un modèle de flexibilité pour le niveau local pour optimiser l'efficacité au niveau collectif (Chapitre 4). L'originalité de ce modèle réside dans un nouveau mécanisme de sélection d'actions basée sur l'internalisation du Modèle Influence/Réaction.

— Un modèle d'adaptabilité pour la coordination du niveau collectif (Chapitre 5). Ce modèle est caractérisé par l'ajustement et le suivi des activités collective en temps réel en fonction de la modalité opératoire.

— Une proposition d'implémentation de notre modèle sous forme de librairie java (Chapitre 6). Cette implémentation montre la cohésion et la compatibilité des deux modèles une fois réunis.

Notre proposition s'inscrit dans la prévention des situations néfastes comme nous avons vu dans la section 2.1.3. Nous estimons que la correction constitue toujours une méthode efficace pour l'adaptation en environnement ambiant. Notre proposition ne vise pas à éliminer complètement la correction des erreurs. Elle constitue une phase qui précède la phase de correction. De cette manière, nous gérons mieux la progression de l'adaptabilité. En effet les problèmes d'applicabilité des décisions seront prises en compte en amont avec notre modèle. Ensuite rien n'empêche de mettre en place une deuxième phase de correction si besoin est. La prévention et la correction ne sont pas exclusives. Cette éventuelle phase de correction ne fait pas l'objet de cette thèse. Toutefois, notre modèle y reste compatible.

Chapitre 3

Modèle conceptuel générique

Nous introduisons ici notre proposition pour les problèmes vus dans le chapitre précédent. Nous proposons un modèle d'architecture générique qui constitue notre contribution principale. Il s'agit d'une solution uniforme qui répond à la fois aux problèmes de flexibilité au niveau local et d'adaptation au niveau global. Ce modèle est basé sur un contrôle d'applicabilité des comportements. Le but est de filtrer ces comportements pour ne faire exécuter que ceux qui sont opportuns. Dans ce chapitre, nous présentons les bases théoriques et le modèle formel de la contribution.

Ce chapitre est structuré comme suit. Nous commençons dans la section 3.1 par une formalisation de notre cadre applicatif que nous appelons plus précisément "contexte d'exécution". Dans la section 3.2 nous examinons en détail les problèmes d'adaptation dans les deux niveaux local et global. Nous développons ensuite le mécanisme de gestion des événements exceptionnels avec ses bases théoriques dans la section 3.3 et son modèle formel dans la section 3.4. Nous discutons de la pertinence de la proposition avec ses avantages et ses limites dans la section 3.5 avant de conclure avec la section 6.7.

3.1 Contexte d'exécution

Avant d'analyser les problématiques relatives aux systèmes informatiques en environnement ambiant, commençons d'abord par définir notre cadre applicatif que nous appelons "contexte d'exécution". Partons pour cela d'un exemple concret pour identifier ses éléments caractéristiques. Cela nous conduit ensuite à sa formalisation.

FIGURE 3.1 Accessoires du kit ADAMM (Automated Device For Asthma Monitoring And Management)

3.1.1 Illustration

Considérons ADAMM [1] (Automated Device For Asthma Monitoring And Management), un système d'objets connectés utilisé dans le domaine de la santé, représenté par la figure 3.1. ADAMM est un kit automatique de supervision et de gestion de l'asthme. Il est composé d'un gadget autonome, d'une application mobile et d'un portail web. Le gadget est porté sur le torse du patient. Son rôle est de surveiller la respiration, les toux, le battement du cœur et la température. Il est équipé d'une batterie rechargeable. Il analyse ces informations pour déterminer s'il y a anomalie et notifie ensuite le patient dans ce cas. L'application mobile est installée sur un smartphone. Le smartphone constitue l'interface de visualisation des données issues du gadget. Il contient également le programme de traitement, ou encore le rappel des médicaments. Le smartphone et le gadget sont connectés entre eux via Bluetooth Low Energy (BLE). Le portail web donne accès à l'ensemble des informations. Il possède d'autres fonctionnalités comme le transfert des informations vers le docteur par exemple, ou le stockage des informations. Une connexion internet met en relation l'application mobile avec le serveur web. A partir de cet exemple, identifions les caractéristiques d'un système informatique évoluant dans un environnement ambiant.

3.1.2 Caractérisation du contexte d'exécution

Dans la littérature, les systèmes informatiques évoluant dans l'environnement ambiant, comme ADAMM, sont dits "situés". Cette propriété "situé" est appelée *Situatedness* [ZOA+15]. Le contexte d'exécution représente alors ce cadre dans lequel se "situent" ces systèmes. C'est le contexte d'exécution qui fait que ces systèmes ont les 3 caractéristiques principales suivantes. Nous les illustrons avec l'exemple d'ADAMM.

1. https://hco2.eagledream-hosting.com/solutions/

La résolution de problème de l'environnement réel

Dans [SM09], une des caractéristiques des systèmes ambiants est sa vocation à résoudre des problèmes de l'environnement réel. ADAMM est une solution pour le suivi permanent de l'état de santé d'un patient asthmatique. Au niveau plus global, nous avons par exemple, les problèmes d'embouteillage avec comme solution les systèmes d'aide à la navigation. Dans ce contexte, nous pouvons citer Waze [2] qui est une application de trafic et de navigation communautaire en temps réel.

Interaction avec des structures physiques ou sociales

Dans [ZOA+15] les systèmes ambiants interagissent avec des structures physiques et sociales. Ces structures sont par exemple, une maison dans le cadre d'un système domotique, ou une personne humaine pour le cas de ADAMM. Elle est constituée par une ville et un ensemble de conducteurs dans le cas de Waze.

Ajustement des actions futures en fonction de l'état de l'environnement

Dans [PDM+99] les systèmes ambiants reçoivent un flux de données ou d'informations issues de l'environnement et produisent en retour un flux d'action sur cet environnement. ADAMM évolue ainsi en temps réel en fonction des données détectées sur le patient. La fonction de notification de ADAMM dépend de la présence des anomalies. A l'échelle d'une ville avec l'application Waze, l'ajustement des actions futures est par exemple le choix de l'itinéraire le plus rapide ou celui de la station service la moins chère. Cela résulte du traitement et de l'analyse des données envoyées par les conducteurs.

Définitions

La résolution de problème de l'environnement réel, l'interaction avec des structures physiques ou sociales et l'ajustement des actions futures en fonction de l'état de l'environnement renvoient à des références spatiales ou temporelles. Dans notre travail, l'ensemble de ces références définit ce que nous appelons **"Contexte d'exécution"** d'un système. Cela représente l'ensemble des conditions opératoires dans lesquelles le système évolue.

La **référence spatiale** est un ensemble d'entités matérielles, immatérielles et fonctionnelles.

— **Les entités matérielles** : sont des unités physiques concrètes. Dans ADAMM, il s'agit du smartphone sur lequel est installée l'application, et du gadget qui collecte et analyse les informations. Dans [PDM+99] la propriété *Situatedness* est associée généralement à

2. https://www.waze.com/fr

l'*Embodiement*. L'*Embodiement* est la propriété d'un système qui possède une structure physique concrète. Cette structure physique fait alors partie de l'environnement. Elle a de l'influence sur les autres composants. Cette propriété est traduite par "incarné" dans [Has03].

— **Les entités immatérielles** : sont les structures logiques ou les logicielles. Dans ADAMM, il s'agit de l'application mobile, des connectivités (BLE et connexion réseau pour le web), puis du cloud pour le stockage des informations.

— **Les activités** : constituent la partie fonctionnelle qui conditionne l'évolution du système. Dans ADAMM les étapes de l'activité sont par exemple : le monitoring du patient (i), l'analyse des informations (ii), ou la notification en cas d'anomalie (iii).

La **référence temporelle** nous renvoie vers la situation de ces activités et de celle des événements extérieurs dans le temps. Il s'agit également de l'enchainement des activités entre elles. Dans le cas de ADAMM, les activités peuvent être simultanées comme celles de (i) et (ii), ou séquentielles comme celles de (ii) et (iii). L'événement extérieur est alors par exemple, la détection d'une anomalie qui n'est pas toujours prévisible.

Nous pouvons définir maintenant le **"Contexte d'exécution"** avec tout ce que nous avons vu.

Définition 3.1.1. Soit C_e le contexte d'exécution, C_e est défini par un ensemble de référence spatiale et temporelle, noté *(espace, temps)* :

C_e = {*{espace}, {temps)}*}
espace = {Entités matérielles, Entités immatérielles, Activités}
temps = {$temps_{event}$, $temps_{activites}$}
C_e = { {Entités matérielles, Entités immatérielles, Activités} {$temps_{event}$, $temps_{activites}$}}

3.2 Analyse du problème d'adaptation au niveau local et global

Nous avons vu précédemment le contexte d'exécution qui conditionne l'évolution des systèmes ambiants. Nous examinons dans cette section comment se présentent les problèmes d'adaptation dans ce contexte d'exécution. Nous les analysons d'abord au niveau local ensuite au niveau global. En effet, pour un même contexte d'exécution, on observe des problèmes particuliers propres à chaque niveau.

3.2.1 Le niveau local

Le niveau local correspond à chacun des composants du système. Chaque composant d'un système ubiquitaire porte généralement les ressources nécessaires à l'exécution de ses actions. Nous avons identifié à ce niveau un problème d'exploitation de ressources. Nous

revenons plus en détail sur ce problème dans le chapitre 4.1. Dans ce qui suit, nous utilisons l'exemple de ADAMM pour mieux comprendre ce problème. Dans cet exemple le niveau local est le gadget ou encore l'application mobile. Ils gèrent chacun eux-même leur ressource même s'il réalisent une tâche collective : activation du capteur, ou bien envoi de signal de notification. Le comportement de suivi du patient qui est nécessaire pour le collectif dépend donc du gadget qui porte les ressources nécessaires. C'est le gadget qui mobilise ses ressources pour réaliser une action. Cette exploitation de ressources peut être perturbée par l'état dynamique du contexte d'exécution. Par exemple, l'action relative au comportement de suivi du patient peut être perturbée si le gadget atteint un certain niveau d'humidité ou s'il est sous forte chaleur.

Un comportement n'est donc jamais isolé même dans une activité collective. Il est toujours associé à l'agent qui le porte et qui lui fournit les ressources (actuateur, capteur) dont il a besoin. La disponibilité du comportement dans un collectif dépend de la disponibilité de l'agent qui le porte. Il nous est donc nécessaire de gérer les comportements par rapport à son contexte local. Cette gestion doit donc tenir compte de l'état du contexte d'exécution. Le but est de faire en sorte que la contrainte locale ne soit pas en opposition avec les besoins du niveau collectif. Cela nous conduit à trouver un modèle de conception qui offre une cohésion entre l'organisation locale (composant), et l'organisation globale (collectif).

3.2.2 Le niveau global

L'ensemble des composants constitue le niveau collectif. Nous parlons de niveau collectif quand chacun des composants met en commun ses ressources, ses compétences pour réaliser une activité commune. Nous avons identifié deux problèmes relatifs à ce niveau. Il s'agit d'un problème d'organisation et d'un problème d'interaction.

Problème d'organisation

La première force du collectif repose sur l'organisation. Les activités collectives sont possibles grâce à une organisation entre les comportements de chacun des composants (communication, suivi de la réalisation des tâches, répartition temporelle des activités,...). Dans le cas de ADAMM, le niveau collectif est l'ensemble formé par le gadget, l'application, et le portail web. Leurs activités collectives sont la supervision et la gestion de l'asthme du patient. Chaque composant de ADAMM a ses propres activités. Ces activités visent à atteindre un but collectif. Le portail web par exemple ne peut rien afficher sans l'information issue de l'application mobile. De même, les données issues du gadget ne peuvent être visualisées qu'en passant par l'application. Ainsi, si on supprime un des éléments, c'est l'ensemble qui est perturbé. La suppression du portail web est par exemple moins problématique. Les deux autres peuvent continuer à fonctionner. Par contre si c'est le gadget qui est supprimé, la

supervision du patient s'arrête.

D'une manière générale, dans un environnement ambiant, l'organisation est aussi soumise à la dynamique de l'environnement. La suppression d'un composant, donc de ses comportements, peut créer une insuffisance de ressource dans les tâches collectives. Cela peut également conduire à la réduction même de la fonctionnalité de l'ensemble. L'ajout d'un nouveau composant occasionne des accès concurrents aux ressources disponibles. Il nous faut donc maintenir l'organisation pour que le système ne se retrouve pas dans des situations défavorables. Ce maintien de l'organisation requiert des solutions d'adaptation.

Problèmes d'interaction

Nous avons identifié une deuxième force du collectif qui est l'interaction. Avant de passer du niveau local au niveau global, une phase d'enrôlement est incontournable. Il s'agit d'une étape où chacun des composants présente ses compétences et ses ressources disponibles pour une activité collective. Cette étape a comme objectif de mettre en relation chacun des composants et d'assurer leur visibilité mutuelle. Dans cette phase, la mise en commun des ressources et leur accessibilité peuvent être contraintes par l'état du contexte d'exécution. Pour ADAMM, les composants ne sont pas isolés individuellement mais interagissent entre eux via internet (application web et serveur) ou par bluetooth (smartphone et gadget). Une éventuelle coupure de ces connexions entraîne alors une perte d'information, ce qui limite l'interaction entre les composants. Entre autre, chaque composant a ses technologies de communication qui ne sont pas toujours compatibles entre eux. C'est le cas entre le gadget avec sa technologie bluetooth et le portail web accessible par internet. Ils ne peuvent donc interagir qu'en passant par l'application sur smartphone qui est compatible avec ces deux technologies.

D'une manière générale, en milieu ambiant les composants sont fortement hétérogènes. Leur protocole et leur technologie de communication ne sont pas toujours les mêmes. Dans le cas où ce sont les mêmes, la communication peut à tout moment être interrompue. Ce qui implique que le composant qui le produit peut rester inaccessible (données, informations) par manque de visibilité. Le composant concerné ne pourra pas non plus participer à l'activité commune même s'il possède les ressources et les compétences nécessaires. Il nous faut donc faciliter l'interaction pour optimiser l'accès aux ressources et leur exploitation collective. Cela requiert aussi des solutions d'adaptation.

3.2.3 Synthèse et proposition

Le caractère dynamique et hétérogène du contexte d'exécution se traduit de manière différente en fonction du niveau considéré. Au niveau local, nous avons un problème

d'exploitation des ressources. Notre objectif à ce niveau est d'assurer l'intégrité de cette exploitation de ressource et éviter ainsi des situations néfastes. Au niveau global, nous avons un problème d'organisation et un problème d'interaction causés par la dynamique du contexte d'exécution. Il nous faut donc maintenir un bon déroulement des activités collectives pour ne pas perturber l'organisation. Il faut également faciliter l'accessibilité des ressources et la communication entre les composants pour favoriser l'interaction.

Nous avons vu dans le chapitre 2, que nous utilisons l'Approche Orientée Comportements (AOC). La sélection d'actions constitue la problématique principale de cette approche. Cela nous permet d'activer seulement les actions opportunes. Avec une telle approche nous répondons à la question de comment gérer les comportements pour faire face aux verrous que nous avons cités précédemment. Au niveau local, nous utilisons la coordination des comportements pour ne faire passer que ceux qui remplissent les conditions d'exploitation des ressources. Au niveau global, cette coordination vise à vérifier la conformité des comportements par rapport aux modalités d'organisation et d'interaction. La vérification conduit ensuite à l'ajustement des comportements futurs.

Pour les deux niveaux local et global, l'ensemble des conditions des ressources, et les modalités opératoires forment les règles permettant de valider ou non un comportement avant son exécution. Nous appelons "***Contrôle d'applicabilité***" la vérification de ces règles. L'adaptation se situe alors dans l'ajustement des futurs comportements par rapport au résultat du contrôle d'applicabilité. Ce dernier constitue la particularité du modèle générique de gestion de comportements que nous proposons. Les caractères propres à chaque niveau seront ensuite gérés dans l'instanciation du modèle. Nous appelons ce modèle ***GMAS (Generic Model for Adaptive System)***. Nous avons vu qu'avec notre contexte d'exécution ambiant, nous avons une même problématique globale d'adaptation. Ces problèmes se traduisent ensuite de manière différente selon le niveau considéré. En parallèle, nous proposons un même modèle conceptuel générique. Nous l'instancions ensuite de manière particulière selon le niveau considéré.

3.3 Gestion des événements exceptionnels et contrôle d'applicabilité

Nous présentons dans cette section le modèle conceptuel générique GMAS. Dans un premier temps, la gestion des événements exceptionnels dans la Programmation Orientée Objet nous apporte une description plus concrète du principe de ce modèle. Ensuite nous le formalisons avec la description de chacun de ses briques. Nous concluons ensuite avec l'architecture résultante.

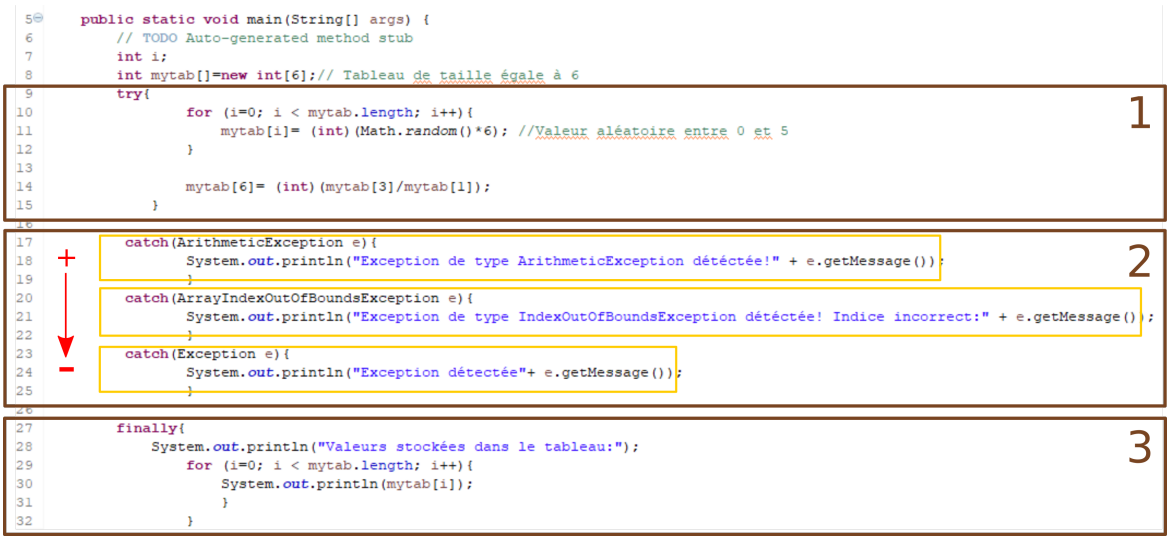


FIGURE 3.2 Gestion des exceptions avec Java

3.3.1 Gestion des événements exceptionnels en Programmation Orientée Objet

Le principe de GMAS rejoint la gestion des exceptions dans la Programmation Orientée Objet (POO). Dans la POO, les exceptions sont des événements indésirables qui peuvent se produire dans un programme. Ces exceptions peuvent conduire à l'arrêt de l'exécution du programme. La gestion des exceptions consiste alors à intercepter et traiter les morceaux de code susceptibles de générer des exceptions. De cette manière, l'exécution du programme n'est pas interrompue quelles que soient les exceptions rencontrées. Cette gestion des exceptions assure la sécurité de gestion des erreurs dans le code. Ce traitement des exceptions sans interruption de programme nous intéresse. Pour notre contexte d'exécution, nous avons également des événements imprévisibles à gérer. On peut citer par exemple, l'accès concurrent à une ressource, ou bien l'indisponibilité de celle-ci, ou encore l'erreur de connexion réseau. En même temps, nous devons garantir un fonctionnement normal. Nous allons donc examiner de près la gestion des exceptions en POO, en particulier en JAVA. L'objectif est d'identifier les points clés pour la gestion des événements exceptionnels dans notre travail.

Exemple de code

La gestion des exceptions fournit un traitement séparé du reste du code. Ce mécanisme se déroule au niveau du bloc *try/catch*. Considérons un exemple de code en JAVA représenté par la figure 3.2. Nous analysons plus en détail les annotations dans la sous-section suivante. Pour le moment, nous allons juste nous intéresser au code. Ce dernier consiste à créer un tableau de taille égale à 6. Ce tableau est ensuite rempli par des valeurs aléatoires entre 0 et 5. Nous allons ajouter une ligne de code qui présente deux exceptions (ligne 14) pour

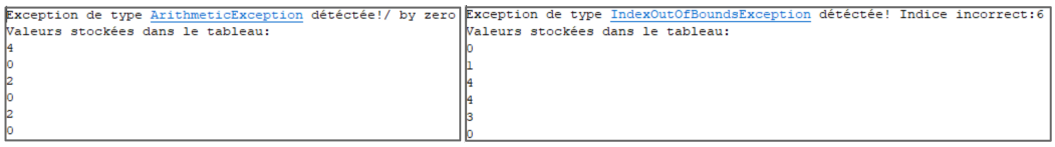


FIGURE 3.3 Deux types de résultat : *ArithmeticException* (à gauche)*IndexOutOfBoundsException* (à droite)

voir comment fonctionne la gestion des exceptions. La première est une exception sur un indice de tableau qui n'existe pas (*mytab[6]*). La deuxième est une division par zéro pour une éventuelle valeur égale à 0 du dénominateur *mytab[1]*. L'exécution de ce programme nous conduit à l'un des deux types de résultat représentés par la figure 3.3 suivante :

— Quand la valeur de ***mytab[1]* = 0** l'*ArithmeticException* est détectée. Le traitement décrit dans le bloc *catch* s'en suit avec l'affichage du type d'erreur . L'exécution continue ensuite avec les instructions dans le bloc *finally*. Ce résultat est représenté à gauche dans la figure 3.3

— Quand la valeur de ***mytab[1]*** $\neq$ **0**, l'*IndexOutOfBoundsException* est détectée. De la même manière que précédemment, le traitement décrit dans le bloc *catch* et l'exécution du bloc *finally* s'en suivent. Ce résultat est représenté à droite dans la figure 3.3.

Méthodologies retenues

Cette illustration simple de la gestion des exceptions en JAVA donne 3 méthodologies intéressantes pour notre modèle. Chaque couleur sur l'annotation de la figure 3.2 en illustre une.

— **Des étapes pour de la gestion des exceptions :** La gestion des exceptions se déroule en trois parties, marquées de 1 à 3 en marron sur la figure 3.2. Le bloc *try* (1) intercepte les lignes de code présentant des exceptions potentielles. Tandis que le bloc *catch* (2) identifie le type d'exceptions et les traite. Le bloc *finally* (3) représente le code qui est toujours exécuté quelle que soit l'exception rencontrée. En JAVA, ce bloc est facultatif.

— **Un traitement par type d'exception :** Le traitement dans le bloc *catch* est structuré suivant le type d'exception. Chaque exception a ses propres traitements. Il y a donc autant de bloc *catch* que de type d'événements. En JAVA, les exceptions sont représentées par une classe particulière : l'*Exception Class*. Cette classe possède ensuite des sous-classes qui correspondent chacune à différents types d'exception : *ArithmeticException*, *IndexOutOfBoundsException*, ou encore *IOException*, *RuntimeException*. Chaque exception est entourée en jaune dans notre exemple.

— **La hiérarchisation des traitements :** Toujours dans le bloc de traitement *catch*, les exceptions sont classées suivant une hiérarchie (en rouge). La liste des exceptions

doit toujours commencer de l'exception la plus particulière vers la plus générale. C'est pour cela que la classe mère *Exception Class* vient en dernier après les deux classes héritées *ArithmeticException*, *IndexOutOfBoundsException*. L'inversion de cet ordre génère une erreur dans le code avant même qu'on puisse l'exécuter. Les classes héritées peuvent par contre être inversées entre elles. Cette hiérarchisation optimise le temps de traitement. En effet, dès qu'une exception correspondant à une classe héritée est détectée, il est inutile de détecter un autre type d'exception plus général décrit dans la classe mère.

D'autres types de gestion des exceptions existent dans des domaines de recherche similaires au nôtre. Par exemple, les travaux dans [RA12] nous montrent que les exceptions dans le contexte d'exécution sont courantes dans les systèmes pervasifs. L'exception contextuelle (Contextual exception) provient des anomalies pouvant se produire de manière asynchrone au cours de l'exécution. Ces anomalies sont dues aux défaillances logicielles, environnementales, ou matérielles. Ce travail, propose le "context-awarness" ou vigilance au contexte pour gérer les exceptions. Ce mécanisme est appelé CAEH (Context-Awarness Exception Handling). Le CAEH sert à définir, détecter, propager et gérer des exceptions. Ce travail utilise une méthode formelle avec des formules logiques appelée *Context Kripke Structures* (C-KS).

3.3.2 Contrôle d'applicabilité

Nous avons vu au début de la section 3.3.1 qu'avec notre contexte d'exécution ambiant, nous avons aussi des éventuels événements indésirables. Nous devons gérer les comportements pour que ces événements n'impactent pas le fonctionnement normal du système. Pour cela, nous allons faire appel à ce même principe de gestion des exceptions en POO. Dans ce qui suit, nous montrons comment se traduit cette gestion des exceptions dans notre contexte. Nous décrivons deux fonctionnalités de cette gestion de comportements : la vérification des conditions d'applicabilité et la notification des comportements pour une rétroaction.

Vérification des conditions d'application

Une manière d'appliquer la gestion des exceptions de la POO consiste à identifier tous les types d'événements exceptionnels et de définir les traitements correspondants. Toutefois, nous avons un contexte d'exécution très dynamique. A la différence des exceptions en POO, les événements exceptionnels dans notre cas d'utilisation sont imprévisibles. Comme nous avons dit dans 1.2 que anticiper les éventuels changements du cadre opératoire est difficile. D'abord, parce qu'il n'est pas toujours possible d'identifier de manière exhaustive les différents cas d'évènements inattendus. Avec notre exemple ADAMM, la liste des événements exceptionnels peut être longue : mauvaise manipulation de l'utilisateur, détection de faux-positifs lors du suivi, forte humidité, et/ou forte chaleur au niveau du gadget, composant

indisponible suite à une faible batterie (gadget, smartphone), interruption de la connexion BLE entre le gadget et le smartphone... Ensuite, même dans les cas où c'est possible d'avoir une liste exhaustive, le système devient rapidement complexe en tenant compte de toutes les contraintes. Toujours avec notre exemple, en ne considérant que deux composants, nous identifions déjà 6 événements indésirables. Donc, plus le système comporte plusieurs équipements plus les événements à considérer sont conséquents. L'identification de tous les éventuels événements exceptionnels n'est donc pas la solution qui nous convient.

Nous identifions une autre manière plus appropriée d'appliquer la gestion des exceptions. Cette méthode est plus centrée sur les exigences du système que sur les événements indésirables. Nous avons vu dans la synthèse 3.2.3 de la section précédente que pour assurer un fonctionnement normal il faut respecter certaines règles de sécurité et d'exploitation des ressources au niveau local, et des modalités opératoires au niveau global. Ainsi, au lieu de lister les événements exceptionnels, il nous apparait plus pertinent d'identifier ces différentes règles et de les vérifier par la suite avant toute exécution. Au niveau local, nous pouvons avoir une connaissance précise des règles d'intégrité des interfaces d'action. Au niveau global, nous pouvons connaître précisément les protocoles de communication, et les autres modalités opératoires relatives au bon fonctionnement du collectif.

Par exemple pour le niveau local, avec le gadget d'ADAMM, les conditions favorables à son fonctionnement sont : être posé sur le torse du patient, être connecté au smartphone via BLE, et avoir un certain niveau de batterie. Si une des conditions n'est pas vérifiée, le gadget ne pourra pas faire correctement le suivi du patient. Les informations transmises risquent d'être fausses. De même, au niveau global, les conditions nécessaires sont la présence de chacun des composants, et leur connexion entre eux selon le type de connectivité.

Dans chacun de ces niveaux, nous maîtrisons mieux les règles et les conditions d'exécution des actions que les événements indésirables. Ces règles vérifient si les actions associées aux comportements sont applicables. Ainsi, de la même manière que les exceptions dans la POO sont interceptée et traitée , nous analysons les comportements pour vérifier leur conformité aux conditions d'exécution courantes. C'est ce mécanisme que nous appelons contrôle d'applicabilité. Le contrôle d'applicabilité reprend ensuite les mêmes principes que la gestion des exceptions. Ce contrôle filtre les comportements sollicités avant de les transformer en une action. Pour faire le parallèle avec la gestion des exceptions en POO, ce sont les comportements que nous allons intercepter. De la même manière que les exceptions sont hiérarchisées, nous créons aussi des hiérarchies de classes de règles d'applicabilité. Ensuite, nous définissons des traitements correspondant à chaque type de règles. Le traitement vérifie ensuite si l'état du contexte d'exécution, et/ou les modalités opératoires définies au

préalable sont favorables à l'exécution du comportement.

Nous parvenons à une première forme d'adaptabilité avec la vérification des conditions d'applicabilité. Avec ce mécanisme, seuls les comportements adaptés à l'état du contexte d'exécution seront transformés en action. Il constitue ainsi une méthode de sélection d'action. Nous avons décrit dans le chapitre précédent 2.3.2 l'évaluation d'une telle méthode avec les critères qualitatifs proposé dans [Pir99]. La vérification des conditions d'applicabilité répond fortement à l'adéquation par rapport au contexte (*Situatedness*) et la planification pour éviter les situations néfastes (*Planning*).

Adaptation par rétroaction

Nous remarquons que cette première capacité d'adaptation peut encore être enrichie. La vérification des conditions d'applicabilité des comportements est une source d'information. Si le comportement vérifie les règles, il sera transformé en action. Le cas contraire, l'action correspondante n'aura pas lieu. Cette information issue de la vérification peut donc être utilisée pour renforcer le potentiel d'adaptation du système. Nous utilisons pour cela un mécanisme de rétroaction pour renvoyer cette information vers les comportements. C'est-à-dire, le comportement sera notifié quel que soit le résultat de la vérification. Le comportement va ajuster ses futures décisions avec cette notification. Nous parvenons ainsi à une deuxième forme d'adaptabilité plus pertinente que la première. En effet, le premier niveau d'adaptation résulte de la vérification des conditions d'applicabilité. Il inhibe les comportements inopportuns. Le deuxième niveau d'adaptation résulte du couplage des informations issues de la vérification précédente avec le mécanisme de rétroaction. Il ajuste les comportements futurs.

Illustration

Considérons un exemple pour illustrer ces deux niveaux d'adaptation. Supposons que l'on souhaite utiliser le contrôle d'applicabilité pour le gadget d'ADAMM. Supposons que le niveau de batterie est inférieur à un certain seuil. Le comportement "se recharger" est alors sollicité. On va alors vérifier si les conditions de rechargement sont remplies. Une des règles est par exemple le branchement du gadget à une source. Si cette condition n'est pas remplie, le comportement "se recharger" ne sera pas exécuté. Cette information va ensuite permettre aux comportements d'ajuster leur future décision. Puisque le niveau de batterie est faible et que le rechargement n'a pas eu lieu, le gadget peut alors passer en mode économie d'énergie. Il peut par exemple désactiver les autres fonctionnalités comme le suivi du battement de cœur et de la température. Il n'active alors que le suivi des symptômes les plus pertinents

comme celui des toux et celui de la respiration.

Synthèse

Le contrôle d'applicabilité offre une gestion des comportements pour éviter les problèmes liés aux événements exceptionnels. Pour simplifier nous désignons par "applicabilité" l'unité qui effectue ce contrôle. Son rôle est d'analyser les comportements pour vérifier si l'état de l'environnement extérieur est favorable à leur exécution. L'applicabilité regroupe les modalités et les règles relatives à la sécurité de l'exécution. Elle est ainsi constituée par un ensemble de règles : règles d'utilisation des interfaces d'action, règles de priorité, règle de sécurité, protocole de suivi, modalités opératoires. Chaque règle constitue une sous-unité de l'applicabilité.

Quand un comportement vérifie toutes ces règles, l'applicabilité demande son exécution. L'exécution est alors déclenchée et suit son cours normal auprès des interfaces d'action. Certains comportements ne vérifient parfois pas les règles. Dans certains cas, deux comportements ont par exemple des accès concurrents sur une même interface d'action. Ou encore, les interfaces d'action sollicitées par les comportements ne sont pas en état de service (panne, insuffisance de ressources). L'exécution sous ces conditions de l'action correspondant à ces comportements peut être fatale. Pour éviter cela, l'applicabilité rejette les comportements concernés. Leur exécution n'aura donc pas lieu. Une fois que l'applicabilité a validé ou rejeté les comportements, elle envoie une notification portant sur le résultat de la vérification aux décisions ayant émis ces comportements. Cette notification est déterminante pour ajuster les futures décisions.

3.3.3 Architecture résultante

Externalisation du contrôle d'applicabilité

Nous avons décrit précédemment le principe général du contrôle d'applicabilité. Dans ce qui suit, nous examinons comment se positionne cette contribution par rapport à l'architecture existante. Nous faisons ici le lien entre le contrôle d'applicabilité et l'architecture orientée comportement. Pour rappel, une telle architecture est caractérisée par la sélection d'action. Celle-ci a lieu entre les comportements et l'action [HG07]. Elle s'effectue dans le bloc appelé "Coordination". La coordination est une unité à part entière distincte des blocs de comportements. Jusqu'ici notre applicabilité est similaire à la gestion des exceptions en POO. Dans cette dernière, l'élément à intercepter et le traitement correspondant sont définis ensemble dans un même code. Ce qui est à intercepter correspond aux comportements et

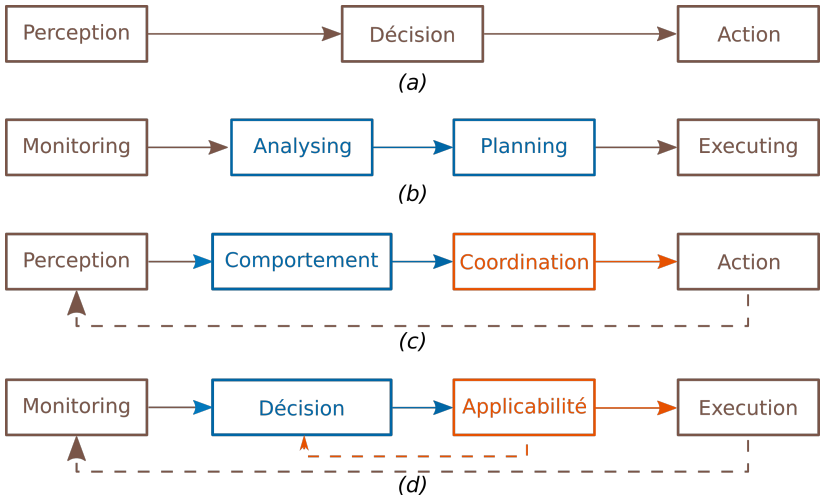


FIGURE 3.4 Comparatif des architectures (a) modèle classique, (b) Architecture MAPE, (c) architecture orientée comportements, (d) architecture avec contrôle d'applicabilité

le traitement correspond à la vérification de l'applicabilité. Il nous faut donc dissocier ces deux éléments pour être en conformité avec l'Architecture Orientée Comportement (AOC). Pour cela, le code de notre comportement se focalise exclusivement sur son propre objectif. Tout ce qui concerne ses conditions d'exécution est externalisé dans une unité à part. Cette dernière est le contrôle d'applicabilité. Ce qui constitue le bloc de coordination par rapport à l'architecture classique.

Éléments constitutifs de l'architecture

Les éléments décrits précédemment sont pris en considération pour concevoir le modèle de notre architecture. Nous partons du modèle classique de base pour positionner la particularité de notre modèle. La figure 3.4 représente le positionnement de notre architecture par rapport aux architectures existantes.

L'architecture classique des SMA est constituée par les fonctions : *Perception - Décision - Action* représentées par la première architecture *(a)* sur la figure 3.4. Les deux fonctions *Perception* et *Action* correspondent respectivement aux fonctions d'interaction avec le milieu extérieur : fonction d'entrée et fonction de sortie. Ces deux fonctions sont reprises au sein de l'architecture MAPE, représentée par *(b)* sur la figure 3.4. Elles correspondent respectivement au *Monitoring* et *Executing*. La fonction Décision dans l'architecture *(a)* se divise en deux fonctions séparées dans *(b)*. On distingue : l'*Analysing* pour identifier une information pertinente à partir des données de perception, et le *Planning* pour décider les actions correspondant aux informations analysées. Avec l'architecture orientée comportement, représentée par *(c)*, ces deux fonctions sont réalisées au sein d'un même bloc qui est le *Comportement*. Comme ce type

d'architecture est plus pour des systèmes réactifs, l'*Analysing* et le *Planning* sont beaucoup moins complexes. Ce qui fait qu'elles sont fusionnées dans un seul bloc. La particularité de l'AOC réside dans le bloc de *Coordination* (en orange sur la figure *(c)*). Ce bloc rompt le lien direct entre les comportements, (donc de la décision), et l'action. La *Coordination* sélectionne un sous-ensemble de comportements parmi un ensemble de comportements sollicités. Notre proposition d'architecture *(d)* est construit avec le même principe : sélection des décisions de comportements avant leur exécution.

La différence de notre modèle avec celui d'AOC réside dans la méthode utilisée dans la *Coordination*. Nous avons vu dans le chapitre précédent 2.3.2 que les méthodes existantes sont l'arbitrage et la fusion. Ces deux méthodes sont tout aussi efficaces mais ne permettent pas d'inclure l'état courant du contexte d'exécution dans leur prise de décision. Nous proposons une nouvelle méthode de sélection d'actions avec le contrôle d'applicabilité. Une autre particularité de notre modèle est sa capacité à notifier les comportements du résultat du contrôle. L'objectif étant de déclencher l'ajustement des comportements futurs.

Tous ces types d'architecture *(a)*, *(b)*, et *(c)* donnent chacun des éléments constitutifs pour notre architecture illustrée par la figure 3.5. Notre modèle GMAS présente quatre unités. Le modèle MAPE nous apporte nos interfaces d'interaction avec le milieu extérieur via le *Monitoring* et l'*Executing*. Nous gardons donc le terme *Monitoring* pour désigner notre interface de lecture. Nous utilisons le terme *Exécution* pour l'interface de mise en œuvre des décisions dans l'environnement. La différence de notre modèle avec celui du cycle MAPE réside dans les deux fonctions *Analysing* et *Planning*. Dans notre modèle, ces deux fonctions deviennent une seule mais sont réparties en deux étapes. Ces deux étapes séparent d'un côté l'analyse relative à l'objectif avec l'unité de *Décision* et de l'autre côté les traitements liés aux événements exceptionnels avec l'unité d' *Applicabilité*. La première étape dans le bloc de *Décision* est la sollicitation de certaines décisions en fonction de l'objectif. La décision *Décision* concerne également l'analyse élémentaire de chacun de ses éléments constitutifs. Tandis que le *Contrôle d'applicabilité* est chargé de l'analyse globale de l'ensemble des décisions sollicitées. L'intérêt de cette unité de *Contrôle d'applicabilité* est de s'assurer à tout instant que les actions à exécuter soient en accord avec l'état courant du contexte d'exécution. Le *Contrôle d'applicabilité* analyse les flux de décisions en fonction des contraintes extérieures. Son rôle est de valider ou de rejeter ces flux. Seules les décisions en accord avec les conditions extérieures courantes seront validées. Ce contrôle est ensuite doté d'une boucle de rétroaction.

Nous avons maintenant tous les éléments constitutifs de notre architecture. Dans ce qui suit, nous détaillons de manière plus formelle chacune des briques de notre architecture.

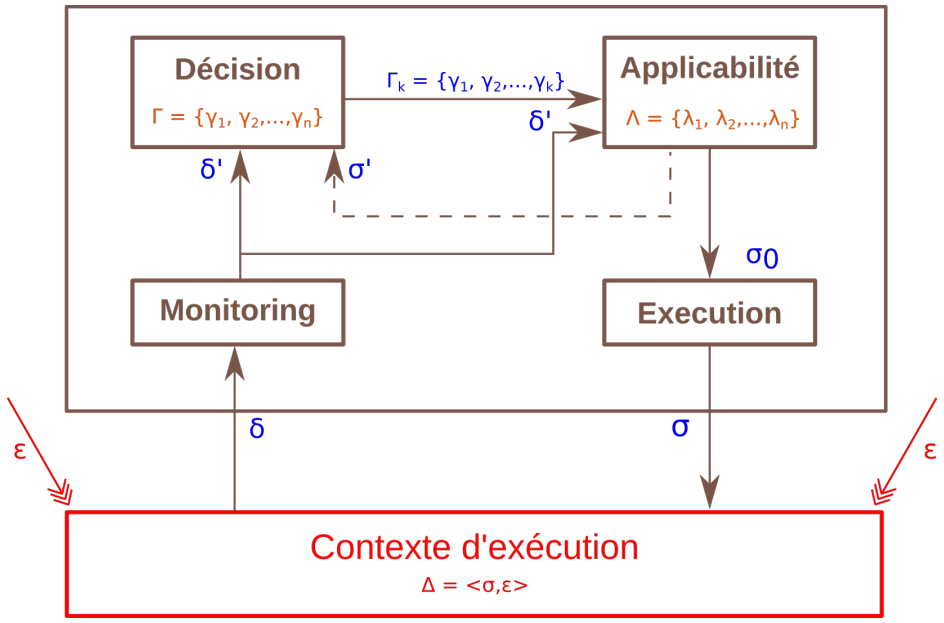


FIGURE 3.5 Modèle conceptuel de l'architecture GMAS

3.4 Modèle formel

Nous présentons dans cette section les bases théoriques de GMAS. Nous nous inspirons des travaux menés dans [FM96] sur le Modèle Influence Réaction pour formaliser notre contribution. Ce modèle est utilisé dans la simulation pour filtrer les décisions avant leur application dans l'environnement. Nous ajoutons à ce modèle existant les éléments particuliers de notre architecture : une base de comportements Γ représentée dans le bloc de *Décision*, une réaction interne σ_0 et un nouveau paramètre ε représentant l'état du contexte d'exécution suite aux événements extérieurs. Les paramètres de GMAS sont représentés sur la figure 3.5. Ils sont définis comme suit :

— Δ : Ensemble des états dynamiques du contexte d'exécution

— ε : Etat du contexte d'exécution causé par les événements extérieurs

— $\Gamma = \{\gamma_1, \gamma_2, ...\gamma_n\}$: Ensemble de décisions portant chacun un objectif

— $\Gamma_k = \{\gamma_1, \gamma_2, ...\gamma_k\}$: Flux de décision de comportements émis

— $\Lambda = \{\lambda_1, \lambda_2, ...\lambda_n\}$: Ensemble de règles d'applicabilité

— $\delta \in \Delta$: Etat perçu du contexte d'exécution

— δ' : Notification de monitoring

— σ' : Notification de rejet de la décision de comportements

— σ_0 : Demande d'action

— σ : Action du système sur le contexte d'exécution

Analysons maintenant chacun de ces paramètres suivant les unités avec lesquelles ils interagissent.

3.4.1 Contexte d'exécution

Nous avons décrit dans 3.1 le contexte d'exécution. Nous le notons ici par C_e. C_e peut être un environnement physique ou un environnement fonctionnel.

L'état dynamique de C_e est représenté par l'ensemble Δ. Cet état est fonction de deux paramètres. Le premier paramètre σ représente l'état du C_e suite aux actions de l'ensemble des équipements du système à gérer. Le deuxième paramètre ε est indépendant de ces équipements. Ils proviennent des événements extérieurs. Il s'agit de la modification du C_e suite aux événements extérieurs, généralement imprévisibles. C'est le cas par exemple des coupures éventuelles de connexion réseau si on considère un système d'objets connectés. L'ensemble de l'état du C_e est représenté par $\Delta <\sigma, \varepsilon>$.

3.4.2 Monitoring

Le Monitoring constitue l'interface d'entrée du système. Cette unité est chargée de la surveillance du C_e. Le Monitoring effectue une lecture des stimuli ou des données issues de l'environnement. Il envoie ensuite la notification correspondante vers les unités d'analyses (*Décision* et *Applicabilité*).

La fonction de monitoring est donnée par la fonction :

$$\Delta \longmapsto \Delta^{'}$$

$\Delta^{'}$ représente toujours l'état dynamique du C_e comme Δ mais $\Delta^{'}$ est sous forme de notification reconnue par les unités de *Décision* et d'*Applicabilité*.

3.4.3 Décision

Chacun des éléments de la *Décision* porte les objectifs à atteindre. C'est-à-dire ce que le système doit faire en fonction du temps et de l'espace. Ce bloc est constitué par un ensemble de décisions élémentaires qui sont des comportements indépendants. L'analyse qui s'y effectue a pour but de solliciter des comportements en fonction des informations issues du Monitoring. Dans *Décision*, seules les informations relatives à l'objectif seront prises en compte. Les conditions d'applicabilité des comportements ne sont pas encore prises en compte à ce premier niveau d'analyse. Chacun des *Décision* compare l'état du contexte d'exécution par rapport à ses objectifs. Il prend ensuite la décision de s'activer ou pas en fonction de l'information reçue.

Soit Γ l'ensemble des comportements élémentaires dans *Décision* tel que $\Gamma = \{\gamma_1, \gamma_2,...\gamma_n\}$, avec $n \in \mathbb{N}$. Chacun des éléments de la *Décision* prend en entrée les flux d'informations $\delta^{'}$

$\in \Delta'$ issues du Monitoring, et $\sigma' \in \Sigma'$ issues de l'*Applicabilité*. L'analyse de ces informations conduit à l'activation d'un sous ensemble de l'ensemble Γ. Cette activation de comportement est donnée par la fonction :

$$Activation : \Delta' \text{ x } \Sigma' \longmapsto \Gamma_k$$

avec $\Gamma_k \subseteq \Gamma$
$\Gamma_k = \gamma_1, \gamma_2, ...\gamma_k$, avec k $\leq$ n.

3.4.4 Applicabilité

L'unité *Applicabilité* constitue le deuxième niveau d'analyse. Elle filtre les flux de décision (Γ_k) en fonction de l'état de l'environnement (δ, Δ). L'*Applicabilité* est ainsi constituée par un ensemble de règles. Son objectif est de vérifier si les comportements sollicités satisfont les conditions définies dans ces règles. Ces règles représentent les règles d'intégrité de l'interface d'action, ou encore les règles de cohérence de l'ensemble des comportements activés. Ce bloc présente deux sorties. Les flux de comportements entrants qui vérifient toutes les règles deviennent une demande d'action auprès de l'*Execution*. C'est ce qui constitue la première sortie. Ceux qui ne vérifient pas les règles sont rejetés. L'*Applicabilité* envoie alors une notification de ce rejet vers la *Décision*. C'est ce qui constitue la deuxième sortie.

Soit Λ l'ensemble des règles constituant l'*Applicabilité*, tel que $\Lambda = \{\lambda_1, \lambda_2, ...\lambda_n\}$. Chaque règle λ_i représente une fonction. L'*Applicabilité* prend en entrée les flux d'informations $\delta' \in \Delta'$ issues du *Monitoring*, et les flux de décisions $\gamma \in \Gamma_k$. L'analyse de ces informations conduit à la sélection ou à l'inhibition des flux de comportements en fonction des règles Λ. La fonction de vérification de l'*Applicabilité* est donnée par :

$$Vérification : \Delta \text{ x } \Gamma \text{ x } \Lambda \longmapsto \Sigma.$$

avec $\Sigma = \Sigma' \cup \Sigma_0$, $\Sigma \cap \Sigma_0$

$\forall\ \lambda_i\ \varepsilon\ \Lambda$, la fonction $\lambda_i\ (\gamma, \delta)$ constitue un filtre pour les flux de comportements γ. La fonction de l'*Applicabilité* λ_i traite un par un chaque comportement du flux en entrée γ en fonction des flux d'informations issues du *Monitoring* δ. Cette fonction a deux valeurs possibles 0 ou 1. Elle prend la valeur 1 si le comportement γ vérifie toutes les règles de l'ensemble Λ. Elle est égale à 0 si une des conditions n'est pas vérifiée. Nous obtenons :

$$Vérification(\delta, \gamma, \lambda) \begin{cases} \sigma_0 \in \Sigma_0 if \lambda_i(\gamma, \delta) = 1 \\ \sigma' \in \Sigma' if \lambda_i(\gamma, \delta) = 0 \end{cases}$$

σ_0 est une demande d'action envoyée vers l'*Exécution*
σ' est une notification d'une décision rejetée renvoyée vers la *Décision*.

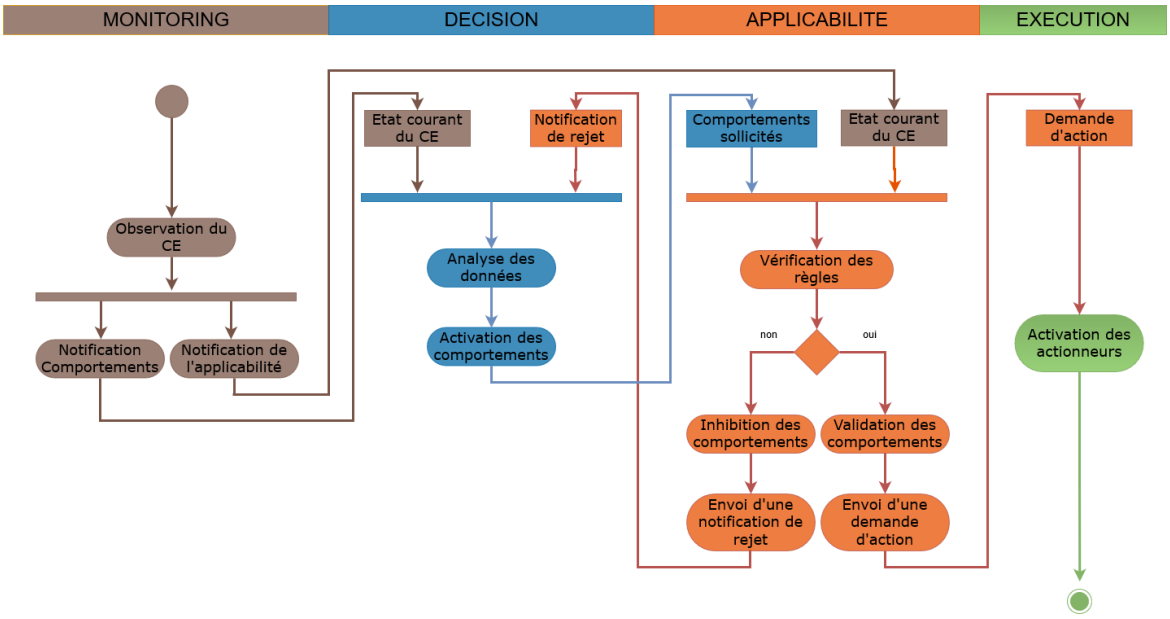


FIGURE 3.6 Diagramme d'activité de GMAS sous UML (CE : Contexte d'exécution)

3.4.5 Exécution

L'*Exécution* constitue l'interface de sortie du système. Cette unité est chargée de transformer les demandes d'action issues de l'*Applicabilité* en une action dans le contexte d'exécution. Dans notre modèle, la *Décision* ne peut pas agir directement sur l'*Exécution* pour des raisons de sécurité. L'intérêt de l'unité d'*Applicabilité* est justement d'assurer l'intégrité de l'exécution.

La fonction *Exécution* est une fonction de la demande d'action Σ_0 dans l'ensemble des actions Σ. Ce dernier modifie directement le contexte d'exécution.

$$Exécution = \Sigma_0 \longmapsto \Sigma.$$

3.4.6 Diagramme d'activité

Le fonctionnement interne de GMAS est caractérisé par l'échange de données entre les fonctions. La figure 3.6 résume ce fonctionnement sous forme de diagramme d'activité sous UML. De gauche à droite, on a le *Monitoring*, la *Décision*, l'*Applicabilité* et l'*Exécution*. Les rectangles en bords droits représentent les données issues de chaque bloc. Ces données sont soit des notifications comme l'état courant du C_e et la notification du rejet, soit des décisions comme les comportements et la demande d'action. Ces données constituent les paramètres des différentes fonctions dans chaque bloc. Les rectangles aux bords arrondis représentent ces fonctions.

3.5 Analyse qualitative

Nous évaluons les points essentiels de GMAS dans cette section. Nous commençons par son originalité par rapport aux méthodes classiques. Ensuite nous détaillons la nouvelle méthodologie de gestion de comportements apportée par notre modèle. Nous terminons avec l'analyse de ses limites par rapport aux critères d'évaluation qualitative.

Une nouvelle méthode de sélection d'action

La prise de décision s'effectue dans tout le processus de sélection d'actions dans l'AOC. La décision se fait en deux étapes. La première étape est le choix des comportements qui seront soumis à la coordination. La deuxième étape est le choix de ceux qui peuvent être transformés en action. GMAS effectue ces deux fonctions et remplit alors la fonction de prise de décision. En même temps le contrôle par rétroaction est aussi pris en compte dans ce modèle. Cela se traduit par le renvoi des résultats des traitements vers les comportements émetteurs pour les ajuster dans le futur. Ces caractéristiques de l'applicabilité font qu'elle correspond aux méthodes spécifiques que nous avons rappelé précédemment.

L'Applicabilité apporte également une nouvelle méthode de sélection d'actions par rapport aux méthodes classiques de l'AOC. Les deux méthodes classiques sont l'arbitrage et la fusion comme cité dans [Pir99] et repris dans [HGM05]. Nous avons décrit ces deux méthodes dans 2.3.2. Vu le caractère distribué de notre contexte d'exécution, il est difficile d'opter pour l'arbitrage. Dans cette méthode, la décision finale de sélection de comportements revient à des comportements privilégiés. Comme notre contexte d'exécution est fortement dynamique, des comportements peuvent être ajoutés ou supprimés à tout moment. La disponibilité des comportements privilégiés n'est plus assurée. Comme la décision y est centralisée, une éventuelle suppression perturbe tout le processus de sélection d'action. Ce problème n'apparait pas dans la méthode de fusion. Dans cette méthode le choix de l'action résulte d'un compromis entre les comportements présents. Néanmoins, la méthode de fusion présente également des limites. Pour arriver à ce compromis, chacun des comportements doit avoir une connaissance des autres comportements pour juger de sa transformation en action. Une autre connaissance à considérer est l'état du contexte d'exécution. Doter les comportements de telles connaissances devient vite complexe dans notre cas, puisque les comportements associés aux couches logicielles des composants sont nombreux. Nous parvenons à contourner ces problèmes avec notre méthode. D'abord, la décision de transformer un comportement en une action ou le rejeter revient à l'*Applicabilité* qui est indépendante des comportements. Ensuite, les critères de validation ou de rejet d'une décision de comportement sont liés aux conditions extérieures. Cela assure une adéquation entre l'état du contexte d'exécution et les actions à réaliser et évite ainsi les situations néfastes.

En résumé, nous pouvons dire que l'*Applicabilité* apporte une nouvelle méthode de sélection d'actions sur deux aspects. D'abord, d'un point de vue plus général, elle englobe à la fois la prise de décision et le contrôle par rétroaction. Elle répond ainsi aux méthodes spécifiques requises pour l'adaptation 2.1.3. Ensuite, par rapport aux méthodes de sélection d'actions dans l'AOC, la nôtre se différencie par la prise en compte de l'état du contexte d'exécution. Elle rend la décision finale indépendante des comportements sollicités. Ainsi elle est plus adaptée pour notre cadre applicatif qui est l'environnement ambiant.

Optimisation du code avec l'externalisation de l'applicabilité

Nous sommes partis du principe de gestion des exceptions dans la POO pour concevoir l'*Applicabilité* 3.3. Dans ce contexte, le traitement des événements exceptionnels est inclus dans le même code que le comportement. Ensuite, nous avons vu dans la section 3.3.3 que pour être en conformité avec l'AOC, il nous fallait séparer le code de comportement de celui de sa coordination. Ce qui nous a conduit à l'externalisation de l'applicabilité. Nous remarquons que cette configuration améliore les codes de comportements. Pour mieux comprendre cet intérêt pour le code, analysons le cas où le comportement et la vérification de ses conditions d'applicabilité ne sont pas dissociés.

Revenons pour cela sur le cas d'ADAMM pour avoir un exemple plus concret. Supposons que nous avons à écrire le code d'un des comportements du gadget. Considérons le comportement qui correspond au suivi de la respiration et l'envoi de l'information correspondante au smartphone.

Algorithm 1 Exemple de code de suivi de la respiration

```
1:  frequence;
2:  try
3:      get(frequence);
4:      get(frequence.date());
5:      send(frequence, frequence.date()) ⟶ smartphone;
6:  catch PatientConnection exception
7:      Unable (SoundNotification);
8:  catch SmartphoneConnection exception
9:      Unable (SoundNotification);
10: catch ResourceInsuffisancy exception
11:     Request(recharging);
12: finally
13:     Send notification
14: end try
```

L'algorithme 1 représente le suivi de la respiration. Son objectif est d'envoyer la fréquence respiratoire du patient vers le smartphone. Cet objectif est décrit dans le bloc *try*.

Le gadget a besoin de ses interfaces d'action pour atteindre cet objectif. L'exploitation de ces interfaces d'action est définie par des règles d'intégrité. Du plus particulier au plus général, il s'agit d'une connexion avec le patient, une connexion au smartphone et un niveau de batterie suffisant. La vérification de chacune de ces conditions constitue les traitements représentés par le bloc *catch*. De ce fait, plus nous avons des conditions plus nous avons des traitements à effectuer. Nous remarquons alors que plus les comportements sont complexes, plus le nombre des traitements va être conséquent. Ce qui conduit assez vite à des codes très chargés. Le code n'est donc plus centré sur son objectif propre mais traite plus les contraintes extérieures. Dans notre exemple, nous avons trois conditions à vérifier pour un seul objectif.

Maintenant considérons que nous ayons à écrire le code d'un autre comportement similaire. Prenons le suivi des toux par exemple. Les interfaces d'actions sollicitées seront les mêmes que précédemment. Dans ce cas, les règles précédentes seront aussi reprises dans le code de ce comportement. La redondance des lignes de code sera inévitable pour les comportements exploitant les mêmes interfaces d'action pour un même contexte d'exécution.

Ces deux aspects montrent qu'inclure la vérification des règles dans le code de comportement conduit à des codes chargés avec des éventuelles redondances. Nous parvenons cependant à éviter ces problèmes en externalisant les traitements liés aux vérifications des règles. D'abord, le code de comportement sera focalisé uniquement sur son objectif. Donc il est réduit à ce qui est décrit dans le bloc *try* dans notre exemple. Ensuite, les traitements correspondants aux mêmes interfaces d'actions seront factorisés pour éviter les redondances. Ce qui revient à avoir une sorte de bloc catch factorisé. Initialement cette externalisation était une contrainte imposée par l'AOC. En la mettant en place, nous constatons qu'elle optimise également le code de comportements.

Une méthodologie pour la gestion des comportements

Outre l'optimisation du code de comportements, l'externalisation de l'applicabilité nous apporte aussi une méthodologie pour la gestion des comportements. La séparation entre les comportements et l'action existe déjà dans l'AOC. Toutefois, la coordination reste dépendante des comportements que ce soit avec la méthode de l'arbitrage qu'avec la méthode de fusion.

Avec ces méthodes, si le comportement change de contexte d'exécution, on aura à modifier son code pour être compatible avec le nouveau cadre opératoire. Avec des comportements centrés uniquement sur l'objectif, il est possible de les ré-utiliser dans d'autres contextes. Par exemple, si nous revenons sur l'algorithme 1, l'objectif est défini dans le bloc *try* (lignes 3 à 5). Ce même code peut être utilisé pour un agent pilotant un capteur puisqu'il consiste à récupérer les données et à les envoyer. Dans ce cas les conditions d'application des comportements seront modifiées en fonction du nouveau cadre applicatif. Il est même

possible d'ajuster le niveau d'adaptabilité en fonction de la criticité de l'environnement sans que le code des comportements change. Par exemple, pour le gadget d'ADAMM, pour une utilisation qui requiert plus de précaution, on peut ajouter la vérification de la température ambiante avant toute action. Ceci pour éviter tout risque de sur chauffage. Dans ce contexte, le code de comportement ne change pas. Cette ré-utilisation va dans les deux sens. Comme les règles et les conditions sont centrées sur les interfaces d'action et de l'exécution, elles peuvent être ré-utilisées pour tout comportement sollicitant ces interfaces.

Efficacité par rapport aux critères d'évaluation

Nous avons décrit dans le chapitre 2.3.2 les critères d'évaluation de la méthode de sélection d'action. Nous y avons conclu que vu le caractère dynamique de notre contexte d'exécution certains critères sont privilégiés par rapport à d'autres. La vérification de l'applicabilité est focalisée exclusivement sur l'intégrité de l'exécution. Nous avons vu dans 3.3.2 que cette vérification répond ainsi au critère ***Situatedness*** et de ***Planning***. Nous montrons dans la section précédente qu'il répond également au critère **Réutilisable**.

Certains critères sont moyennement vérifiés dans notre modèle. La ***Robustesse*** implique que le dysfonctionnement d'une partie du système ne doit pas impacter tout le reste. Notre solution est robuste du moment où le dysfonctionnement se trouve dans les comportements. Par contre une perturbation du mécanisme de vérification peut impacter tout le reste. En effet, les comportements ne sont pas définis avec leur condition d'application. Les exécuter directement sans contrôle est alors risqué. Le critère de réactivité (***Reactivity***) concerne les réponses rapides et opportunes. De notre côté, le contrôle est fonction de l'état courant du contexte d'exécution. Dans ce sens sa réponse est opportune. Par contre, l'ajustement des comportements futurs par rétroaction peut être moins réactif. Comme nous n'utilisons pas de méthode d'apprentissage quelques tentatives de comportements peuvent se produire avant de trouver le comportement opportun.

Notre modèle de sélection d'actions répond très peu aux critères relatifs à l'objectif. Il s'agit notamment de l'adéquation par rapport à l'objectif (***Goal-orientedness***), la persistance (***Persistence***), l'***Opportuniste*** et le ***bon compromis***. En effet, notre principal objectif est d'éviter toute situation néfaste. Cela implique que c'est l'adéquation par rapport au contexte qui sera priorisée au dépens de l'adéquation par rapport à l'objectif.

3.6 Conclusion

Nous avons proposé dans ce chapitre notre modèle conceptuel générique appelé GMAS. Nous avons formalisé le cadre dans lequel il s'inscrit avec le contexte d'exécution dans 3.1. Nous avons identifié les problèmes d'adaptation liés à ce contexte d'exécution dans les niveaux local et global dans 3.2. Cela nous a conduit à la conception du modèle théorique de notre proposition. Nous avons décrit le principe de GMAS avec le principe de gestion des exceptions dans les POO dans 3.3. GMAS est caractérisé par le contrôle d'applicabilité des comportements avant leur transformation en action dans le contexte d'exécution ou leur inhibition. Son objectif est d'apporter un potentiel d'adaptation pour les systèmes évoluant dans un environnement ambiant. Son originalité réside dans la gestion uniforme des comportements tant pour le niveau local que pour le niveau global. Le niveau local fait référence à un composant, tandis que le niveau global correspond à l'ensemble formé par ces composants. En proposant ce modèle, nous apportons également des méthodologies intéressantes pour la sélection d'action, l'optimisation du code de comportement, et la gestion de comportements 3.5.

Notre cahier des charges décrit dans le chapitre précédent 2.1.3 nous indique une approche SMA comme approche principale, une architecture comme outil global, un mécanisme de contrôle par retroaction et un mécanisme de prise de décision comme méthodes spécifiques. Pour l'approche principale, GMAS s'inscrit dans une démarche multi-agents. La décision est distribuée entre les comportements. Pour l'outil global, GMAS est un modèle d'architecture. Il définit une couche physique pour l'interaction avec le contexte d'exécution et une couche logique pour l'analyse et la décision. Cette architecture favorise l'implémentation des deux mécanismes correspondant aux deux méthodes spécifiques. Le contrôle par rétroaction se fait à partir de deux niveaux : à partir d'une réaction interne et d'une réaction externe. La réaction interne suite à un rejet d'un comportement et la réaction externe issue de l'exécution d'une action. Le mécanisme de prise de décision se déroule dans le processus de sélection d'actions comme nous avons vu dans 3.5. GMAS constitue donc une solution complète répondant aux exigences des systèmes ambiants.

Après cette description théorique de GMAS, nous développons maintenant son instanciation dans les chapitres suivants. Le chapitre 4 décrit l'instanciation au niveau local pour résoudre les problèmes d'accès aux ressources locales. Tandis que l'instanciation pour le niveau global est décrite dans le chapitre 5. L'objectif est de montrer comment GMAS gère les problèmes d'interaction et d'organisation à ce niveau.

Chapitre 4

Flexibilité individuelle

Ce chapitre décrit l'instanciation de notre modèle conceptuel correspondant au niveau local. Nous estimons que les limitations des solutions existantes sont dues à l'insuffisance de potentiel du niveau unitaire. Les modèles d'architecture agent existant en SMA sont plus orientés sur les aspects collectifs. Nous partons ici de l'hypothèse qu'une flexibilité individuelle peut être un levier pour l'adaptation globale du système. Nous proposons un modèle d'architecture interne d'agent, appelé MECA. Il est caractérisé par la réutilisation du Modèle Influence/Réaction.

Nous commençons ce chapitre avec la description du niveau local dans la section 4.1 et ses problèmes caractéristiques dans la section 4.2. Nous décrivons ensuite le modèle Influence Réaction dans 4.3 pour montrer sa pertinence par rapport à ces problèmes. Par la suite nous présentons le modèle classique de l'agent dans 4.4 pour mieux positionner notre proposition de modèle agent dans 4.5. Nous faisons une analyse du modèle dans la section 4.6. Nous terminons dans la section 4.7 avec la conclusion.

4.1 Le niveau local

Nous nous intéressons au niveau local dans le but d'optimiser l'activité collective. Comme nous avons décrit au début de ce manuscrit dans la section 1.3.2, nous estimons qu'il est nécessaire d'instaurer un équilibre entre l'individuel et le collectif. La solution la plus évidente dans ce contexte revient alors de doter chaque élément individuel d'une capacité d'adaptation. Nous avons vu dans la section 2.2.3 que dans la littérature l'adaptation au niveau individuel requiert un potentiel cognitif. Cette exigence de capacité cognitive ne peut malheureusement pas être satisfaire dans notre cadre applicatif. En effet, nous avons vu également que les systèmes ambiants sont constitués de composants miniaturisés avec une capacité de calcul limitée. La première solution ne peut donc pas être utilisée. Par conséquent, nous proposons une autre solution pour les composants individuels pour faire face aux événements imprévus. Dans ce sens, nous parlons alors de flexibilité au lieu d'adaptation

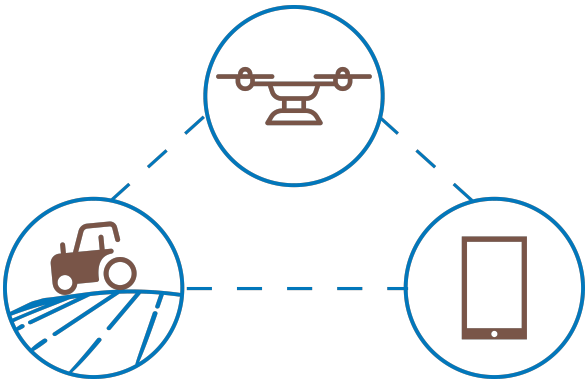


FIGURE 4.1 Exemple d'objets connectés pour l'agriculture raisonnée

puisque nous n'utilisons pas d'algorithmes de raisonnement ou d'apprentissage comme pour les agents adaptatifs. Comme nous avons des composants réactifs réagissant par rapport à ses perceptions, notre modèle d'architecture pour le niveau local s'appuie également sur la perception. Dans la suite de ce chapitre, nous utiliserons le terme de "flexibilité" pour désigner cette souplesse de l'agent face aux imprévus. Nous évitons ainsi l'emploi du mot "adaptation individuelle" que nous réservons pour les solutions basées sur un fort niveau de raisonnement.

4.1.1 Contexte

Considérons un exemple d'objets connectés pour dresser le cadre du contexte de niveau local. Il s'agit d'un outil informatique utilisé dans le cadre de l'agriculture raisonnée (Smart Agriculture) représentée par la figure 4.1. Dans ce contexte, l'agriculteur utilise trois équipements différents : un drone, un robot et une application smartphone. Le drone survole le champ et y collecte en temps réel les informations qui en proviennent. Cela peut être par exemple, le ciblage d'un domaine nécessitant une attention particulière : irrigation, désherbage. Le robot assure deux fonctions : le désherbage et le binage. Ce robot utilise les informations issues du drone et de l'application smartphone pour identifier la zone à traiter. Le robot est autonome et dispose d'un lidar pour détecter les obstacles. L'application smartphone est utilisée comme panneau de contrôle du système.

Ce système d'objets connectés a comme objectif d'aider l'agriculteur. Pour cela, différentes compétences issues de chacun des équipements sont mises en commun : collecte d'information en temps réel, désherbage, binage, affichage des informations. Ces compétences requièrent des ressources pour être exploitées. Par exemple, le drone nécessite une caméra, différents capteurs et aussi des moyens de déplacement pour collecter les informations.

Cette mise à disposition des comportements est soumise aux conditions dynamiques du contexte d'exécution. Analysons de près le cas de la collecte d'information. Cette compétence

est importante pour le collectif puisque les informations collectées sont déterminantes dans la prise de décision au niveau du panneau de contrôle. Cependant lorsque le drone se trouve dans des conditions défavorables comme de fortes pluies ou du vent fort, cette collecte d'information sera perturbée. L'exploitation de cette compétence fait partie des comportements du drone. Un comportement n'est donc jamais isolé. Il est avant tout soumis à des contraintes locales. Ces contraintes ne sont pas uniquement dues aux conditions extérieures d'exécution. Une autre contrainte importante est le composant qui le porte. En effet même dans le cas où les conditions d'exécution sont favorables, la décision d'activer ou pas le comportement appartient uniquement au drone. Aucun autre composant ne peut alors décider à sa place.

Même si un comportement est engagé dans une activité collective, quand le composant qui le porte est hors service, ce comportement n'est plus accessible. Il est donc impossible de dissocier un comportement de ses contraintes locales.

4.1.2 Définitions

Nous allons partir de la caractérisation du composant unitaire dans les SMA et les systèmes auto-adaptatifs (SAS) pour mieux définir notre niveau local.

Dans les SMA le composant unitaire est l'agent [Jac95]. Il s'agit d'une unité autonome évoluant dans un environnement. Il est animé par un ou plusieurs objectifs. Il possède les moyens d'action pour les atteindre. Les objectifs correspondent aux comportements. Nous avons vu dans le chapitre 2.3.1 que dans l'AOC [1], les comportements sont des entités plus ou moins indépendantes entre lesquelles est réparti le modèle décisionnel.

Pour les SAS, un système est constitué de deux composants : la partie physique qui représente les ressources à gérer et la partie logique qui inclut le contrôle de l'adaptation. Quand la partie logique de l'adaptation est implémentée dans la partie physique, on parle d'implémentation interne. Dans le cas où la partie logique est séparée de la partie physique, on parle d'implémentation externe.

Dans notre travail, le niveau local correspond à la fois à l'agent dans les SMA et à l'ensemble de ces deux composants constitutifs d'un SAS. Plus précisément, nous définissons le niveau local par l'ensemble indissociable formé par les comportements et les contraintes relatives à leur réalisation concrète dans le contexte d'exécution. Ces contraintes proviennent des entités physiques et des entités logicielles. Les entités physiques sont les interfaces de perception et les interfaces d'action. L'interface de perception collecte les informations à l'entrée. L'interface d'exécution réalise l'action dans l'environnement. L'entité logicielle renvoie à la gestion interne des comportements. Elle représente l'ensemble des mécanismes d'activation ou de gestion des comportements avant leur exécution concrète. Dans notre travail, la partie physique et la partie logicielle sont séparées. Par rapport au SAS, nous

1. Approche Orientée Comportement

avons donc une approche externe. Cette séparation est pertinente en termes de maintenance et de réutilisation du contrôle de la flexibilité.

Cette description du niveau local nous montre que l'élément déterminant du niveau local est l'ensemble de ses comportements. Il nous faut donc prendre en considération les contraintes relatives à ces comportements pour exploiter le niveau local. Par ailleurs. Dans la suite de ce chapitre, analysons comment tenir compte de ces contraintes pour exploiter ces comportements.

4.2 Gestion de comportements

La gestion des comportements a été un des axes des travaux de recherche de notre équipe de recherche. Dans ce contexte, l'équipe a déjà exploité ces comportements dans la simulation avec une approche dite l'approche pluri-comportementale [GPC12] [PCSR06]. La souplesse offerte par la manipulation des comportements nous conduit dans cette thèse à se focaliser davantage sur les comportements.

La gestion des comportements soulève deux problèmes majeurs. Le premier concerne l'adéquation entre un comportement et les ressources qui lui sont nécessaires. Le deuxième problème concerne l'éventuel accès concurrent de plusieurs comportements par rapport à une même ressource. Le deuxième problème concerne les " accès concurrents", c'est à dire le déclenchement simultané, au niveau d'une ressource donnée, de plusieurs comportements mutuellement exclusifs.

4.2.1 Exploitation des ressources

Les comportements sont des entités décisionnelles comme nous avons vu dans la définition précédente 4.1.2. Les décisions peuvent être réalisées grâce aux interfaces d'action qui fournissent les ressources nécessaires. Par exemple, quand le drone décide de réaliser une collecte d'information, il doit mobiliser sa caméra et ses moyens de déplacement. Dans notre travail, chaque agent possède ses propres comportements et ses propres interfaces. Un comportement est toujours associé à un seul et unique agent. Aucun comportement n'est isolé. Cela n'empêche pas cependant que deux agents différents présentent le même comportement. Avec notre exemple précédent, on peut avoir deux robots qui peuvent faire le désherbage. Nous avons dans ce cas un comportement commun. Toutefois, la gestion du comportement n'est pas commune. Chaque agent gère son comportement de manière indépendante. Il en est de même pour les interfaces de perception et d'action. Une interface ne peut appartenir à la fois à deux agents. Cela implique que la gestion des ressources relatives à ces interfaces sont propres à l'agent. Par exemple avec le robot, pour le comportement de désherbage, la ressource à mobiliser est le moteur, et les roues sont les interfaces d'action.

Un comportement ne peut donc s'exécuter que lorsque les ressources sont favorables. Le problème est ici de déterminer comment mettre en adéquation les décisions avec les ressources.

4.2.2 Accès concurrent

Avec notre approche AOC, notre agent possède plusieurs comportements. Un autre problème relatif au niveau local est l'accès concurrent aux ressources. Deux comportements différents peuvent par exemple solliciter simultanément les mêmes ressources alors qu'ils ne peuvent pas être exécutés en même temps. Toujours avec notre robot, nous avons l'exemple de deux comportements qui sollicitent la même interface. Il s'agit du comportement de désherbage et celui du rechargement de la batterie. Le premier nécessite l'activation des roues tandis que le deuxième ne peut se faire que lorsque les roues sont au repos. Cela implique qu'il est impossible pour le robot de réaliser en même temps ces deux comportements. Comme les comportements sont indépendants, ils n'ont pas de connaissance des autres comportements sollicités. Il faut donc que nous gérions ce cas de figure pour éviter l'endommagement des interfaces d'action.

Notre objectif au niveau local est de résoudre ces deux problèmes. L'instance de GMAS que nous spécialisons vise à atteindre cet objectif. Nous décrivons notre proposition dans la suite de ce chapitre.

4.3 Internalisation du Modèle Infuence-Réaction

Les comportements que nous avons à gérer se trouvent à l'intérieur de l'agent. Nous instancions dons le contrôle d'applicabilité dans l'agent. Comme ce contrôle d'applicabilité est basé sur l'IRM, nous revenons dans un premier temps sur l'IRM avant de définir la fonction de régulation qui est l'instance du contrôle d'applicabilité pour le niveau local.

4.3.1 Motivation

L'IRM [FM96] utilisé dans la simulation gère l'interaction entre l'agent et l'environnement. Dans ce modèle, l'agent n'agit pas directement dans l'environnement mais y émet des influences. Il revient alors au code de gestion de l'environnement de calculer les réactions à associer à ces influences en fonction des caractéristiques et des dynamiques souhaitées pour représenter l'environnement. La figure 4.2 suivante représente à gauche le principe de l'IRM tel qu'il est défini par Ferber dans [FM96]. L'image de droite correspond à la réutilisation de l'IRM dans notre travail.

L'IRM a été conçu en particulier pour répondre au problème de la simultanéité. Au cours d'un même cycle de simulation, les influences émises par l'ensemble des agents sont

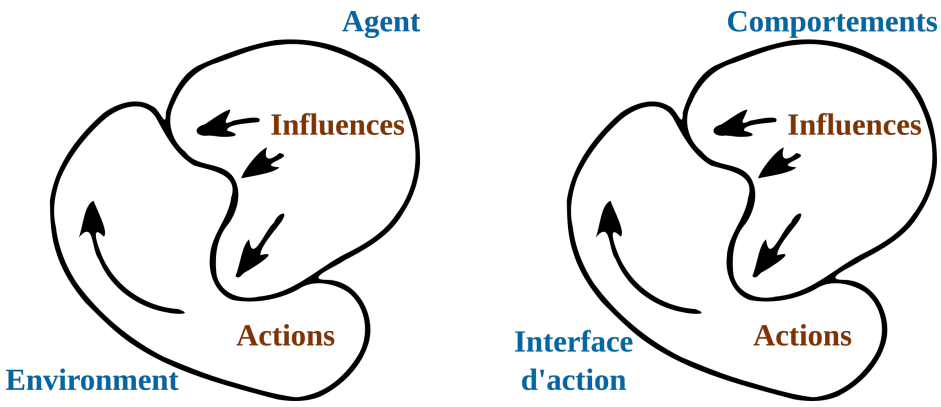


FIGURE 4.2 Modèle Influence Réaction : dans [FM96] (*à gauche*), et dans notre modèle (*à droite*)

considérées comme étant des demandes simultanées et les calculs de réactions sont établis en ce sens. Mais de façon générale, ce modèle exprime surtout le fait que les agents ne sont pas maîtres de l'environnement. Les actions qu'ils souhaitent entreprendre ne doivent pas conduire à modifier directement l'état de l'environnement selon leur volonté. Au contraire, l'environnement dispose de ses propres modalités de fonctionnement et celles-ci s'imposent à l'agent. Comme nous pouvons voir sur l'illustration de gauche de la figure 4.2, l'agent émet trois influences représentées par les trois flèches en direction de l'environnement. On remarque cependant que l'influence au milieu est repoussée tandis que les deux autres arrivent à modifier l'état de l'environnement.

Cette approche évite un certain niveau d'irréalisme dans les simulations [MGF03]. Un des points clés de l'IRM est la soumission des décisions des agents aux contraintes auxquelles ils ne peuvent pas déroger. Les comportements bénéficient ensuite d'une rétroaction (la réaction) sur ce qu'est devenu leur demande (l'influence). En simulation, l'IRM est une interface de filtrage entre les décisions de l'agent et l'environnement.

Cette problématique d'interaction entre les agents et l'environnement apparait dans notre travail (Figure à droite), avec un cadre applicatif réel non simulé. Elle concerne cette fois-ci l'interaction entre les comportements qui souhaitent s'exécuter et les ressources requises pour les réaliser. Ces ressources sont les interfaces d'action. De plus, nous venons de voir précédemment dans 4.2.2 qu'un accès concurrent entre les comportements peut apparaitre. Ce qui rejoint le problème de simultanéité pour lequel l'IRM a montré sa pertinence. Ces deux problèmes que nous rencontrons d'un côté avec l'interaction entre comportements et ressources, et de l'autre côté avec la simultanéité sont similaires à ceux gérés par l'IRM. La nature des problème est la même mais se trouve à l'intérieur de l'agent cette fois-ci. C'est la raison pour laquelle nous réutilisons l'IRM dans l'architecture interne de l'agent.

4.3.2 Modèle formel

Les travaux dans [FM96] donnent une formalisation du modèle influence/réaction. Ce modèle utilise différents notions et paramètres. L'état dynamiques $\delta \in \Delta$ représente l'état de l'environnement qui est défini par les influences émises et les réactions effectuées. Les influences correspondent aux souhaits des comportements qui veulent s'exécuter. Les réactions sont les comportements qui après validation sont transformés en action réelle dans l'environnement. Cette notion d'état dynamique $\delta \in \Delta$ est définie par le couple $< \sigma, \gamma >$, où $\sigma \in \Sigma$ est l'état de l'environnement et $\gamma \in \Gamma$ décrit les influences. La dynamique de l'environnement est une fonction de Δ sur Δ. Elle est représentée par deux fonctions *Exec* produisant les influences, et *React* produisant le nouvel état dynamique. Ces fonctions sont définies par :

$$Exec : \Sigma \times \Gamma \rightarrow \Gamma \; React : \Sigma \times \Gamma \rightarrow \Sigma$$

La combinaison de ces fonctions donne le nouvel état défini par $\delta' = < \sigma', \gamma' >$, tel que :

$$\sigma' = React(\sigma, \gamma) \text{ et } \gamma' = Exec(\sigma', \gamma)$$

Pour passer d'un état à un autre, la fonction *React* est munie d'une fonction *Laws*. Cette fonction est un ensemble formé par des règles, $\lambda \in Laws$. La fonction *React* devient alors :

$$React : Laws \times \Sigma \times \Gamma \rightarrow \Sigma$$
$$React\ (\lambda, \sigma, \gamma) \rightarrow \Sigma$$

4.3.3 La fonction de régulation

Avec le modèle formel précédent, nous avons vu que la fonction *React* porte la loi de l'environnement avec la fonction *Laws*. C'est avec cette fonction *React* que nous spécifierons le contrôle d'applicabilité correspondant au niveau local. Nous appelons "Régulation" la fonction correspondante. Cette fonction a comme rôle de filtrer les influences. Elle soumet les décisions des comportements à des contraintes courantes du contexte d'exécution. Dans la fonction *Laws* nous définissons une règle de gestion des accès aux ressources. La fonction *React* prend en entrée les décisions prises par les comportements. Elle introduit ensuite une modulation de ces comportements en entrée. La modulation prends en compte les évolutions de l'environnement sans qu'il soit nécessaire de les prévoir précisément dès la phase de conception.

4.4 Modèle classique d'agent

Avec la réutilisation de l'IRM dans l'architecture interne de l'agent, nous parvenons à un nouveau modèle agent. Dans cette section, nous développons d'abords le modèle classique

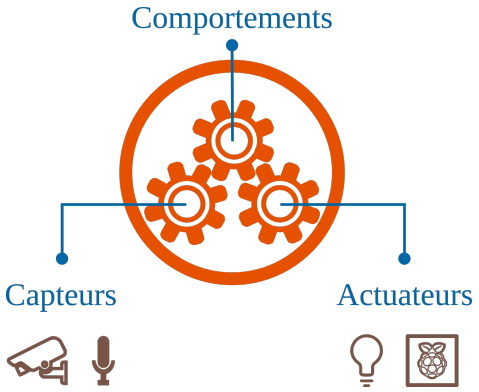


FIGURE 4.3 Modèle classique d'agent

d'agent. L'objectif est de mieux positionner notre nouveau modèle agent incluant l'instance de GMAS dans la section 4.5 suivante.

4.4.1 Anatomie du code de l'agent

Définition théorique

Dans [FM96] un agent est défini comme une entité physique ou virtuelle qui est capable d'agir dans un environnement (i) (...) , qui possède des ressources propres (ii) qui est capable de percevoir son environnement (iii) et qui possède des compétences et offre des services (iv). Nous avons ici une définition conceptuelle d'un agent.

Définition technique

En raisonnant en termes de code informatique nous proposons la définition suivante. Un agent est une entité logicielle. Il est animé par un code informatique. Ce code résulte de l'agrégation d'un ensemble de sous-unités fonctionnelles que nous pouvons grossièrement organiser en 3 sous-ensembles représentés par la figure 4.3.

Les capteurs correspondent aux unités de codes pour la collecte d'information dans l'environnement physique ou l'environnement logiciel (propriété (iii)) dans la définition théorique. Cela peut être par exemple le code de lecture d'un dispositif de mesure de température, ou encore celui qui interprète les échos d'un émetteur/récepteur d'ultrasons pour mesurer une distance.

Les actuateurs correspondent aux unités de code pour la réalisation d'une action dans l'environnement (propriétés (i) et (ii)). Par exemple un code asservissant le moteur d'une roue, ou un code qui règle la température d'un chauffage.

Les comportements sont les unités de code qui a pour objectif d'invoquer une certaine action de l'agent. Ce sont ces actions qui offrent les services (propriété (iv)). Les agents peuvent en posséder un seul ou plusieurs opérant simultanément, tel que décrit dans la

section. Pour la suite, nous employons la formulation : "le comportement de l'agent" pour désigner l'ensemble des unités de comportements présent dans son code.

Synthèse

Avec ces trois codes constitutifs de l'agent, ce qui nous importe est l'identification précise du rôle joué par les différentes portions du code. Ces rôles vont nous servir à décrire de quelles façons les constituants de l'agent s'organisent entre eux. Cela exprime également comment cette organisation opère dans les approches classiques, et de montrer quels sont les changements que nous proposons sur cette organisation au travers de notre modèle d'agent.

4.4.2 De la perception à l'action

Parmi ces trois types d'unités de code, les comportements jouent un rôle central. Les capteurs et les actuateurs sont des éléments de code non proactifs. Au contraire, les comportements sont conçus dans l'optique d'un objectif qui leur est propre. Leur code est élaboré pour atteindre cet objectif. Pour ce faire, le code du comportement réévalue en permanence la situation afin de choisir les nouvelles actions à entreprendre. Aussi, ce code prend le plus souvent la forme d'une boucle. C'est le cas des agents disposant de leur propre fil d'exécution. Le code peut aussi prendre la forme d'une fonction appelée à intervalle régulier. C'est la cas des agents monitorés par un ordonnanceur. Dans une approche classique, le code qui se trouve dans cette boucle, ou cette fonction, suit le déroulement suivant :

— **L'observation de la situation courante** qui comprend : la lecture des réactions de l'environnement suite aux décisions prises antérieurement, la consultation de l'état actuel de l'agent (sa mémoire et ses connaissances) et son évaluation par rapport à l'objectif du comportement, puis des perceptions dans l'environnement pour compléter les données sur lesquelles la prise de décision va s'appuyer.

— **L'analyse décisionnelle** dont la finalité est d'établir la liste des décisions à prendre pour guider les prochaines actions de l'agent.

— **La mise en application des décisions** qui consiste à identifier et activer les actuateurs nécessaires.

Même si dans la réalité certaines phases peuvent s'entrecroiser pour des raisons de simplification d'écriture, du point de vue fonctionnel nous pouvons résumer ce mode opératoire par la chaine d'implications :

$$\text{Perception} \Rightarrow \text{Décision} \Rightarrow \text{Action}$$

4.4.3 Les limites de l'analyse décisionnelle

La phase d'analyse décisionnelle, encore appelée "prise de décision", a fait l'objet de nombreux travaux, comme ceux cités dans la section 2.1 [BSG+09] [ST09]. Il existe maintenant des approches performantes pour élaborer des agents dits "cognitifs" qui sont capables de déployer une pertinence d'analyse remarquable. Mais nous pensons que si effectivement ces contributions sont efficaces pour doter les agents d'une proto-intelligence, ce n'est pas au niveau de cette analyse décisionnelle que se situe l'enjeu de l'adaptabilité.

En effet, l'imprévisibilité de l'environnement physique fait qu'il peut potentiellement exister une multitude de raisons susceptibles d'interférer avec une décision à prendre. Concevoir un algorithme capable de gérer cette multitude de cas aboutirait à une analyse exhaustive de ces cas (si tant est que cela soit possible), contre finalement peu de lignes de code pour traiter effectivement l'objectif du comportement. Ainsi, dans une telle approche, doter l'agent d'une aptitude d'adaptabilité conduirait à complexifier fortement le code de son comportement avec l'analyse des situations possibles. Néanmoins, cela n'offre pas la garantie qu'une situation contradictoire non prévue ne se présentera pas le moment venu.

Raisonnablement, il nous faut acter le fait que l'incapacité d'envisager l'intégralité des possibles lors de l'analyse décisionnelle est une réalité incontournable. Ce n'est donc pas une chose qu'il faut corriger, mais plutôt un fait avec lequel nous devons composer. Pour autant, face aux situations imprévues les plus critiques, il n'en reste pas moins vrai que les décisions prises se verront parfois être non applicables.

4.5 Modèle agent avec régulation

4.5.1 Agent avec régulation

Nouvelle anatomie de l'agent

Comme nous avons vu précédemment, dans une approche classique, doter l'agent d'une aptitude d'adaptabilité conduirait à complexifier fortement le code de son comportement. Le code supplémentaire qui réalise l'analyse des situations possibles, et qui ajuste les décisions en conséquence, est ce que nous appelons "un code de régulation". Ce code de régulation est une spécialisation de contrôle d'applicabilité. Nous y externalisons la portion de code destinée à gérer les comportements en fonction des situations possibles. Le but est de faire évoluer l'analyse de façon indépendante de celui du comportement.

Le rôle du code de régulation est d'interfacer les décisions prises par les comportements avant leur application sur les actuateurs. Cet interfaçage a pour objectif d'éviter que, face à une configuration du contexte non prévue par le comportement, la décision prise par ce dernier conduise à une action inappropriée pour l'agent. En ce sens, l'analyse établie dans

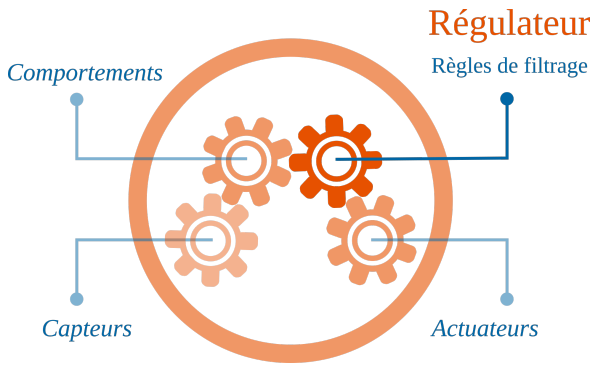


FIGURE 4.4 Modèle agent avec régulateur

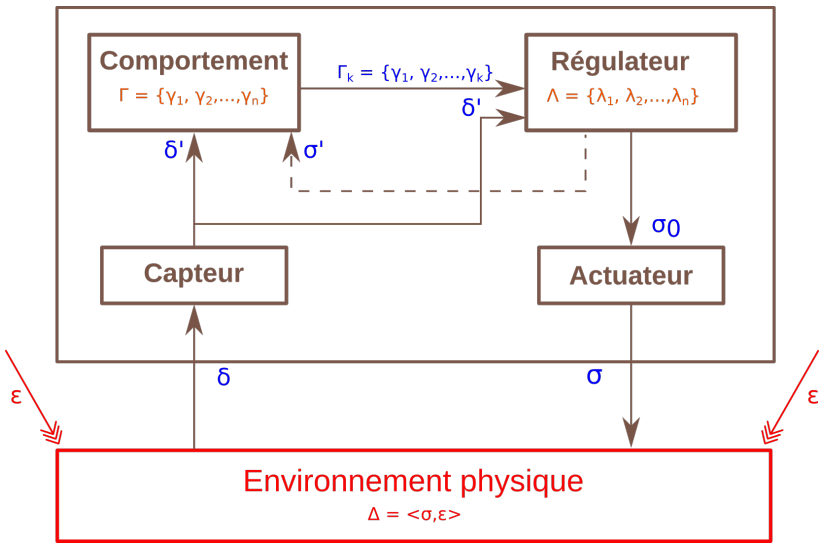


FIGURE 4.5 Instance de GMAS correspondant au niveau individuel

cette régulation ne repose pas sur un objectif particulier à atteindre (comme c'est le cas dans l'analyse faite par les comportements), mais se concentre essentiellement sur le respect de règles délimitant l'usage des actuateurs en fonction de la situation environnante.

Nous avons ainsi une nouvelle structuration de l'anatomie de l'agent caractérisée par l'unité de régulation. L'architecture résultante est représentée par la figure 4.4

L'instance de GMAS au niveau individuel est représentée par la figure 4.5 ; Nous retrouvons les mêmes briques correspondant à celles du pattern génériques GMAS dans la figure 3.5 précédente.

Cette structuration apporte un nouveau fonctionnement interne au sein de l'agent. Pour avoir une information sur l'environnement, l'unité *Comportement* envoie une demande de perception auprès du *Capteur* (1). L'unité *Comportement* émet ensuite des influences à destination du centre de régulation (2). Une influence est constituée par deux composants : l'action demandée et l'*Actuateur* ciblé. Le centre de *Régulation* regroupe l'ensemble des unités

de régulation. Chacune de ces unités filtre les influences reçues (3). Celles qui sont validées donnent lieu à l'activation des actuateurs (4). Dans le même temps, le comportement à l'origine de l'influence est informé des résultats de ce traitement par une réaction interne (4'). Cette régulation représente la fonction *React* définie dans 4.3. Les régulateurs qui y participent spécifient chacun l'une des règles de la fonction *Laws*. Notons qu'ici ces règles sont dynamiques dans la mesure où un régulateur peut émettre des requêtes sur d'autres composants. Cela peut être par exemple une demande de détection d'obstacle à un capteur de proximité ou une demande de disponibilité d'un actuateur ciblé par une influence.

La régulation conduit à deux finalités possibles pour une décision donnée :

— Soit cette décision est jugée comme applicable et donnera lieu aux demandes d'actions correspondantes sur les actuateurs concernés. On parlera alors de "validation".

— Soit un ou plusieurs régulateurs, au vu du contexte courant, auront identifié une contradiction de la décision avec la règle d'intégrité qu'ils supervisent. Dans ce cas la décision ne donnera lieu à aucune sollicitation d'actuateur, et on parlera d' "invalidation".

Régulation avec l'IRM

Pour des agents évoluant dans un environnement ambiant, l'IRM n'a pas besoin d'être implémenté. L'environnement ambiant impose de façon naturelle ses modalités de fonctionnement à l'artefact matériel animé par l'agent. Le champ d'action de ce dernier, et plus précisément de son comportement, se limite donc à activer ses actuateurs. Dans ce contexte, il ne peut pas pleinement maîtriser l'incidence de ces manipulations sur l'environnement ambiant qui l'entoure.

Cependant, face à l'enjeu de l'adaptabilité, il nous est apparu que même la maîtrise de l'activation des actuateurs devrait être retirée du champ d'action du comportement de l'agent. Le but est de permettre l'adjonction d'un niveau de régulation dont le rôle est de s'assurer de l'adéquation entre les actions demandées et la situation environnante. C'est pour créer ce découplage "comportement-actuateur" que nous réutilisons l'IRM mais cette fois au niveau de l'architecture interne de l'agent. La réutilisation de l'IRM apporte à l'agent deux types de réactions en fonction de l'applicabilité de l'influence.

— La réaction externe correspond à la réaction de l'environnement face à la transformation de l'influence en action. C'est le seul niveau de réaction qui existe dans l'approche d'action classique. L'influence donne directement lieu à l'action.

— La réaction interne correspond à l'invalidation de l'influence par le code de régulation. Dans ce cas, l'influence non validée ne donne pas lieu à une action dans l'environnement.

4.5.2 Régulation et analyse décisionnelle

Les difficultés de couvrir l'espace des possibles, que nous avons soulevées dans 4.4.3, au niveau de l'écriture du code des comportements se trouvent donc déportées maintenant au niveau de l'écriture du code des régulateurs. Toutefois, nous avons une différence majeure par rapport au modèle classique. Dans notre modèle, le code de régulation n'est pas parasité par la nécessité d'atteindre un objectif en particulier. Cette fois le code en question n'est là que pour veiller, de façon objective, à ce qu'aucune action "néfaste" ne soit déclenchée. Les possibilités d'actions étant limitées par l'étendue des capacités des actuateurs. Le nombre de vérifications à entreprendre peut-être considérablement réduit et plus facilement identifiable. Pour les agents les plus complexes, la complexité d'écriture du code de régulation restera conséquente. Il s'agit des agents disposant de nombreux actuateurs, avec de nombreux comportements et évoluant dans un environnement fortement dynamique. Plus encore que la variabilité de l'environnement, le nombre de facteurs de sureté à prendre en compte complexifie la tâche de la régulation. Analysons dans l'exemple suivant comment la régulation gère ce genre de problème.

Exemple de comportements complexes

Revenons sur l'exemple de notre drone agricole pour illustrer l'analyse décisionnelle avec la régulation pour le cas d'un comportement complexe. Selon le modèle, un drone peut présenter comme comportement la pulvérisation ciblée de fertilisation ou d'herbicide. La pulvérisation consiste à identifier la zone à traiter et d'y pulvériser ensuite la quantité de liquide correspondante. La pulvérisation est assez complexe dans la mesure où elle mobilise plusieurs interfaces d'action. Elle nécessite également une précision tant sur la surface à traiter que sur la quantité à verser. De plus, plusieurs contraintes extérieures doivent également être prises en compte. Nous pouvons citer : l'évitement des obstacles lors du déplacement, la quantité de produit à pulvériser, l'énergie nécessaire de déplacement. Avec notre modèle de régulation, notre objectif est d'identifier les règles d'utilisation des interfaces d'action au lieu de faire l'inventaire des situations possibles. Pour ce faire, nous subdivisions le comportement de pulvérisation en trois sous-comportements ayant chacun un objectif propre :

1. Se déplacer
2. Effectuer un balayage topographique
3. Pulvériser

Ensuite pour chaque comportement, nous identifions les interfaces d'action sollicitées afin de dégager leur condition d'utilisation. Le tableau 4.1 montre cette subdivision. Chaque numéro dans le tableau correspond au numéro de comportement que nous avons cité précédemment.

	Objectif	Interfaces d'action	Condition d'utilisation
1	Survoler la zone agricole	Batterie pour l'énergie L'ensemble du drone	Energie suffisante Eviter les obstacles
2	Identifier la zone à traiter	Telemetre laser ultrasonique	Eviter la forte humidité
3	Traiter la zone identifiée	Buses de pulvérisation	Produits suffisants Distance de sécurité

TABLE 4.1 Exemple de comportement : pulvérisation ciblée de fertilisant ou d'herbicide

En subdivisant le comportement de *pulvérisation*, nous réalisons que les deux premiers : *Se déplacer* et *effectuer le balayage* sont nécessaires mais ne sont pas propres à la *pulvérisation*. En effet, ces deux comportements sont utilisés par tout autre comportement nécessitant une cartographie du champ de culture. Ils peuvent donc être ré-utilisés par d'autres comportement sans avoir à les re-écrire. Seul le troisième comportement est propre à la pulvérisation. Ce comportement répand la quantité de liquide nécessaire sur la zone identifiée. Pour la sécurité du drone et du champ, ce comportement nécessite une certaine distance entre les buses et les plantes. Toutefois, avant d'appliquer le produit, pour éviter une erreur de dosage, le drone doit vérifier si la quantité de produit restant est encore suffisante.

En identifiant les sous-comportements nous parvenons à identifier les différentes conditions d'utilisations des interfaces sollicitées. Nous pouvons alors définir une règle pour chaque condition. Ce qui constitue la base de règle de notre unité de régulation pour ce comportement. Lorsqu'une des conditions n'est pas vérifiée, le comportement ne pourra pas être réalisés.

Synthèse

Nous montrons avec l'exemple précédent comment l'unité de régulation gère les comportements complexes. Cela consiste à subdiviser le code du comportement complexe en plusieurs unités. On subdivise ensuite le code de régulation en plusieurs unités de régulations plus simples. De la même façon que l'atteinte d'un objectif particulier constitue la raison d'être d'une unité comportement, les unités de régulation pourront être chacun élaborées sur la base d'une règle d'intégrité particulière. Par exemple, pour le sous-comportement *Pulvériser* l'objectif particulier est de *traiter la zone identifiée*. Nous avons deux règles d'intégrité particulières qui sont : la *distance de sécurité entre les buses et les plantes*, puis la *quantité de produit suffisante*. L'ensemble des unités de régulation est sollicité à chaque fois qu'un flux de décision est produit par le(s) comportement(s) de l'agent. Cette sollicitation est effectuée par un déclenchement des unités de régulation en série, de sorte que la régulation puisse être hiérarchisée en fonction de l'importance des règles d'intégrité de chacun des régulateurs. Toujours avec notre exemple, cette hiérarchie d'importance des règles est la même que l'ordre des conditions d'utilisation dans le tableau 4.1. En effet, dès que la première condition

sur le niveau d'énergie n'est pas vérifiée, l'influence correspondant au comportement de déplacement sera rejetée. Ce qui implique que les deux autres sous-comportements ne seront pas sollicités.

Dans le meilleur des cas, la subdivision réduit la redondance dans l'analyse décisionnelle. En effet, comme nous avons vu avec notre exemple, certain sous-comportement peut être ré-utilisé par d'autres comportements.

4.5.3 Couplage avec l'environnement

L'interaction de l'agent avec son environnement fait partie des éléments à prendre en compte dans le modèle agent. Considérons deux modèles agent en SMA pour analyser le lien entre le modèle agent et l'environnement. Il s'agit d'un modèle utilisé en simulation et d'un modèle utilisé en environnement ambiant.

Modèle en simulation

Notre approche orientée comportement fait que nous avons un agent avec des comportements multiples et indépendants. L'agent avec ses comportements multiples doit tenir une activité globale cohérente. Il est nécessaire que l'activité produite par l'exécution parallèle des comportements soit, d'une façon ou d'une autre, coordonnée avant de donner lieu aux actions à opérer dans l'environnement. Dans le contexte des simulations, [PCSR06], utilise l'environnement comme point de couplage pour répondre à ce problème. L'environnement y est structuré en dynamique relative à chaque comportement. Une telle approche n'est pas toujours applicable pour le cas d'un environnement ambiant.

Modèle pour environnement ambiant

Nous avons toutefois identifié un cas où sous certaines conditions cela est possible. Il s'agit des travaux de [WOO07] [Har08]. Dans ce travail, ils rajoutent une couche d'environnement virtuel local aux agents. L'architecture résultante est utilisée pour animer des véhicules automatiques appelés AGV (*Automatic Guided Vehicule*) pour le transport de charges dans un entrepôt un environnement réel, donc non simulé. Elle est représentée par la figure 4.6 suivante :

La couche d'environnement virtuel est distribuée entre les AGV. Le couplage entre l'AGV et l'environnement virtuel est réalisé avec les fonctions de perception, d'action et de communication.

Synthèse

Dans les deux cas, nous identifions le besoin de couplage entre l'agent et l'environnement. Nous parvenons à gérer ce problème de couplage avec la régulation. Le boucle de

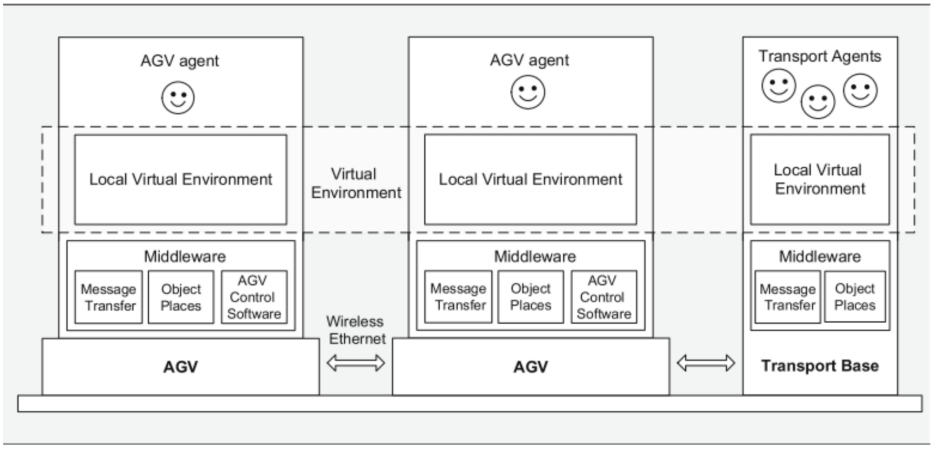


FIGURE 4.6 Environnement virtuel pour un agent dans un système de transport AGV [WOO07]

rétroaction apporte une cohésion entre l'environnement et l'agent. Comme chaque agent a ses propres interfaces de perception, ils ont chacun leur propre représentation de leur environnement local. Leurs réactions correspondantes sont aussi propres. Ainsi avec la régulation le couplage entre l'environnement et l'agent est géré du côté de l'agent. Dans les deux modèles précédents, ce couplage est géré uniquement au niveau de l'environnement. Pour le premier, l'environnement est structuré pour chaque comportement. Pour le deuxième, l'environnement est représenté virtuellement au sein de l'agent.

4.6 Analyse qualitative

Nous présentons dans cette section une analyse qualitative de notre modèle. Dans les deux premières sous-sections, nous faisons une étude comparative avec les modèles existants. Ensuite, dans les deux dernières, nous présentons les avantages supplémentaires par rapport à celles de GMAS que nous avons citées dans 3.5.

4.6.1 Similarité avec l'approche émergence/contrainte

Notre approche présente ainsi des similarités avec l'approche émergence/contrainte dans [MDSM12] qui utilise également le modèle influence/réaction pour un système à plusieurs niveaux. L'émergence et les contraintes sont deux formes particulières d'influence. L'interaction entre les agents dans le niveau micro conduit à une émergence dans le niveau macro, tandis que les contraintes résultant de cette émergence au niveau macro impactent le niveau micro. Dans ce travail, le système laisse se produire les situations inappropriées et ce sont les comportements ensuite qui les résolvent après-coup. Dans notre approche, c'est avant d'être exécutées que les influences produites par les comportements sont filtrées par le

régulateur de l'agent. De ce fait, les comportements ne résolvent pas les problèmes car le régulateur se charge de les éviter.

4.6.2 Vers une intelligence comportementale

Les travaux dans [Has03] nous donnent les principes caractérisant l'intelligence comportementale. Il s'agit de l'absence d'une représentation symbolique de l'environnement, la mise en situation, l'incarnation, la boucle "sensori-motrice" qui guident l'agent. Nous retrouvons ces principes dans notre modèle. Comme nos équipements sont des composants à faible capacité de calcul, nous ne pouvons pas avoir une représentation symbolique de l'environnement au sein de nos agents. Nous avons vu dans le chapitre 3.1.2 précédent qu'au niveau local, nous avons des agents associés à des composants à la fois situés et incarnés. Nos agents ne possèdent pas non plus une capacité cognitive pour délibérer sur ses actions. En remplissant ces critères, nous affirmons que notre proposition de modèle d'agent avec régulation vérifie les principes menant vers l'intelligence comportementale.

4.6.3 Une évolution incrémentale

La séparation entre l'objectif et les règles apporte une souplesse sur l'évolution de la flexibilité de l'agent. Les adjonctions d'analyse s'effectuent au niveau du code de régulation. Il est donc possible d'ajuster dynamiquement l'intensité de cette analyse au cours du temps en fonction de l'évolution de la criticité du contexte.

Revenons à notre exemple de robot pour illustrer cette propriété. Le champ agricole dans lequel le robot se déplace peut inclure ou pas des humains. Dans ce genre de situation, nous pouvons définir deux types de régulation pour le robot. Pour un cadre opératoire à présence humaine, la vitesse de déplacement sera systématiquement amoindrie et plafonnée. Dans le cas contraire où aucun humain n'est présent, ce niveau de régulation peut être relâché. Il est ainsi possible de faire l'économie de l'analyse précédente sur la vitesse. La flexibilité du robot évolue ainsi sans qu'on modifie le code de son comportement.

4.6.4 Vers une adaptation collective

La régulation constitue un point de jonction de premier choix pour coupler les activités des agents au niveau collectif. C'est dans cette direction que nous poursuivons nos travaux en vue de proposer un modèle d'architecture globale de SMA adaptatif. Cette jonction nous aiguille vers une flexibilité individuelle des agents en fonction de considérations collectives.

Considérons maintenant l'exemple où nous avons une file de quelques robots qui se déplacent en formation linéaire. Le code de comportement du robot n'est pas forcément conçu pour prendre connaissance des robots qui se déplacent avec lui. Cependant dans un collectif, on cherche à avoir un comportement collectif. Dans notre cas, on évite le décrochage

entre les robots au cours de leur déplacement. Au lieu de modifier le code de comportement de chacun des robots pour éviter ce décrochage, il est possible de n'agir que sur la régulation du robot de tête. Sa vitesse sera automatiquement ralentie afin d'éviter les décrochages. Ce robot de tête n'a pas besoin d'intégrer l'information qu'il remplit cette fonction collective. Pour autant la régulation de ses actions fait qu'il agit bien en conformité avec cette fonction. Nous reviendrons sur cette relation entre niveau local et niveau global dans la partie 5.5.2 du chapitre suivant.

4.7 Conclusion

L'efficacité d'un logiciel destiné à opérer dans un environnement physique dépend de sa capacité d'adaptation. La diversité des paramètres à prendre en compte dans l'objectif d'une adaptabilité performante fait des SMA une approche prometteuse. Pour autant, cette faculté adaptative est essentiellement traitée comme une propriété globale. Or nous pensons qu'un renforcement de cette aptitude au niveau individuel peut démultiplier le potentiel atteignable au niveau collectif. Comme nos composants sont miniaturisés et limités en terme de stockage et de calcul, nous choisissons des méthodes moins complexes pour donner une flexibilité au niveau individuel. Nous proposons pour cela MECA, un modèle d'architecture interne d'agent centré sur un processus de filtrage de l'expression de ses comportements primaires. A cet effet, nous réemployons le modèle influence/réaction pour interfacer les décisions prises par les comportements afin d'introduire une modulation du comportement apparent de l'agent. Une modulation capable de prendre en compte les évolutions de l'environnement sans qu'il soit nécessaire de les prévoir précisément dès la phase de conception. Pour faciliter sa mise en application, nous implémentons notre architecture MECA sous la forme d'une librairie Java que nous présentons dans le chapitre 6. Ce modèle d'architecture a fait l'objet d'une publication dans [VPC16]. Avant de décrire son implémentation, nous continuons la modélisation d'architecture avec l'instanciation du modèle GMAS au niveau global dans le chapitre 5.

Chapitre 5

Adaptabilité collective

Ce chapitre décrit l'instance de GMAS correspondant au niveau global pour une adaptation collective. Nous réutilisons l'expression "adaptation" ici parce que contrairement à l'adaptation individuelle, l'adaptation collective ne nécessite pas une forte capacité de raisonnement. Elle peut émerger de l'interaction et de l'organisation entre les composants. Ces deux aspects constituent la force du collectif. Notre proposition de modèle d'architecture s'inscrit dans ce contexte. L'objectif est ici d'optimiser les activités collectives. Pour cela, nous définissons des modalités opératoires à la place du contrôle d'applicabilité. Son rôle est de vérifier la cohérence d'un comportement par rapport à l'objectif collectif.

Nous définissons le niveau global dans la section 5.1 pour décrire le cadre applicatif qui nous intéresse. Cette définition nous conduit ensuite à l'identification des problèmes du niveau global dans la section 5.2. Nous décrivons l'architecture répondant à ces problèmes dans la section 5.3, avant de la formaliser dans la section 5.4. Pour évaluer son efficacité, nous proposons une analyse qualitative dans la section 5.5 et une analyse quantitative dans la section 5.6. Nous concluons avec la section 5.7.

5.1 Le niveau collectif

5.1.1 Définition

Dans ce travail, nous utilisons la notion de *niveau global* pour désigner un ensemble de composants collectifs. Cet ensemble a des particularités. Les différents types de systèmes collectifs suivants nous les décrivent.

Définition 5.1.1. Un SMA est défini comme un système composé par un environnement E, un ensemble d'objets O, un ensemble A d'agents qui sont des objets particuliers, et un ensemble de relations R qui unissent les objets [Jac95].

Définition 5.1.2. Un Système Collectif Adaptatif est défini comme un ensemble d'unités (nœuds) hétérogènes avec leurs propriétés individuelles, leurs propres objectifs et leurs

propres actions [CH15]. Ces unités interagissent entre elles et sont connectées au sein d'un réseau dans un environnement dynamique.

Définition 5.1.3. Un système informatique pervasif (pervasive computing) est défini dans [ZOA+15] comme une population de composants autonomes qui interagissent entre eux. Cette population conduit à l'émergence d'une infrastructure dense et décentralisée.

Avec ces trois définitions, nous identifions deux éléments déterminants d'un système collectif : l'ensemble de composants et l'interaction entre eux. Ces trois définitions supposent également que chacun des composants est autonome. En effet, cette autonomie est évoquée explicitement dans la définition 5.1.3 . Elle est décrite par les propriétés, objectifs et actions propres dans la définition 5.1.2 . Dans la définition 5.1.1, elle est implicite dans la notion d'agent. Comme nous avons vu précédemment dans le chapitre 4.1.2, un agent est une entité autonome.

Nous désignons donc notre niveau global par le collectif formé par des composants autonomes qui interagissent entre eux dans un environnement ambiant. Chaque composant autonome correspond au niveau local que nous avons développé dans le chapitre 4.

Le chapitre 3 précédent montre les deux forces du collectif qui sont l'organisation et l'interaction. Analysons de près maintenant la pertinence de ces deux forces du collectif.

5.1.2 Interaction

La notion d'interaction est déterminante dans la définition d'un collectif. En effet, nous remarquons qu'avec ces définitions, le collectif formé par les composants ne se limite pas à une simple collection. L'ensemble est caractérisé par l'interaction entre les composants. Cela signifie que les composants peuvent communiquer entre eux. Ils peuvent aussi partager des données ou informations pour atteindre leurs objectifs. Dans [Vin14], l'interaction est dite riche, non linéaire, et généralement mais non pas exclusivement, avec le voisin immédiat. L'interaction peut également être compétitive ou coopérative [CH15].

5.1.3 Organisation

Dans [Jac95] l'organisation constitue avec l'interaction le concept de base des SMA. Une organisation suppose "qu'il existe un ensemble d'entités formant une certaine unité et dont les différents éléments sont subordonnés entre eux dans un ensemble solidaire et dans une activité convergente (...) Il existe de nombreuses interrelations entre les agents, par le biais de délégation de tâches, de transfert d'informations, d'engagements, de synchronisations d'actions".

Nous retrouvons ces interrelations dans notre travail. Il s'agit principalement du transfert d'informations et de l'ajustement des comportements collectifs par rapport à ces informations.

Différence entre système collectif et système non collectif

Compte tenu de ces propriétés, nous pouvons vérifier si un système est collectif ou non. Si chacun des composants d'un système réalise chacun ses objectifs de manière isolée des autres, le système n'est pas collectif. Il ne l'est pas non plus si les composants sont ni visibles, ni accessibles entre eux. Le système est collectif uniquement lorsque les composants interagissent et le contrôle est distribué grâce à l'organisation [SM09]. Considérons l'exemple suivant pour illustrer un ensemble non collectif.

La domotique utilise souvent des équipements connectés qui ne sont pas forcément collectifs. Considérons par exemple un système composé d'une caméra, d'un portail, des éclairages, d'une climatisation, d'un volet, et des rideaux métalliques qui sont tous automatiques. C'est un ensemble de composants autonomes. Cependant, chaque composant a ses propres objectifs et les réalise sans interagir avec les autres : la surveillance pour la caméra, l'ouverture et la fermeture automatique pour le portail, et aussi pour le rideau et les volets, le contrôle de la température pour la climatisation. Dans ce type de système, la coordination de l'ensemble est centralisée auprès de l'utilisateur. Par conséquent, ce type de système ne favorise pas l'interaction entre les composants. Il n'est donc pas un système collectif.

5.1.4 Synthèse

Cette section nous montre l'importance de l'organisation et de l'interaction dans le collectif. Dans le chapitre 3 nous avons vu que ces deux forces du collectif peuvent être à tout moment perturbées par les événements imprévisibles de l'environnement ambiant. Nous avons ensuite fixé l'objectif de maintenir l'organisation entre les composants et de leur assurer une interaction. C'est dans ce contexte que nous proposons une solution d'adaptation en proposant une instance de GMAS.

5.2 Illustration du problème

Dans cette section, considérons un exemple concret de structure sociale pour illustrer ce qu'est une activité collective. Notre objectif est de parvenir à une généralisation et une formalisation du problème. Nous choisissons comme exemple le rechargement de véhicule électrique avec une borne publique. Cet exemple nous intéresse dans la mesure où il met en évidence les problèmes caractéristiques d'une activité collective.

5.2.1 Contexte

La recharge de véhicule électrique est constituée par deux composants principaux. Il s'agit des voitures électriques et de la borne de recharge. Ces composants sont autonomes et indépendants. La voiture électrique a comme objectif de faire le plein d'énergie. Tandis que la

borne a comme objectif de recharger chaque véhicule qui la sollicite. Avec cette configuration, nous ne pouvons pas encore parler de système collectif. En effet, la seule interaction entre les deux composants se résume au chargement du véhicule par la borne de recharge. Dans ce contexte, il n'y a aucune interaction au préalable entre les composants, ni de transfert d'informations, ni un ajustement du comportement de l'un par rapport aux comportements de l'autre.

Cette configuration sans dimension collective représente généralement la situation actuelle existante en terme de recharge de véhicule électrique. Elle est efficace mais toutefois nous avons identifié des contraintes qui font que le fonctionnement peut être parfois perturbé.

Les bornes de recharge sont limitées. le nombre de véhicules est largement supérieur au nombre de points de recharge. La moyenne nationale en France est de 1 point de recharge public pour environ 5,7 voitures [1]. Ce nombre peut toutefois augmenter en fonction de la situation géographique de la borne. Cela implique que pour un véhicule, la disponibilité d'une borne n'est pas toujours garantie à tout moment parce que d'autres peuvent l'utiliser. Cette situation a des répercussions tant au niveau de la borne qu'au niveau des véhicules. Pour la borne, une forte demande de rechargement augmente le nombre de voitures en attente. La borne aura du mal ainsi à atteindre son objectif qui est de recharger toutes les voitures. Du côté des voitures électriques, le chargement nécessite une certaine durée minimum. Si nous considérons quelques voitures déjà en attente de chargement, le temps d'attente minimum de la prochaine voiture va être multiplié par le nombre de véhicules dans la file d'attente. Dans ce sens, ce temps d'attente présente pour la voiture un coût de rechargement. Ce qui constitue une contrainte pour atteindre son objectif de se recharger.

5.2.2 Vers un système collectif

Nous avons donc deux obstacles au bon déroulement du rechargement : d'un côté, la limitation du nombre de borne disponible et de l'autre côté le temps d'attente requis pour le chargement. Nous estimons qu'il est possible de limiter l'impact de ces obstacles sur leur objectif. Pour cela, nous nous proposons de les faire collaborer au sein d'un même collectif. L'idée est de permettre à chaque composant de tirer profit du collectif, donc de l'organisation et de l'interaction, pour mieux atteindre l'objectif. Ce collectif aura ensuite un objectif commun qui est l'optimisation du rechargement des véhicules. Pour la borne cet objectif limite le nombre de véhicules en attente. Pour le véhicule, cet objectif réduit le temps nécessaire pour le chargement.

Maintenant que nous avons l'objectif collectif, décrivons maintenant les moyens d'organisation et d'interaction à utiliser. Pour l'interaction, nous proposons un partage d'informations

1. https://www.ecologique-solidaire.gouv.fr/developpement-des-vehicules-propres

entre la borne et le véhicule. En ce sens, différentes applications comme NextCharge [2] ou ChargeMap [3] existent déjà pour les automobilistes. Elles fournissent des informations sur la localisation de la borne. Une fois en charge, l'application donne l'état du chargement : niveau d'énergie, temps restant. Toutefois les informations sur le nombre de véhicules en attente, et donc le temps d'attente estimé avant d'être rechargé, ne sont pas disponibles dans les applications que nous avons identifiées. Ces informations sont pertinentes dans la mesure où elles peuvent modifier le comportement de la borne et des véhicules. Pour que cette information soit disponible, il faut donc que la borne la mette à disposition. Nous n'avons identifié aucune borne avec une telle fonctionnalité. Nous proposons alors du côté de la borne, un système de planning qui affiche le nombre de véhicules en cours avec le temps d'attente estimé. En résumé, pour favoriser l'interaction entre le véhicule et la borne, il faut un système de mise à disposition de l'état courant du chargement.

Une fois l'interaction établie, nous nous intéressons maintenant à l'organisation entre les composants. Cette organisation découle de l'objectif collectif qui est d'optimiser le processus de chargement. Chaque composant a ensuite des actions à entreprendre pour atteindre cet objectif. La borne est chargée d'accélérer le temps de charge de chaque voiture lorsque le nombre de voitures en attente atteint un certain seuil. Deux solutions sont possibles, soit la borne inhibe le mode de chargement normal et bascule exclusivement en mode semi-rapide ou rapide, soit il reste en mode normal mais limite chaque chargement à 80 %. Une des caractéristiques des véhicules électriques est que la batterie atteint 80% de sa capacité de chargement après environ 50 % du temps de charge [4]. De son côté, le véhicule doit éviter les longues files d'attente. Deux solutions s'offrent également à lui. La première consiste à ne pas se recharger quand il y a une forte demande. La deuxième consiste à être plus économe une fois que la charge est limitée à 80%.

En résumé, nous avons une cohésion de l'ensemble de ces activités collectives à gérer d'un côté, et de l'autre côté nous avons à maintenir une stabilité en terme de temps de chargement. Cela rejoint la problématique principale de cette thèse 1.2. Pour la suite, nous généralisons ce problème pour mieux concevoir la solution.

5.2.3 Généralisation du problème

Cet exemple simple nous montre les aspects des éléments clés à gérer dans les activités collectives en général. Ces aspects sont : la gestion d'une ressource limitée, le besoin d'un support d'interaction pour échanger les informations et la distribution du contrôle de

2. https://nextcharge.network/
3. https://fr.chargemap.com/
4. https://evbox.fr/apprendre/faq/temps-de-recharge

l'ensemble. Pour ce qui suit, nous nous intéressons à chacun de ces éléments. Notre objectif est d'identifier les spécifications requises pour concevoir une architecture adéquate.

Besoin de gestion de ressources limitées

Dans notre exemple, nous identifions deux types de ressources à gérer : la borne et le temps. La première ressource est la borne de chargement. Nous avons vu que le nombre de bornes est inférieur au nombre de véhicules. Une borne est alors partagée entre plusieurs véhicules. La deuxième ressource est le temps que le véhicule dépense avant de se charger et ensuite pour se charger. Contrairement à la première ressource, le temps n'est pas une ressource à partager. C'est plutôt une ressource à minimiser.

Dans une activité collective en général nous retrouvons ces deux types de gestion de ressource. Soit il s'agit de la gestion d'un accès concurrent à une ressource limitée (comme le cas de la borne avec les véhicules), soit il s'agit d'une optimisation de l'emploi d'une ressource (comme le cas du temps d'attente pour les automobilistes). Cette gestion de ressource est déterminante dans un collectif parce que dans la plupart des cas c'est ce qui définit l'organisation de l'ensemble. Des modalités opératoires sont ainsi définies pour une activité collective afin de mieux gérer les ressources.

Besoin d'un support d'interaction

Nous avons vu précédemment que, tant que les véhicules électriques et la borne ne peuvent pas échanger des informations, il n'est pas possible d'optimiser le chargement.

Pour permettre une activité collective cohérente, l'architecture qui la supporte doit avant tout permettre le partage d'informations. Pour cela, il faut un outil donnant accès aux informations relatives à l'évolution de l'activité collective.

Besoin de distribution du contrôle

Dans l'exemple du chargement des véhicules, nous sommes passés d'un système non collectif à un système collectif. Ce passage apporte un nouvel objectif et une nouvelle organisation. Toutefois, pour réaliser cela aucun nouvel élément n'est ajouté dans le système pour assurer les nouvelles fonctionnalités collectives. Toutes ces nouveautés sont gérées au niveau de la borne et du véhicule.

L'architecture que nous souhaitons mettre en place doit donc tenir compte de cet aspect distribué. Cela revient à dire que le contrôle ne doit pas être centralisé dans un nouveau composant ou un autre déjà existant. Chaque composant participe au contrôle de l'ensemble.

Dans cette section nous sommes partis d'un cas particulier pour parvenir au problème général. Cette section nous montre les exigences d'une activité collective. Avec ces exigences, décrivons maintenant les spécifications pour notre proposition d'architecture.

5.3 Spécification de l'architecture

Dans cette section, nous revenons sur chacune de ces exigences pour concevoir la solution correspondante.

5.3.1 Modalités opératoires

Nous avons vu précédemment que la gestion d'une ressource limitée requiert une organisation. Cette dernière implique la définition des modalités opératoires qui régissent les composants dans leur activité collective. C'est à ce niveau que le contrôle d'applicabilité que nous avons décrit dans le chapitre 3 est pertinent. Pour rappel, son rôle est d'analyser les comportements pour vérifier si l'état de l'environnement extérieur est favorable à leur exécution. Les modalités opératoires contraignent l'exécution des comportements, tout comme l'environnement extérieur. Par conséquent, la vérification de ces modalités opératoires est similaire à la vérification de l'état de l'environnement. Ces modalités opératoires forment donc au niveau global le contrôle d'applicabilité des comportements. Avant de spécifier ces modalités opératoires, analysons d'abord le type d'environnement sur lequel s'appuie ici la vérification.

L'environnement

Avec le niveau local dans le chapitre 4 précédent, nous avons un environnement physique. L'applicabilité regroupe alors les règles relatives à l'intégrité des interfaces d'action et celle de l'exécution. La dimension physique est gérée au niveau local. Ce qui fait que l'environnement, donc le milieu cible, que nous avons au niveau global n'est pas un environnement physique. Pour identifier la nature de cet environnement, revenons à la définition du contexte d'exécution dans le chapitre 3.1. Nous avons vu que le contexte d'exécution représente l'ensemble des références spatiales et temporelles. Si la référence spatiale renvoie à l'environnement physique, la référence temporelle renvoie à l'ensemble des activités qui évoluent dans le temps. Au niveau global, nous avons un objectif collectif et des activités collectives. De ce fait, nous observons particulièrement l'évolution de ces activités. L'environnement dans lequel évoluent ces activités est donc un environnement fonctionnel. Dans notre exemple précédent cet environnement fonctionnel est l'ensemble des activités de la borne et des véhicules. Ces activités vont vers l'objectif commun qui est l'optimisation du rechargement.

Contrôle d'applicabilité au niveau collectif

Nous avons vu que le contrôle d'applicabilité s'appuie sur l'état courant de l'environnement. Comme nous avons un environnement fonctionnel, cet état courant représente l'évolution des activités collectives dans le temps. Les modalités opératoires définies pour l'organisation deviennent alors les règles constituant le contrôle d'applicabilité. En effet, ces modalités opératoires indiquent ce que chaque composant doit faire par rapport à l'objectif. Ainsi, si une décision émise par le comportement du composant est jugée non favorable à l'objectif collectif, elle sera rejetée. Dans le cas contraire, elle sera validée et conduira à la modification de l'environnement fonctionnel, donc conduira à l'atteinte de l'objectif. Nous retrouvons de cette manière le mécanisme du contrôle d'applicabilité dans sa capacité de filtrage des décisions de comportements qui sont non opportunes.

Dans notre exemple, les modalités opératoires pour l'activité collective d'optimisation de chargement sont :

1. Pour la borne : rechargement à 80% quand le nombre de voitures atteint un certain seuil ;
2. Pour la voiture : ne pas se recharger quand la file d'attente est importante, et d'être économe quand le chargement est limité à 80 % ;

Ces modalités opératoires ne sont pas toujours en cohérence avec leurs propres objectifs qui sont : faire le plein pour la voiture, et recharger les véhicules à 100% pour la borne. Ces objectifs propres peuvent être réalisés lorsque que le nombre de véhicules en attente ne dépasse pas le seuil. Dans le cas contraire, les demandes d'actions associées à ces objectifs seront rejetées.

5.3.2 Interface d'interaction

La deuxième exigence que nous avons identifiée auparavant consiste à fournir un moyen d'interaction pour les composants. Le but est de leur permettre le partage d'informations. Il nous faut donc une interface comme support à cette interaction. Cette interface doit permette d'un côté l'envoi de l'information, et de l'autre côté l'accès à cette dernière. En d'autres termes, il faut une interface pour l'écriture et la lecture. Notre modèle générique GMAS répond à ce besoin. GMAS est constitué de deux interfaces 3.4 qui sont : la fonction de monitoring pour la lecture, et la fonction d'exécution pour l'écriture. Par rapport à notre environnement fonctionnel, ces interfaces ne seront donc pas des interfaces physiques comme celles du niveau local. Ici nos interfaces seront aussi fonctionnelles. Ici le monitoring représente les ressources d'informations sur l'environnement fonctionnel. Cela peut être par exemple, la boîte de réception des notifications, ou la fonction d'écoute des événements. La fonction d'exécution représente les transactions qui modifient l'état de l'environnement. Cela peut être par exemple les fonctions de mise à jour. Ces fonctions d'interaction requièrent

parfois des supports physiques ou logiciels : une application mobile, ou bien une plateforme web.

Dans notre exemple de chargement de voiture, nous avons deux fonctions pour le monitoring. La première est du côté du véhicule, il s'agit de l'observation du planning de la borne. La deuxième est du côté de cette dernière, il s'agit de la fonction d'écoute d'arrivée et de départ des véhicules. Nous avons également deux fonctions pour l'exécution. La première concerne l'enregistrement sur la file d'attente du côté du véhicule. La deuxième représente la mise à jour du nombre de voitures en attente du côté de la borne. Pour le véhicule, le support de l'interaction peut être une application mobile. Pour la borne, il peut s'agir d'une plateforme web accessible depuis l'application du véhicule.

5.3.3 Distribution du contrôle au niveau local

Un des aspects à gérer dans une activité collective est la distribution du contrôle. En simulation, il est possible de créer un agent destiné à contrôler le bon déroulement de l'activité collective. Son rôle est de faire le suivi de l'état de l'activité. Ensuite, il peut demander un ajustement des actions d'un des composants quand cette action ne va pas dans le sens de l'objectif. Dans un environnement ambiant, il n'est pas toujours possible d'ajouter un élément externe pour piloter l'ensemble. Même dans le cas où c'est possible, centraliser le contrôle limite la robustesse du système. Le contrôle peut être interrompu à tout moment par ces événements imprévisibles. De ce fait, dans notre cadre d'application, tout le contrôle doit être distribué dans les composants du collectif. Chaque composant est donc amené à considérer cette dimension collective.

Cette distribution du contrôle est fondamentale dans notre modèle GMAS. En effet, nous avons vu qu'elle repose sur une approche orientée comportements 2.3. Dans une telle approche, un comportement est défini comme une entité plus ou moins indépendante sur l'ensemble duquel est réparti le modèle décisionnel [HG07]. Ainsi dans notre modèle, les comportements de chacun des composants participants incluront cette dimension collective. Les comportements à vocations collectives peuvent être obtenus de deux manières. Soit, ils sont définis à part puis ajoutés à la liste des comportements du composant. Soit, on modifie les comportements existants pour supporter les contraintes sociales. Dans tous les cas, la gestion de ces comportements à vocation collective est propre au niveau local qui les présente.

Avec notre exemple, certains comportements ont été ajoutés une fois passé au niveau collectif. C'est le cas par exemple des comportements menant à la requête d'informations sur le chargement. Certains comportements ne sont pas ajoutés mais se basent sur l'existant. C'est le cas du comportement de chargement à plein du côté de la borne. Ce sont les modalités opératoires qui vont faire que ce comportement sera modifié en fonction des contraintes

sociales.

Au terme de cette section, nous avons maintenant toutes les spécifications attendues de notre architecture compte tenu des exigences de notre application. Avec notre modèle GMAS, nous parvenons à dégager les éléments de notre architecture qui répondent à ces exigences. Nous proposons les règles et les modalités opératoires pour la gestion des ressources limitées. Aux besoins d'interaction, nous proposons les interfaces de monitoring et d'exécution correspondant à la lecture et à l'écriture. Notre approche orientée comportement résout le problème de distribution du contrôle et de la décision. Nous retrouvons ainsi l'ensemble des 4 unités constitutives de GMAS, spécifiées pour l'organisation globale. Formalisons maintenant notre modèle d'architecture pour le niveau global dans la section qui suit.

5.4 Formalisation de l'architecture

Dans cette section, notre objectif est de formaliser notre architecture avec les spécifications propres au niveau global. Nous l'illustrons ensuite avec l'exemple de chargement de voiture.

5.4.1 Architecture pour l'adaptabilité collective

Nous avons montré que notre modèle générique répond pleinement aux besoins d'une activité collective sans avoir recours à d'autres nouveaux éléments. Nous retrouvons donc la même structure de GMAS mais avec des spécifications orientées activités collectives cette fois-ci. La figure 5.1 résume cette nouvelle instance de GMAS. Nous retrouvons le même pattern de GMAS dans la figure 3.5 précédent et celui de l'instance de GMAS au niveau individuel dans la figure 4.5. Nous décrivons chacun des éléments dans la suite de cette section.

Environnement fonctionnel

Comme nous avons vu précédemment, le contexte d'exécution des activités collectives est un environnement fonctionnel. Cet environnement est caractérisé par son état dynamique, fonction du temps. Nous revenons sur une analyse plus détaillée de cet environnement fonctionnel dans la sous-section formalisation 5.4.2 suivante.

Unité de supervision

L'unité de supervision est l'instance du *Monitoring* de GMAS. Elle donne les informations sur l'évolution de l'activité collective. C'est une fiche directive qui regroupe les notifica-

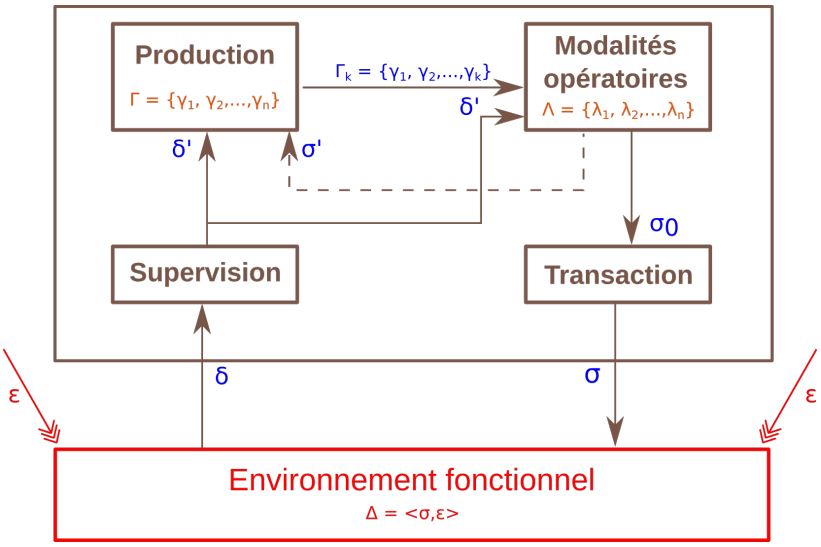


FIGURE 5.1 Instance de GMAS correspondant au niveau collectif

tions provenant des composants : notification de début et de fin d'activité, notification de changement d'état de l'environnement fonctionnel.

Unité de production

L'unité de production correspond aux *Décisions* dans GMAS. C'est l'ensemble des comportements des différents agents qui sont mobilisés pour une activité collective.

Modalités opératoires

Les modalités opératoires assurent le *Contrôle d'Applicabilité* qui est l'unité clé de GMAS. Ce sont l'ensemble des règles et conditions pour le bon déroulement des activités collectives. Ces modalités sont des filtres qui laissent passer ou non une sollicitation d'action. Comme nous avons des agents réactifs, ce filtrage ne nécessite pas de raisonnement ou de négociation. C'est une fonction à valeur binaire : soit elle rejette, soit elle valide. Ces modalités vérifient la cohérence des comportements avec l'état courant de l'environnement. Elles ne donnent pas de recommandation d'action sur ce qui est à faire en cas de rejet.

Unité de transaction

L'unité de transaction est l'instance de l'*Execution*. Cette unité regroupe les actions collectives qui font passer l'environnement fonctionnel d'un état à un autre.

Objectif	Optimisation du temps de chargement des véhicules
Composants	Une borne et des véhicules
Modalités opératoires	Pour un important nombre de véhicules : Borne : Limiter la quantité d'énergie à 80% Véhicule : Choisir une autre plage horaire de chargement

TABLE 5.1 Fiche directive de l'activité collective

5.4.2 Formalisation de l'exemple de chargement

Pour mieux expliquer chacun des éléments de cette architecture revenons sur l'exemple du chargement de voiture. Considérons également les différentes fonctions définies dans GMAS. Le tableau 5.1 nous décrit l'activité collective entre la borne et les véhicules.

Environnement fonctionnel

La figure 5.1 reprend les fonctions caractérisant chacune des briques de GMAS. L'environnement cible est représenté par son état dynamique : $\Delta = <\Sigma, \varepsilon>$. Pour le chargement de véhicule, cet état dynamique de l'environnement est représenté par le nombre de véhicules en attente qui évolue dans le temps. Cela nous donne deux variables de la fonction associée à cet état : le temps t, et le nombre de véhicules n à l'instant t. Nous obtenons : $\forall\ \delta \in \Delta$, δ(t,n) ou encore δ_t(n) tel que n $\in \mathbb{N}$.

La valeur de cet état dynamique évolue avec l'arrivée de nouveaux véhicules pour le chargement et le départ des véhicules une fois chargés. Nous associons deux fonctions $In_t \longmapsto \mathbb{N}$ et $Out_t \longmapsto \mathbb{N}$ qui correspondent respectivement au nombre de nouveaux véhicules qui arrivent à l'instant t, et le nombre de véhicules qui partent à l'instant t. Ces deux fonctions In_t et Out_t sont réalisées directement par les participants à l'activité collective. Un nouveau véhicule qui vient d'arriver s'enregistre sur la borne et met à jour la valeur de la fonction In_t, tandis qu'une fois le chargement fini, la borne met à jour la valeur de la fonction Out_t. Dans certain cas, il se peut qu'un véhicule sur la file d'attente parte avant d'être chargé. Dans ce contexte, le véhicule modifie la valeur de la fonction Out_t. Ces deux fonctions modifient directement l'état dynamique de l'environnement fonctionnel. Elles sont issues de l'organisation interne des composants. Ces deux caractéristiques font que ces deux fonctions constituent l'ensemble Σ. Nous obtenons : $\forall\ \sigma \in \Sigma$, $\sigma_t = In_t - Out_t$

Si l'ensemble Σ résulte de la dynamique interne du système, l'ensemble ε est quand à lui issu d'une dynamique externe au système. Ici, ε représente les événements extérieurs imprévisibles. Il peut s'agir par exemple de la panne de la borne, ou bien d'un problème d'accès au planning de la borne. Ce paramètre va donc modifier la valeur de l'état dynamique. Par exemple, en cas de panne de la borne, la fonction ε va bloquer la mise à jour de la fonction σ_t.

En résumé, l'état dynamique de l'environnement fonctionnel relatif à l'activité collective d'optimisation est décrit comme suit : $\forall\ \delta \in \Delta$:

$$\delta_t = \delta_{t-1} + \sigma_t + \varepsilon_t$$
$$\delta_t = \delta_{t-1} + (In_t - Out_t) + \varepsilon_t$$

Les unités de l'architecture

Les **unités de supervision** sont formées par les interfaces d'écoute de l'environnement.

— Pour la borne c'est le planning. Il fournit l'information sur le nombre courant de véhicule.

— Pour le véhicule c'est l'application mobile. Elle reçoit le nombre courant de véhicule après avoir envoyé une requête auprès de la borne.

Les unités de supervision de GMAS ont comme rôle de collecter l'information δ puis de les formater en δ' pour que l'information soit reconnue par les autres unités. Dans notre exemple, nous ne faisons pas face à un problème de formatage. En effet, la valeur de l'état dynamique de l'environnement perçue par l'unité de surpervision est directement exploitable par les autres unités. Par conséquent, nous avons donc : $\delta'_t = \delta_t$.

Les **unités de production** sont l'ensemble de comportements Γ des composants. Ces comportements concernent généralement l'ajout et la suppression d'un véhicule dans la file. Chacun des comportements est ensuite géré auprès de chaque composant. Nous avons dans GMAS la fonction Γ qui devient :

$$\Gamma = \{\gamma_{B1}, \gamma_{B2}, \gamma_{V1}, \gamma_{V2}\}$$

avec :

— γ_{B1} : Faire le plein aux véhicules

— γ_{B2} : Mettre à jour, après le chargement, le nombre de véhicules en attente (suppression)

— γ_{V1} : Enregistrement sur le planning de la borne (ajout)

— γ_{V2} : Se retirer de la liste d'attente (suppression)

Les **modalités opératoires** vérifient l'applicabilité de ces comportements une fois qu'ils demandent leur exécution. L'ensemble des règles Λ de GMAS devient :

$$\Lambda = \lambda_B, \lambda_V$$

Avec

— λ_B : Rejeter le chargement à 100% (γ_{B1}) si $\sigma' > k$, $k \in \mathbb{N}$ représente le nombre maximal de voiture en attente que l'on peut charger à 100%.

— λ_V : Rejeter l'enregistrement sur le planning (γ_{V1}) si $\sigma^{'}$ > x, x étant le nombre maximal de voitures en attente supporté par la borne, x > k.

Lorsque les conditions de rejet sont vérifiées, deux réactions internes sont renvoyées vers les composants qui ont émis la demande du comportement. $\sigma^{'}_{B1}$ correspond au rejet du comportement γ_{B1}, tandis que $\sigma^{'}_{V1}$ sera renvoyé vers le véhicule qui a émis γ_{V1}. Avec cetre réaction interne, chacun des composants va ajuster ses comportements. Pour la borne, la réception de la notification $\sigma^{'}_{B1}$ conduira au chargement à 80%, tandis que pour le véhicule, la notification de rejet $\sigma^{'}_{V1}$ l'amène à choisir une autre plage horaire.

Lorsqu'un comportement γ_i est validé, une demande de l'exécution de l'action correspondante σ_{Oi} sera envoyée vers l'unité de transaction.

Les **unités de transaction** mettent à jour l'état dynamique de l'environnement suite à une demande d'action. Une notification de type σ_{Oi} conduit à la réalisation de σ_i. σ_i est l'action associée à γ_i.

Par exemple, supposons que le comportement γ_{B2} de mise à jour du nombre de véhicules en attente après le chargement (suppression) soit validé. Une notification de type σ_{OB2} est envoyée. Cela conduit à la réalisation de σ_{B2} qui correspond à la fonction Out_t puisque la fonction supprime une voiture de la file. La valeur de l'état dynamique δ_t va donc être modifiée en conséquence.

5.4.3 Synthèse

Nous spécifions à travers cette section chacun des éléments de GMAS correspondant à l'activité collective. Nous illustrons l'architecture résultante avec l'exemple du chargement de véhicule électrique. Nous arrivons aux conclusions suivantes :

— Chaque élément de GMAS possède une instance bien définie dans la nouvelle architecture globale ;

— Chaque unité de l'architecture globale est distribuée parmi les composants, il en est de même pour le contrôle représenté par les modalités opératoires ;

— L'architecture ne fait appel à aucun élément additionnel, l'instance de GMAS assure toutes les fonctions requises.

Ces conclusions nous montrent l'importance du niveau local. Toutes les fonctions sont réalisées à ce niveau. Nous reviendrons sur cette cohésion entre le niveau global et le niveau local dans l'analyse qualitative de la section 5.5 suivante.

5.5 Analyse qualitative

Nous abordons dans cette section les différents points clés de notre architecture. Nous analysons particulièrement le suivi et l'ajustement dynamique des comportements qui conduisent à l'adaptation. Ensuite, nous décrivons comment le modèle de flexibilité au niveau local et le modèle d'adaptation au niveau global sont compatibles et complémentaires.

5.5.1 Adaptation par suivi et ajustement dynamique

L'objectif principal de cette thèse est de fournir un modèle conceptuel d'architecture pour l'adaptation en environnement ambiant. Nous parvenons ici à cet objectif avec le filtrage des décision de comportements avec les modalités opératoires. Cette adaptation résulte de l'organisation et de l'interaction entre des agents réactifs simples. La structure de notre architecture fait qu'elle fournit de manière implicite deux fonctions qui conduisent à l'adaptation. Ce sont la fonction de suivi, et la fonction d'ajustement.

La fonction de suivi

La vigilance au contexte (*context-awareness*) est une des propriétés requises pour s'adapter en environnement ambiant. Cette propriété est définie comme la capacité du système à détecter les changements de son environnement opérationnel [KRV+15] [ST09]. La vigilance au contexte est présente dans notre modèle. Avec notre unité de supervision, il est possible d'observer en permanence l'évolution de l'environnement.

Toutefois, plusieurs composants peuvent participer à une activité collective. Ces composants sont réactifs. Ils n'ont pas une vue globale ou une représentation de leur environnement alors qu'ils sont censés surveiller ce dernier. Nous résolvons ce problème de visibilité avec l'interaction. L'interaction est le support des échanges d'information. Chaque composant partage et rend accessible les informations sur son état, par envoi de message ou par notification. C'est uniquement dans ces conditions, que les unités de supervision ont accès aux informations qui leur sont pertinentes.

Pour avoir cette propriété de vigilance au contexte nous avons donc deux conditions complémentaires. Nous avons les unités de supervision pour l'écoute d'un côté, et l'interaction pour le partage d'informations de l'autre côté. Les unités de supervision ne peuvent pas réaliser leur mission tant qu'elles n'ont pas accès aux informations. Par exemple, le planning de la borne ne peut pas donner le nombre de véhicules entrant tant que le véhicule ne s'est pas enregistré sur ce planning. Ces deux conditions sont présentes dans notre modèle. Elles conduisent à l'émergence d'une nouvelle fonction qui est la fonction de suivi. Elle est émergente dans la mesure où elle est produite par une interaction entre les composants. Si les composants opéraient en parallèle de manière isolée sans interagir avec les autres compo-

sants, cette fonction ne peut pas se réaliser. Cela nous montre encore une fois l'importance de l'interaction.

La fonction d'ajustement dynamique

La fonction d'ajustement s'appuie sur la fonction de suivi. C'est la dernière étape conduisant à l'adaptation. Cette fonction a pour but d'activer les actions opportunes par rapport aux valeurs de la fonction de suivi. Cette fonction vise à ce que le système évolue en cohésion avec l'objectif collectif.

La fonction d'ajustement est un mécanisme de contrôle par rétroaction. Un tel mécanisme est présent dans plusieurs systèmes comme les Systèmes Auto-Adaptifs [BSG+09], ou dans les systèmes autonomiques fournis par IBM's *autonomic computing* [KC03] [DDF+06]. Cependant, il existe une différence majeure entre ce même mécanisme dans ces modèles et dans notre modèle. Dans ces systèmes, les informations sont soumises à des analyses et planifications pour identifier la future action à entreprendre. Le composant qui les réalise est ainsi doté d'une capacité cognitive. Ils sont proactifs.

Notre cadre applicatif est constitué de composants sans grande puissance de calcul. Il est donc impossible de leur doter d'un algorithme d'analyse et de capacité de raisonnement importants comme dans ces systèmes cognitifs. Nos composants sont des agents réactifs simples. Ils réagissent selon leur perception. Cette perception n'est pas forcément physique mais peut très bien être fonctionnelle comme c'est le cas ici. Leur mécanisme d'ajustement se résume par le choix du comportement futur en fonction de l'activation ou le rejet du comportement précédent. L'ajustement est alors dynamique puisqu'il est fonction de l'état courant de l'environnement.

Ainsi, nous pouvons conclure que dans notre modèle il est possible de réaliser un mécanisme de contrôle par rétroaction en utilisant des agents réactifs.

5.5.2 Uniformité et cohésion avec le niveau local

Comparaison avec la relation micro-macro classique

Nous analysons dans cette partie la relation entre le modèle pour le niveau local et le modèle pour le niveau global. Pour cela, partons de la relation micro-macro dans la littérature SMA. Cette relation a été longtemps définie de manière explicite. Dans [Jac95], "Toute organisation est le résultat d'une interaction entre agents, et le comportement des agents est contraint par l'ensemble des structures organisatrices." Cette relation est représentée par la figure 5.2.

Toutefois, même si cette relation est explicite dans l'ensemble des SMA, elle n'apparait pas pour autant sur toutes les problématiques liées au SMA. Par exemple, nous avons vu dans 2.4 qu'en terme d'adaptation les méthodes pour le micro et celles pour le macro sont

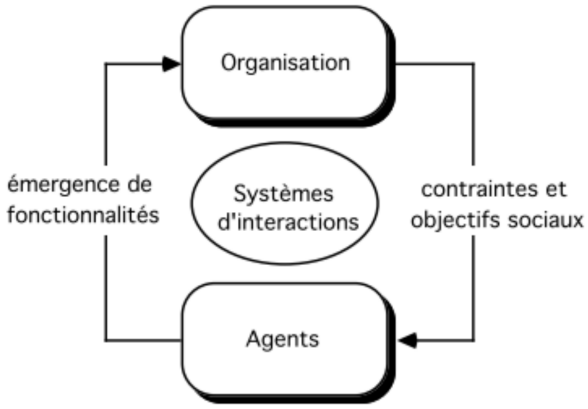


FIGURE 5.2 La relation micro-macro dans les système multi-agents [Jac95]

généralement différentes. Par conséquent, les modèles d'architecture qui supportent cette adaptation sont également différents. L'intégration des deux modèles une fois rassemblés n'est pas toujours garantie. Avec notre modèle uniforme, nous obtenons les caractéristiques suivantes, illustrées par la figure 5.3 qui suit.

— Les deux niveaux local et global utilisent le même modèle d'architecture

— Ces deux modèles sont compatibles entre eux

— Ces deux modèles sont conformes à la relation micro-macro des SMA illustrée par la figure 5.2

Nous décrivons en détail cette cohésion agent et organisation dans la partie suivante.

Analyse de la cohésion entre niveau local et global

La figure 5.3 reprend la même représentation de la relation micro-macro de la figure 5.2. Nous retrouvons alors les mêmes blocs : de haut en bas, l'organisation, le système d'interaction, et l'agent. Dans notre modèle l'organisation et l'interaction constituent le niveau global. Par souci de visibilité, considérons uniquement un agent, mais la relation est valable pour tous les agents.

L'agent (en marron) est défini avec ses 4 unités que nous avons vu dans 4.5 : *Capteurs, Comportements, Régulateurs, et Actuateurs*. Pour participer à une activité collective l'agent est doté de nouvelles sous-unités à caractère social pour chacune des unités. Les carrés en jaune et en bleu dans l'agent les représentent. Tandis que les carrés en marron représentent les sous-unités à caractère individuel. Les sous-unités collectives des unités de *Comportement*, *Capteur*, et *Actuateur* sont ensuite mises à disposition dans le collectif, respectivement dans les unités de Production, Supervision et Transition. Ces deux dernières unités représentent le système d'interaction, pour l'écoute et la mise à jour. La sous-unité collective du régulateur est par contre différente des trois autres. En effet, cette sous-unité (en bleu dans le régulateur

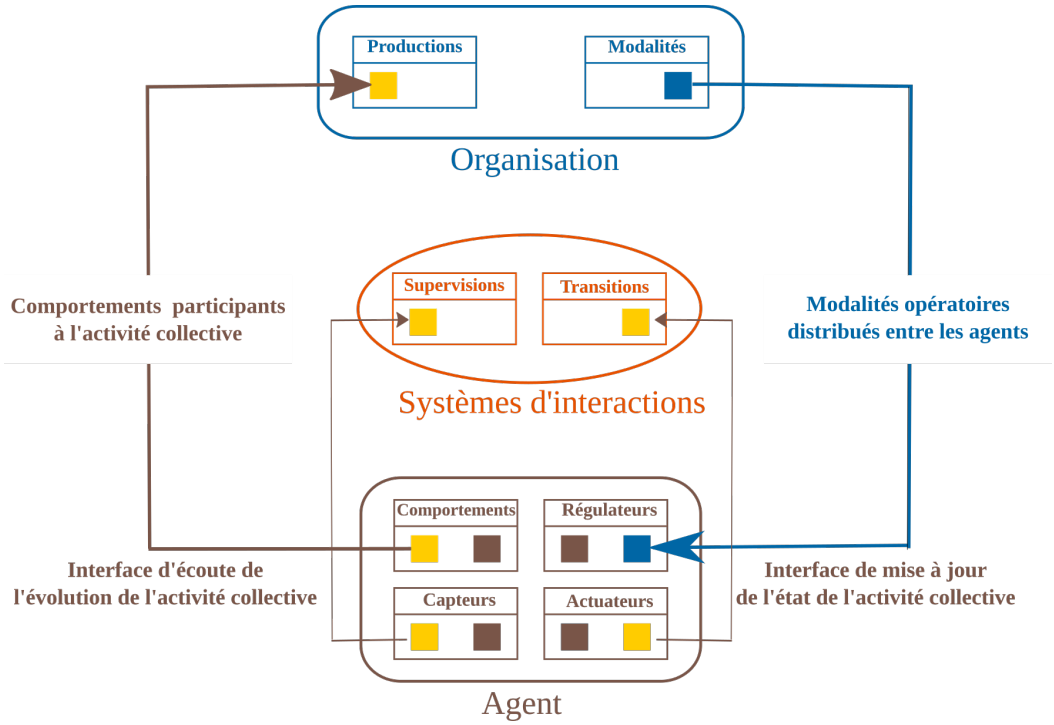


FIGURE 5.3 La relation agent et organisation avec GMAS

de l'agent) n'est pas issue de l'agent. C'est une contrainte issue du collectif qui s'impose à l'agent. C'est la modalité opératoire qui une fois distribuée dans les agents rejoint le régulateur. Régulateur et modalités opératoires sont compatibles puisque ce sont tous les deux des règles.

Une telle configuration montre le caractère fortement distribué de notre architecture. En effet, la réalisation concrète de chaque action revient à l'agent. Toute sous-unité d'organisation et d'interaction appartient obligatoirement à un agent distinct. Aucune sous-unité du niveau global n'est isolée d'un agent. Toutefois, avec ce modèle, il peut y avoir potentiellement des conflits entre les sous-unités de nature différente. Par exemple, un comportement purement individuel peut être en conflit avec un comportement à vocation collective. Pour éviter une telle situation, nous définissons arbitrairement une priorité entre les sous-unités. Les exigences locales propres à l'agent doivent être satisfaites en priorité. C'est seulement après que l'agent peut prendre en compte ses obligations collectives. Cette priorisation est une choix stratégique que nous proposons. Elle peut être modifiée en fonction de l'application souhaitée.

5.5.3 Limites

Nous avons montré à travers ce chapitre l'importance de l'interaction dans une activité collective. La moitié de notre modèle d'architecture elle-même est formée par les interfaces

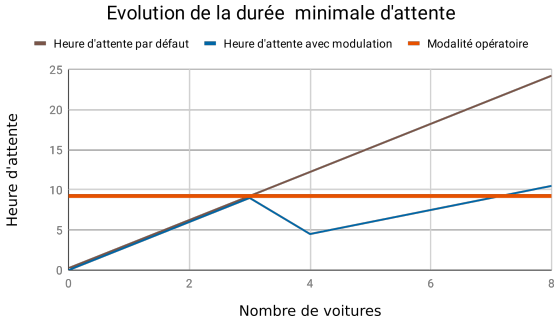


FIGURE 5.4 Evolution du chargement sans limitation de nombre de voitures

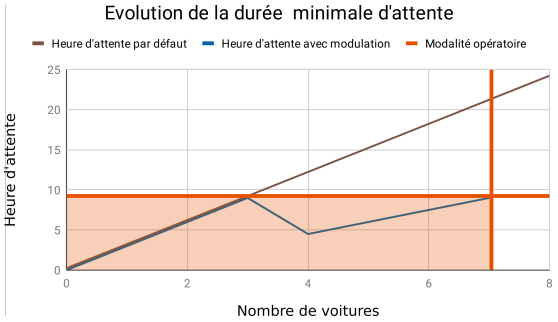


FIGURE 5.5 Evolution du chargement avec limitation de nombre de voitures

d'écoute et de mise à jour de l'état de cette activité. Nous avons également montré que les fonctions de suivi et donc d'ajustement sont des fonctions émergentes qui n'ont pas lieu si aucune interaction n'existe.

Pour la plupart des composants en environnement ambiant, il est facile d'avoir accès à un portail web ou une application mobile pour interagir. Néanmoins, nous avons identifié des cas où l'interaction est difficile. Lorsque les technologies de communication des composants sont fortement diversifiées, ils ne peuvent pas communiquer directement.

Pour illustrer ce problème, considérons un exemple en domotique. Il existe deux technologies différentes d'ampoule connectée : ampoule bluetooth, et ampoule wifi. Le partage d'informations entre ces deux ampoules ne peut pas se faire directement. Il est donc impossible d'utiliser notre modèle pour coordonner une activité collective entre ces deux ampoules. Un exemple d'activité collective possible est l'optimisation de la consommation d'énergie. Cela ne peut être possible que si nous ajoutons une interface supplémentaire entre les ampoules. Un smartphone peut par exemple s'appairer avec l'ampoule bluetooth. Ce même smartphone peut être connecté à l'autre ampoule via un pont wifi. C'est une fois que cet interfaçage avec le smartphone est établi qu'une activité collective peut avoir lieu.

Pour pallier à ce problème, nous nous appuyons sur une plateforme développée dans notre équipe de recherche. C'est un espace d'interaction sociale. Dans cet espace un ensemble de machines connectées en réseau peut interagir. Le but est de faire en sorte que les problèmes liés aux infrastructures du réseau physique n'affectent pas directement les moyens d'interaction et la visibilité des agents. Nous reviendrons sur cette plateforme dans le chapitre 6 qui décrit son implémentation.

5.6 Analyse quantitative

5.6.1 Analyse de l'évolution du temps de chargement

Analysons dans cette partie la gestion de ressource dans notre modèle. Considérons pour cela les deux figures 5.4 et 5.5. Ces figures montrent l'évolution du temps d'attente minimal pour une voiture qui souhaite s'enregistrer sur la borne. On suppose que le temps de passage d'un chargement d'une voiture à une autre est négligeable. Les axes verticaux représentent le nombre d'heures d'attente tandis que les axes horizontaux représentent le nombre de voitures qui sont déjà enregistrées sur la borne. Nous supposons que le chargement de la voiture en moyenne dure 3h pour un chargement à 100%. Ce qui revient à environ 1h30 pour un chargement à 80%. Dans cet exemple, nous choisissons la limitation de la quantité d'énergie rechargée. Toutefois, il est tout à fait possible de choisir la vitesse de chargement comme paramètre de limitation du temps d'attente. En effet, avec les bornes électriques, il existe des modes de chargement : normal, semi-rapide et rapide.

Les droites en marron (heure d'attente par défaut sur les figures) représentent l'évolution du temps d'attente sans prise en compte des contraintes sociales. Aucune modalité opératoire ne s'applique. Nous obtenons alors une droite strictement croissante. Les droites en orange (Modalité opératoire) représentent l'évolution de l'heure d'attente une fois que l'on passe au niveau collectif. Ce qui veut dire, une fois que les contraintes sociales, dont la modalité opératoire, sont appliquées. Dans la figure 5.4 la droite en orange représente la limitation du temps de chargement à 9h maximum. Les droites en bleu représentent alors l'évolution du temps d'attente une fois que l'on réduit à moitié le temps de chargement. Nous observons pour n = 3 que la durée d'attente est aux alentours de 9h. Ce qui fait que la limitation à 80% va être activée. C'est ce qui explique la décroissance de la courbe. La courbe redevient ensuite croissante mais le taux de croissance avec la courbe par défaut (en marron) n'est pas le même. Néanmoins, au bout d'un certain nombre de voitures (ici pour n=7), nous remarquons que la courbe commence à dépasser les 9h. C'est pour éviter cela que nous choisissons le nombre correspondant à l'intersection n=7, comme le seuil au delà duquel, la demande d'enregistrement du véhicule sera rejetée. Nous obtenons alors la droite verticale en orange dans la figure 5.5 qui limite le nombre maximal de voiture en attente à 7. Les deux droites de modalité opératoires définissent un espace (en orange sur le figure 5.5) dans lequel évolue l'état de l'environnement fonctionnel. L'objectif collectif correspond au maintien du nombre d'heure et de voiture compris dans cet espace.

Nous en concluons donc qu'en appliquant les modalités opératoires sur notre système, nous parvenons à maintenir une durée minimale d'attente à 9h pour un nombre maximal de voiture égal à 7. Pour cette même durée de 9h, nous sommes passés de 3 voitures à 7 voitures.

5.6.2 Généralisation

Dans ce chapitre, nous modélisons la gestion d'une borne par rapport aux accès concurrents des véhicules. Nous retrouvons ainsi une problématique courante à une organisation

collective. Il s'agit de la gestion d'une ressource limitée. Le formalisme que nous présentons est indépendant de notre exemple. Ce qui fait que si un problème collectif peut être formalisé comme un accès à une ressource limitée, notre modèle devient alors pertinent pour résoudre ce problème.

Nous avons vu dans 5.2.3 que nous avons deux types de gestion de ressources : soit une gestion d'accès concurrent, soit une optimisation de l'exploitation d'une ressource. Pour la première gestion de ressource, l'objectif est d'avoir une répartition équitable de la ressource. Supposons que le nombre de composants qui sollicite une ressource soit égale à n. La disponibilité de la ressource est inversement proportionnelle à n. Ce qui correspond à 1/n. Cela signifie que pour une gestion équitable, chacun des composants doit avoir accès à la ressource avec environ le même coût. Avec le cas de la borne, si nous supposons que la durée totale du chargement des n voitures soit égale à T. Le coût d'accès à la borne est le temps d'attente. Le but de la limitation de chargement est d'éviter un important écart entre les coûts. Ce qui revient à les stabiliser aux alentours de T/n.

Dans le premier cas, il est question de répartition équitable de l'accès à la ressource entre les composants. Tandis que dans le deuxième cas, le but est de minimiser le temps d'attente. Pour le deuxième type de gestion de ressource, l'objectif est de limiter au maximum son utilisation. C'est le cas par exemple du temps d'attente pour le véhicule. La quantité d'énergie est modifiée pour minimiser le temps de chargement.

Dans ces deux cas, nous sommes amenés à maintenir certains paramètres. Soit c'est le rapport équitable de la disponibilité d'une ressource, soit c'est la quantité de ressource minimale à exploiter. Ce maintien requiert des organisations entre les participants à l'activité. Ce sont les règles relatives à cette organisation qui constituent les modalités opératoires caractéristiques de notre modèle. En d'autres termes, les modalités opératoires constituent les fonctions visant à maintenir une gestion optimale des ressources limitées.

5.7 Conclusion

Nous montrons à travers ce chapitre comment l'instance de GMAS résout les problèmes liés au niveau global. Nous partons d'un exemple de chargement de voiture électrique pour montrer l'importance de l'organisation et de l'interaction. Notre modèle s'appuie sur ces deux forces du collectif. Sa particularité repose sur un mécanisme de vérification des décisions de comportements par rapport aux modalités opératoires. Ces modalités opératoires correspondent au contrôle d'applicabilité de GMAS. Ici son rôle est de vérifier si un comportement va ou non dans le sens de l'objectif collectif.

L'architecture globale est caractérisée par les points clé suivants. Chaque composant de l'architecture est distribué entre les agents participants (i). Les éléments d'interaction et

ceux de production sont issus des agents. Tandis que les modalités opératoires leur sont imposées par rapport à leur objectif collectif. Cette architecture s'appuie ainsi sur le niveau local puisque c'est à ce niveau que toutes les actions mêmes sociales s'exécutent (ii). Cette architecture répond aux besoins d'une optimisation d'une activité collective. Elle vise à ce que l'évolution de l'activité soit en accord avec l'objectif collectif (iii).

Cette architecture montre son efficacité dans la résolution de problème de gestion de ressource limitée. Ce qui constitue un problème assez courant dans un collectif. Notre modèle offre une solution complète aux exigences de ce problème sans faire appel à d'autres nouveaux éléments supplémentaires. Néanmoins, le fort besoin d'interaction peut limiter l'efficacité de notre modèle pour les cas où les moyens de communication sont limités.

Nous possédons maintenant le modèle complet de notre proposition incluant le niveau local et le niveau global. Nous nous intéressons maintenant à la partie technique du modèle. Nous décrivons dans le chapitre 6 suivant l'implémentation de ces deux modèles pour le niveau local et le niveau global.

Chapitre 6

Implémentation

Nous abordons les parties techniques de la thèse dans ce chapitre. Nous présentons ici l'implémentation des modèles conceptuels d'architectures décrits dans les chapitres précédents. Notre objectif est de présenter les méthodes d'implémentation, et les différentes plateformes que nous utilisons pour concrétiser notre contribution dans des applications.

Ce chapitre commence par l'expérimentation de notre architecture pour le niveau global avec la librairie MECA dans la section 6.1. Ensuite, nous aborderons le passage du niveau local au niveau global avec notre plateforme d'interaction sociale *Ubiquity* dans la section 6.2. Pour le niveau global, nous commençons dans la section 6.3 par une description des fonctionnalités requises pour chaque brique du modèle. Ensuite, nous présentons deux applications pour notre architecture. La première, dans les sections 6.4 et 6.5, est un cas d'école. C'est l'optimisation de la gestion d'une ressource limitée avec le modèle proie-prédateur. La section 6.4 s'intéresse particulièrement à la question comment définir les différents éléments du code, les classes par rapport à l'objectif collectif. Nous abordons davantage dans la section 6.5 la structuration du code par rapport aux fonctionnalités attendues. La deuxième application est décrite dans la section 6.6. C'est l'exemple du chargement des véhicules électriques vu dans le chapitre 5 précédent. Nous revenons ensuite sur les points clés de ces implémentations dans la conclusion dans la section 6.7.

6.1 Le niveau local

Dans cette section, nous présentons un exemple d'application de notre architecture pour le niveau local. Il s'agit de l'implémentation de notre architecture MECA sous forme de Librairie Java. La librairie résultante est ensuite utilisée pour le pilotage d'un robot agricole appelé Tropic'Oz.

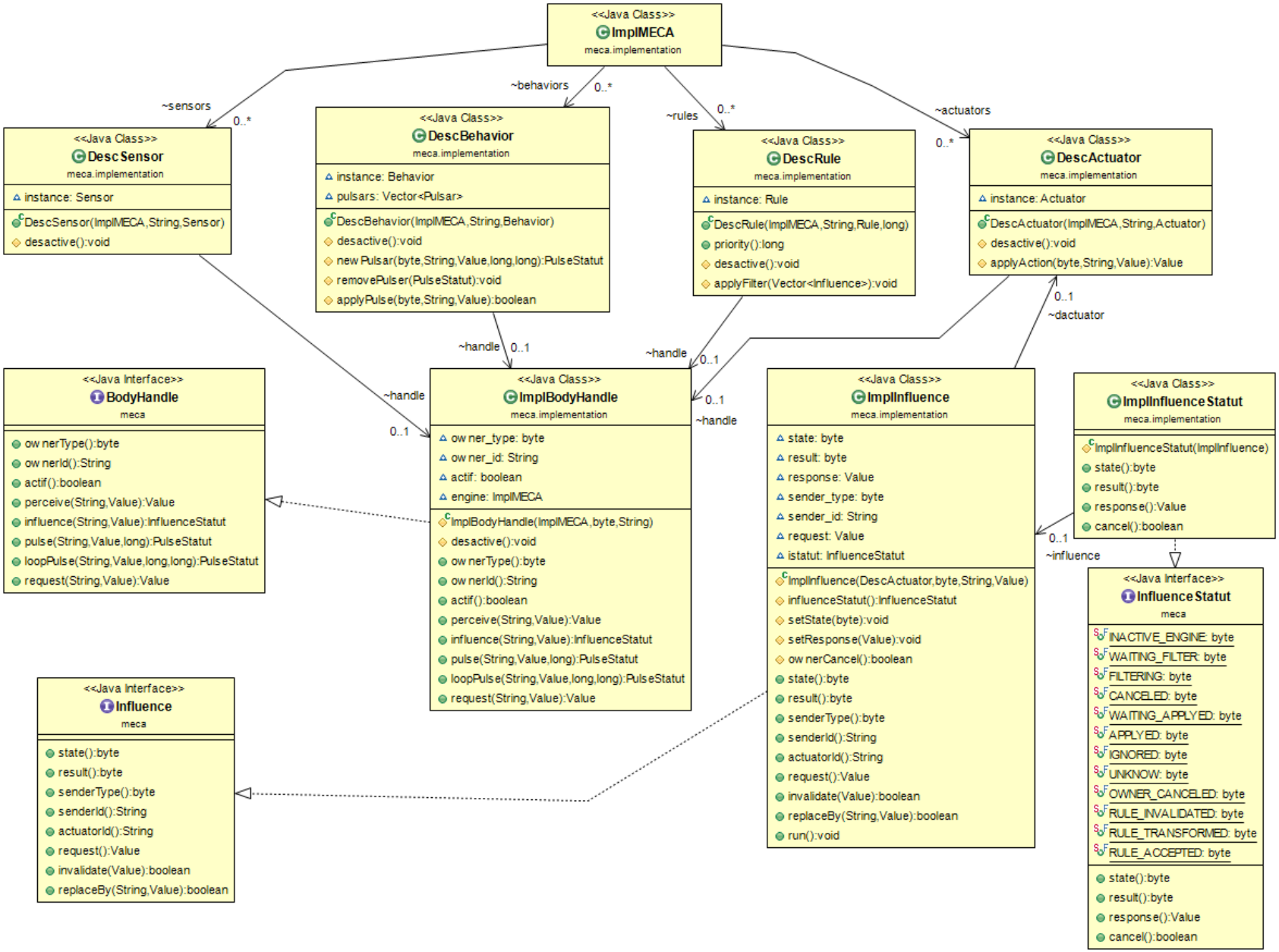


FIGURE 6.1 Diagramme de classes principales de MECA

6.1.1 La librairie MECA

Nous avons mis au point une librairie Java pour implémenter des agents selon l'architecture MECA [Pay16]. C'est ce qui constitue la *MECA library*. Cette librairie a fait l'objet d'une publication présente dans [VPC19]. Le diagramme de classes sur la figure 6.1 nous montre quelques classes caractéristiques de MECA. La classe principale de l'architecture MECA est la classe ImplMECA (tout en haut sur le diagramme). C'est l'instance de cette classe qui anime et organise le traitement de l'ensemble. Toutes les classes représentant les 4 briques de l'architecture sont associées à cette classe. Les 4 classes *DescSensor*, *DescBehavior*, *DescRule* et *DescActuator* correspondent respectivement au capteur, comportement, régulation, et actuateur. Ces classes se trouvent sur la deuxième ligne en bas de la classe principale ImplMECA. Les trois classes du bas sont des classes spécifiques à la librairie. Nous les décrivons particulièrement dans ce qui suit.

6.1.2 ImplBodyHandle : interaction entre unités de code

ImplBodyHandle est une classe qui implémente une interface appelée *BodyHandle*. Cette interface porte l'information pour la description de son propriétaire : identifiant, indication de l'état actif ou inactif. Ce descripteur est propre pour chaque unité de code. Ce dernier le reçoit au moment de son enregistrement. Comme nous pouvons le voir sur le diagramme, ces unités de code ne communiquent pas directement entre eux. C'est *BodyHandle* qui gère leur interaction. Par exemple, supposons qu'une unité de code de *DescBehavior* demande la vérification de l'applicabilité d'un comportement. Pour cela, il utilise son descripteur *ImplBodyHandle* pour envoyer cette demande avec l'identité de l'unité de code cible.

6.1.3 ImplInfluence : suivi des influences

La librairie MECA offre à l'unité de code de comportement une interface *Influence*, qui est implémentée par la classe ImplInfluence. Cette classe se charge de l'envoi de l'influence avec l'identifiant de l'actuateur ciblé. Le statut du traitement de l'influence est décrit par l'interface *InfluenceStatut*, implémentée par la classe *ImplInfluenceStatut* L'interface *InfluenceStatut* fournit à tout moment l'état du traitement de l'influence : validé ou rejeté. Ces deux classes sont représentées par les deux classes en bas à droite de la figure 6.1.

6.1.4 Expérimentation

Nous utilisons notre architecture MECA dans notre programme de recherche *Agrobotic-OI*. Le projet vise à développer des outils pour aider les agriculteurs dans des tâches difficiles, dangereuses en termes d'effort physique. L'un des axes du projet est l'expérimentation du robot agricole Tropic'Oz pour l'automatisation du désherbage. Pour réaliser le désherbage, le robot circule entre les rangs de culture. Dans ce contexte, une mauvaise direction prise

par le robot présente un danger pour le champ. Le robot risque d'écraser le champ. Ensuite, au cours de son déplacement le robot peut rencontrer des obstacles qu'il ne doit pas non plus écraser. Nous proposons le modèle MECA pour piloter ce robot et éviter les situations néfastes lors du désherbage. L'ensemble de la solution proposée inclut la librairie MECA mais aussi une plateforme de simulation pour faire le test avant le déploiement sur le robot. La plateforme d'essai s'appuie sur l'outil de simulation 3D V-REP [RSF13]. Nous y avons modélisé le robot Tropic'Oz.

Tropic'Oz

Tropic'Oz est une version ajustée du robot de désherbage Oz conçu par Naïo Technologies [1]. Tropic'Oz est équipé de plusieurs capteurs : un lidar, un compteur kilométrique, un gyroscope et un GPS. Il a également 4 roues motrices et un écran LCD comme interface utilisateur. Le système est composé de deux Arduino et d'un Odroid-XU. Les deux Arduino gèrent les capteurs et l'interaction avec l'utilisateur. L'Odroid-XU assure la fonction décisionnelle. Son système d'exploitation est une distribution GNU-Linux XUbuntu 13.10. Avec une telle configuration, un code Java peut être exécuté sur Tropic'Oz.

Avant de lancer MECA sur Tropic'Oz, nous effectuons d'abord une simulation avec une représentation virtuelle du robot. Pour ce faire, nous développons une plateforme de test appelée *Virtual'Oz*. C'est une librairie Java. Nous utilisons ensuite le simulateur V-REP pour concevoir le robot et visualiser la simulation.

Virtual Oz

Virtual Oz est une plateforme expérimentale pour l'architecture MECA. Elle met en œuvre un logiciel agent gérant un robot virtuel de type Tropic'Oz dans un environnement 3D produit par la plate-forme V-REP. Virtual Oz a un double objectif. Le premier consiste à illustrer la faisabilité et le potentiel de MECA. Le second est de concevoir un outil de simulation pour la mise en œuvre du mécanisme de flexibilité au Tropic'Oz réel. La librairie Virtual Oz inclut la librairie MECA et une librairie appelée *Connecteur V-REP*. Ce dernier est une interface entre Virtual Oz et V-REP. *Connecteur V-REP* associe un objet du code à un élément physique de l'interface utilisateur V-REP. Il a un listener pour surveiller les changements sur les capteurs. Il envoie également un signal à *Actuator* lorsqu'il doit exécuter des actions.

V-REP [2] offre un environnement de simulation en 3D. C'est un simulateur open-source. Il présente une architecture de contrôle distribué. Il gère de manière indépendante la partie

1. https://www.naio-technologies.com/en/agricultural-equipment/weeding-robot-oz/
2. http://www.coppeliarobotics.com/

FIGURE 6.2 Tropic'Oz dans le champ

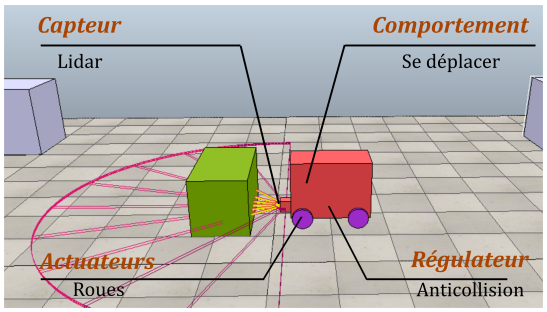


FIGURE 6.3 Tropic'Oz sous V-REP

physique (actionneurs, et capteurs) et la partie pilotage et contrôle. Dans les SMA, cette plateforme a déjà montré son efficacité dans une simulation d'exploration multi-robot [KCS14].

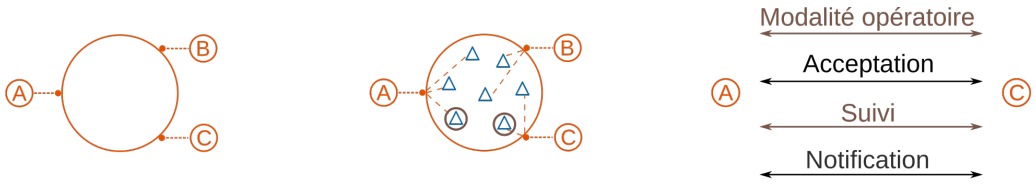
Simulation

Notre robot Tropic'Oz peut disposer de plusieurs comportements, mais dans notre expérimentation, pour simplifier l'illustration des mécanismes de l'architecture MECA, nous nous intéressons au seul déplacement du robot dans les rangs de culture. Nous avons décrit au début de cette section 6.1.4 les dangers potentiels de ce déplacement. Notre but est de maintenir un bon déroulement du désherbage quel que soit les événements extérieurs. Cela nous conduit à la spécification des règles constituant la régulation. Nous devons éviter les collisions dont nous proposons une régulation anticollision. Tropic'Oz est équipé d'un lidar pour la détection des obstacles dans un périmètre de 40 cm. C'est donc le capteur à utiliser. Les éléments mobilisés pour le déplacement sont les moteurs du robot et les roues. Ce sont les actuateurs. Les deux figures 6.2 et 6.3 représentent respectivement le robot Tropic'Oz dans le champ et sa représentation virtuelle sous V-REP.

6.1.5 Résultat

Le scénario test consiste à perturber le mouvement du robot avec différents obstacles, dont le cube en vert sur la figure 6.3. En réaction, les influences émises par le comportement de déplacement sont modulées par le régulateur anticollision. Lorsque le capteur lidar détecte un obstacle dans un rayon de 40 cm, le robot s'arrête. Il peut choisir alors d'aller à gauche ou à droite. Ce comportement de s'arrêter n'est pas définit dans le code de son comportement qui est de se déplacer. C'est le régulateur anti-collision qui a provoqué cet arrêt.

Cette illustration volontairement minimaliste montre qu'à l'aide de l'architecture MECA, nous sommes effectivement capables d'écrire des comportements simplement centrés sur



1. Mise en relation 2. Découverte des comportements 3. Negociation et suivi

FIGURE 6.4 Etapes de constitution du collectif

l'objectif. Ici il ne fait que solliciter l'activation des moteurs tant que la destination n'est pas atteinte. Nous obtenons un comportement flexible par rapport à la situation imprévue, comme la présence d'obstacle. La librairie MECA, sa documentation et les fichiers de cette expérimentation sont disponibles sur le site [Pay16]. Une fois ce modèle validé, l'étape suivante est son déploiement sur Tropic'Oz. L'expérimentation avec ce robot réel a été effectuée par une autre équipe.

6.2 Interaction sociale

Nous présentons dans cette section les solutions d'architecture que nous proposons pour passer du niveau local au niveau collectif. Nous commençons par une description des étapes de ce passage. Nous continuons ensuite avec notre plateforme d'interaction sociale et la mise en commun des comportements.

6.2.1 Constitution du collectif

Nous avons identifié quelques étapes nécessaires pour former un collectif à partir des composants individuels. Ces étapes peuvent être fusionnées en fonction des applications. La figure suivante 6.4 résume la constitution du collective :

La mise en relation des comportements est représentée par la figure de gauche. Elle est fortement dépendante des moyens d'interaction. Dès le début de ce manuscrit nous avons vu qu'une des propriétés majeures de l'environnement ambiant est l'hétérogénéité de ses composants 1.1.1. Une autre spécificité des composants est leur caractère distribué. Ils ne partagent pas forcément les mêmes localisations. Toutes ces contraintes font que la mise en relation de chacun des composants doit être gérée à part quand elle n'est pas automatique. Le cas de la borne et du véhicule électrique nous donne un exemple de mise en relation sans interface supplémentaire grâce à l'accès par le portail web. Dans le cas contraire nous sommes amenés à proposer une plateforme d'interaction.

Une fois que les agents peuvent interagir, l'étape suivante consiste à rendre visibles et accessibles les comportements de chacun des agents. Un agent qui sollicite un comportement d'un autre agent peut alors interagir directement avec l'agent qui présente le comportement. Cette étape correspond à la découverte des services ou comportements disponibles. Elle est représentée par la figure du milieu.

La négociation commence une fois que les agents sont en contact. Les modalités opératoires sont alors définies en fonction des besoins de chacun des agents. Dans l'exemple précédent, toujours avec le rechargement de véhicule, cette étape est gérée directement en amont par le concepteur. Dans le cas contraire, un protocole de négociation est nécessaire pour mettre à disposition et/ou choisir les comportements. Après la négociation, l'activité collective commence avec la mobilisation des comportements. Chacun des participants réalisent ses actions en tenant compte des contraintes sociales. Cette dernière étape est illustrée par la figure de droite.

Après cette description, nous présentons maintenant les outils que nous utilisons pour l'implémentation. Nous nous intéressons particulièrement ici à la mise en commun des comportements. L'implémentation n'inclut pas la phase de négociation. D'abord, parce qu'elle peut être gérée au préalable par le concepteur. Ensuite, parce que pour un souci de simplicité pour notre agent réactif, nous estimons que l'ajout d'un protocole de négociation complexifiera son code. Toutefois, dans les perspectives de la thèse nous pouvons proposer des agents plus complexes pouvant inclure cette fonctionnalité si besoin est.

6.2.2 La plateforme Ubiquity

Description

Dans cette thèse, notre modèle d'architecture repose sur une plateforme d'interaction sociale appelée *Ubiquity*. Cette plateforme met en relation les comportements. Le développement de cette plateforme ne constitue pas l'objet de la thèse. Il a été effectué par d'autres membres de notre équipe de recherche. *Ubiquity* génère un espace d'interaction sociale. C'est un espace où un ensemble de machines connectés en réseau, donc des machines distribuées sur des environnements ambiants différents peuvent interagir. Son objectif est d'éviter que les problèmes liés aux infrastructures du réseau physique, affectent directement les moyens d'interaction et la visibilité des agents. *Ubiquity* joue ainsi le rôle d'interface. Il fait le lien entre les composants physiques et les agents logiciels qui pilotent ces composants. Ces agents se retrouvent ensuite dans un même espace et sont visibles et accessibles entre eux. Avec Ubiquity, il est possible de créer plusieurs espaces d'interaction sociaux distincts. La figure 6.5 donne un aperçu du fonctionnement d'ubiquity.

Nous retrouvons les composants individuels sur la couche inférieure de la figure 6.5. Ils sont distribués dans des environnements physiques différents. Chacun de ces composants

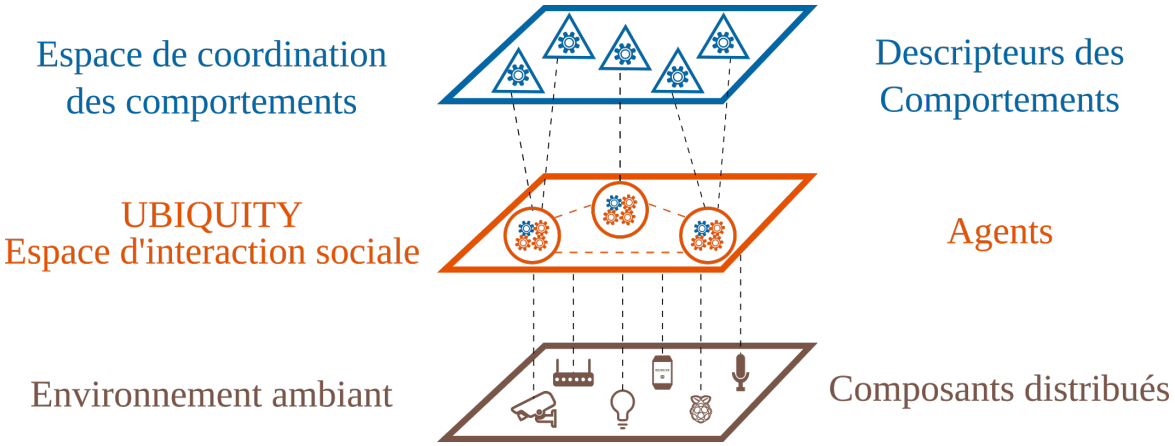


FIGURE 6.5 Ubiquity et mise en commun des comportements

est associé à un et un seul agent logiciel. Par contre, un agent peut piloter un ou plusieurs composants physiques. L'agent qui pilote un composant se trouve sur la partie logique de celui-ci. Ils sont donc encore distribués et isolés les uns des autres.

Nous déployons ensuite Ubiquity sur cette partie logique. Ubiquity génère un espace d'interaction. Chacun de ces agents reçoit une représentation virtuelle dans cet espace. C'est ce qui correspond à la couche du milieu dans la figure 6.5. Nous retrouvons ici la représentation de l'agent avec ses quatre unités comme dans la figure 4.4 précédente. Nous représentons particulièrement en bleu l'ensemble de ses comportements étant donné que nous nous intéressons particulièrement ici au comportements. Les agents se voient et interagissent entre eux par le biais de leur représentation, un peu à l'image des communautés d'utilisateurs qui interagissent au sein d'un réseau social en utilisant un avatar virtuel.

Ubiquity est une plateforme de création de plusieurs espaces d'interaction sociaux distincts. *Ubiquity* sépare les échanges entre agents en fonction de la thématique qu'ils traitent. Là aussi, l'exemple des réseaux sociaux thématiques donne une bonne image de la finalité recherchée à travers cette possibilité. C'est ce qui correspond à la couche supérieure où les descripteurs des comportements sont présentés. Par souci de lisibilité de la figure, nous présentons uniquement ici un espace thématique au lieu de plusieurs. Nous pouvons tout à fait en avoir plus d'un pour un même seul espace d'interaction. L'objectif est de séparer les échanges entre agents en fonction de la thématique qu'ils traitent. Parmi ces espaces d'interaction, nous nous intéressons à un espace particulier. Il s'agit de l'espace de coordination des comportements. C'est dans cet espace de coordination que les agents exposent ses comportements à vocation collective. C'est ce qui correspond à la couche supérieure sur la figure 6.5. Dans cette couche nous décrivons les comportements avec les triangles en blue. Les comportements restent dans l'agent mais possèdent des descripteurs permettant leur coordination.

Représentation de l'agent

Nous représentons par le diagramme de classes de la figure 6.6 les classes principales de Ubiquity. Nous nous intéressons particulièrement ici aux classes représentant un agent. La librairie est constituée de plusieurs packages. Nous nous intéressons uniquement ici au package gérant la représentation de l'agent dans le space.

Pour tirer profit de la plateforme, *Ubiquity* offre des outils d'interaction aux agents. Pour cela, un agent est représenté par cinq supports. Ce sont : le *Mailbox* pour l'échange de message, l'*Alias* pour l'identité, le *State* pour l'état, le *Skill* pour les compétences, et l'*Activity* pour les activités. Ces supports encore appelés "facette" décrivent un agent dans l'espace d'interaction sociale. Dans notre travail, nous nous intéressons particulièrement aux comportements. Au lieu d'utiliser ces 5 facettes, nous en utilisons principalement deux : *Alias*, et *Skill* et *Mailbox*. L' objectif du comportement est décrit dans *Skill*. Le *Mailbox* intervient dans la réception et l'envoi des messages. Il remplit ici le rôle d'interface d'écoute et d'interface d'envoi également.

Dans *Ubiquity*, Les agents se voient et interagissent par des représentations particulières. Pour cela, la librairie offre deux interfaces spécifiques pour cette interaction : *Avatar* et *Participant*.

— L'interface *Avatar* représente l'ensemble des facettes définissant l'agent. Un agent est donc représenté par un et un seul *Avatar* dans le space. Nous avons ici le même image que comme au sein d'un réseau social où les utilisateurs interagissent en utilisant un avatar virtuel.

— L'interface *Participant* donne l'accès en lecture des information décrites dans l'avatar. Ses méthodes retournent par exemple les valeurs associées aux *Skill*, *State*, *Alias*.

Caractéristiques

Dans notre plateforme, l'interaction ne se limite pas uniquement à la communication explicite comme dans d'autres architectures logiciels tels que Corba, JavaSpace, ou Erlang. L'interaction inclut ici à la fois les interactions explicites, comme l'échange de message, et les interactions implicites résultants de la propagation d'information produite par l'observation automatisée des changements dans la représentation des agents (pattern observeur/observé). La topologie de cet espace est dynamique et supporte l'ajout et la suppression de nœuds du réseau à tout instant. Dans le cas d'une interruption de connexion du réseau physique, l'espace d'interaction sociale sera partitionné. Les partitions fusionnent ensuite pour former un espace unique une fois que la connexion est rétablie. Une autre particularité de cet espace est qu'il n'intervient pas directement dans le contrôle des agents. Toute l'organisation et tout le contrôle sont gérés au sein de l'agent.

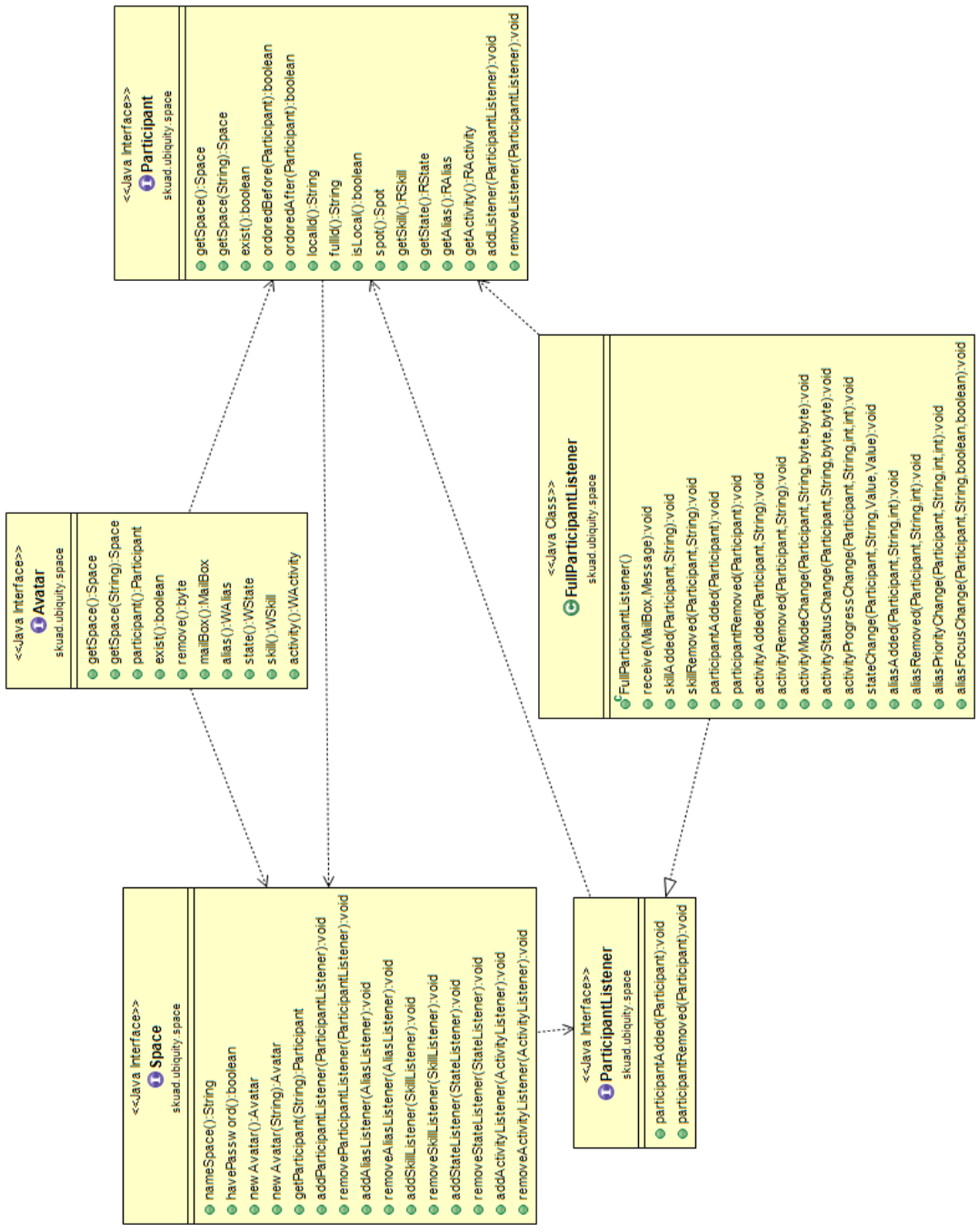


FIGURE 6.6 Ubiquity Diagramme de classes

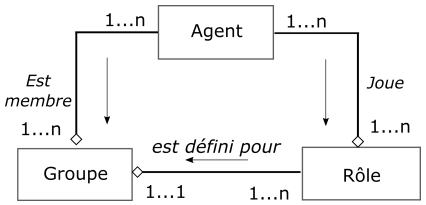


FIGURE 6.7 Modèle Agent Groupe Rôle (AGR)

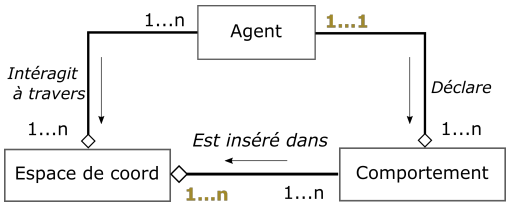


FIGURE 6.8 Modèle Agent et Espace de Coordination

6.2.3 Similarité avec l'architecture *Agent Groupe Rôle* (AGR)

Notre architecture présente une similarité avec l'architecture *Agent Groupe Rôle* (AGR) [FGM03] [GF00] représentée par la figure 6.7. AGR est également utilisée pour des systèmes ouverts et dynamiques [MF07]. Plus précisement, cette similarité concerne d'un côté le *Groupe* et l'espace de coordination et de l'autre côté le *Rôle* et le comportement. Le *Groupe* est défini comme un ensemble formé par les agents ayant les mêmes caractéristiques. Le *Groupe* apporte une organisation et une structuration des agents. Dans notre modèle nous utilisons l'espace de coordination qui est un ensemble formé par les comportements des agents différents mais participant à un service. Cet espace constitue une base de ressources disponibles où les agents peuvent chercher des comportements.

Le *Rôle* est une représentation abstraite d'une fonction. L'agent peut choisir de jouer un rôle ou pas. Ce rôle est défini en fonction de sa position sociale. L'agent ne peut accéder aux services que par rapport à son rôle. Dans notre modèle, le comportement est défini en fonction des compétences propres à l'agent mais non en fonction de sa position sociale. Comme il n'y a pas de hiérarchie entre les comportements, les agents ont les mêmes droits d'accès aux services. Une autre différence entre les deux modèles réside dans la cardinalité entre les entités. Un *Rôle* est propre à un *Groupe* alors qu'un comportement peut être présent dans différents espaces de coordination. De même, un même *Rôle* peut être joué par un ou plusieurs agents. Dans notre cas, chaque comportement est propre à l'agent. Cela n'empêche cependant pas qu'un ou plusieurs agents présentent des comportements avec le même objectif. Les deux figures 6.7 et 6.8 montrent les deux modèles avec à la fois la similarité et la différence entre eux.

Nous avons montré dans cette section un exemple de plateforme de mise en commun des ressources pour les agents. Cette étape est nécessaire pour constituer le collectif lorsque les moyens d'interaction directe n'existent pas (comme le portail web par exemple).

6.3 Organisation de l'architecture globale

Nous montrons dans cette section l'organisation interne de notre architecture globale. Nous illustrons son utilisation avec des exemples dans les sections qui suivent.

6.3.1 Aperçu général de l'architecture

La figure 6.9 montre le diagramme des packages correspondant à notre modèle d'architecture globale. Nous avons regroupé les briques de notre architecture au sein de trois principaux packages : le package *Production* (en haut à gauche) pour les unités de production, le package *Modalities* (en haut à droite) pour les modalités et le package *Environment Management* (en bas à gauche) pour les interactions : la supervision et la transaction. Le quatrième package (en bas à droite) correspond à l'organisation interne de l'agent. Nous avons déjà décrit l'architecture agent dans les partie précédentes. Ici nous nous intéressons uniquement à sa cohésion avec l'architecture globale.

6.3.2 Package AgentManagement

Ce package a comme but de créer un agent avec ses quatres unités de code. La classe *Agent* est une composition des 4 classes représentant les 4 briques de son architecture. Nous avons une relation de composition parce l'instance des 4 classes qui définissent l'agent ne peut appartenir qu'à une seule classe *Agent*. Nous retrouvons ici l'autonomie de l'agent qui gère seul ses ressources et prend seul ses décisions. Ensuite, lorsque l'agent est supprimé, toutes les instances des 4 briques le seront aussi automatiquement.

Lorsque l'agent ne participe pas à une activité collective, les éléments de son code se limitent aux classes contenues dans le package AgentManagement. Ce qui veut dire que, les associations avec les autres packages n'existent pas. C'est uniquement lorsque l'agent est engagé dans une activité collective que ses unités de code s'enrichissent. Dans ce contexte, une nouvelle classe orientée activité collective va être crée en héritant de la classe de ces unités. Par exemple, une classe fille de la classe *Behaviour* va être créée. C'est la classe fille qui implémente ensuite les différentes interfaces du collectif. Dans le diagramme précédent, les unités de code représentant l'agent sont déjà les classes filles. Avec cette configuration, nous pouvons faire la distinction entre les codes natifs de l'agent au niveau individuel et les éléments de code rajoutés lorsqu'il participe à une activité collective.

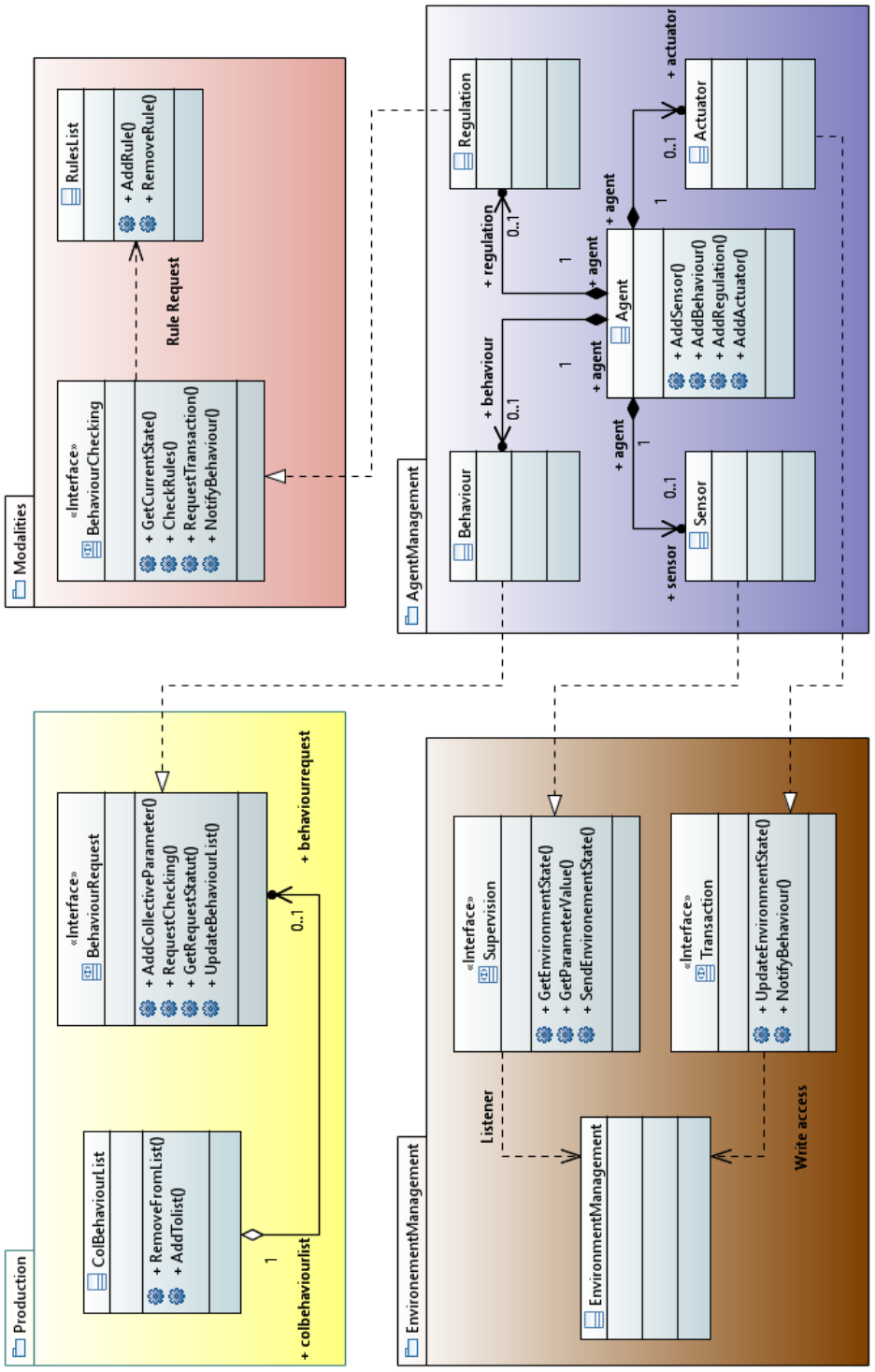


FIGURE 6.9 Diagramme de packages et Diagramme de classes de l'architecture globale

6.3.3 Package Production

Le package *Production* gère l'ensemble des comportements des agents qui participe à une activité collective. Nous identifions deux fonctionnalités principales dans ce package.

La première fonction consiste à lister les comportements participant à l'activité collective. Le comportement est retiré de la liste une fois que son exécution a pris fin. Dans la figure 6.9, cette première fonction est représentée par la classe *ColBehaviourList*. Une activité collective possède une et une seule instance de cette classe.

La deuxième fonction gère l'interaction avec le package *Modalities*. Ici nous représentons cette fonction avec l'interface *BehaviourRequest*. Son rôle est d'associer au comportement les paramètres sociaux pouvant influencer son exécution (*AddCollectiveParameter()*). Il l'ajoute ensuite sur la liste des comportements collectifs (*UpdateBheaviourList()*). La méthode *RequestChecking()* envoie le comportement pour une demande de validation. Le résultat de la vérification est suivi par la méthode *GetRequestStatut()*. Le retour attendu est alors soit validé, soit rejeté.

6.3.4 Package Modalities

Le package *Modalities* est chargée du contrôle d'applicabilité des comportements. Nous identifions également deux fonctions principales de ce package.

La première consiste à lister les modalités opératoires et de les décrire une par une sous forme de condition. La classe *RulesList* assure cette fonctionnalité. Les paramètres de la condition sont les paramètres définissant l'état de l'environnement. Ces conditions décrivent quand est-ce qu'un comportement peut être exécuté. Le type de retour est booléen.

La deuxième fonction s'occupe de la vérification des comportements par rapport aux modalités opératoires. Les méthodes relatives à cette fonction sont regroupées dans l'interface *BehaviourChecking*. Cette interface offre les méthodes pour récupérer l'état courant de l'environnement (*GetCurrentState()*), pour vérifier les règles (*CheckRules()*), pour demander l'exécution si les règles sont vérifiées (*RequestTransaction*) et pour notifier le comportement dans le cas de rejet (*NotifyBehaviour()*). C'est cette interface qu'une instance de la classe *Regulation* de l'agent implémente pour être en conformité avec les contraintes sociales.

6.3.5 Package EnvironmentManagement

Nous gérons dans le package *EnvironmentManagement* l'interaction avec l'environnement. En passant au niveau collectif, les classes pour les interactions de l'agent (Sensor, Actuator) implémentent les deux interfaces du package *EnvironmentManagement* : Supervision et Transaction.

La première interface *Supervision* joue le rôle de Listener pour l'écoute du changement d'état de l'environnement (*GetEnvironmentState*). Il peut également écouter uniquement le changement de valeur d'un paramètre en particulier (*GetParameterValue()*).

La deuxième interface *Transition* donne l'accès en écriture pour modifier l'état de l'environnement (*UpdateEnvironmentState()*). Elle notifie également le comportement à l'origine de l'action (*NotifyBehaviour*).

6.3.6 Remarques

Nous décrivons dans ces packages les fonctions principales de l'architecture. Ce modèle peut ensuite être enrichi en fonction de la complexité de l'activité collective.

L'environnement peut être représenté sous diverses formes. Cela peut être un environnement graphique, une carte ou bien une collection ou une liste en fonction des paramètres qui définissent l'état de l'environnement

Nous remarquons sur le diagramme des packages qu'aucun lien direct n'existe entre les classes de deux packages distincts. Par exemple, aucune classe ou interface du package *Production* n'est associée directement à celles du package *Modalities*. Toutes les associations passent par l'agent. Par exemple, la demande de validation d'un comportement collectif est envoyé par la Classe *Behaviour* et traitée par la classe *Regulation*. C'est uniquement les méthodes pour le faire qui proviennent des autres packages. Cela montre l'autonomie de l'agent dans ses activités, même collectives.

6.4 Modèle prédateur proie 1

Nous décrivons précédemment l'organisation interne de notre architecture globale. Dans cette section nous illustrons cette organisation avec l'exemple d'un cas d'école. Il s'agit du modèle prédateur-proie. Nous le choisissons pour montrer l'efficacité de notre modèle avec un exemple classique d'activité collective en SMA. L'objectif est de valider d'abord le modèle dans des cas connus. Nous continuons ensuite dans des cas plus pratiques dans la section 6.6 suivante.

6.4.1 Description

Le modèle proie-prédateur montre la co-évolution des proies et ses prédateurs. Dans ce modèle, les prédateurs se nourrissent des proies et se reproduisent. Les proies se nourrissent d'herbe et évitent les prédateurs. Si la chasse est trop importante, les proies disparaissent. Les prédateurs finissent alors par mourir de faim. De même, si la reproduction chez les prédateurs est trop importante, les proies ne vont plus suffire donc les prédateurs finissent aussi par disparaître. Il y a donc un équilibre à chercher dans ce modèle pour que les deux espèces co-evoluent.

Nous nous intéressons particulièrement aux prédateurs. Les proies représentent des ressources à gérer pour que l'espèce ne meure pas de faim. Dans ce contexte, nous retrouvons les problèmes de gestion de ressources limitées. Ce qui fait que nous pouvons utiliser notre modèle comme solution à ce problème.

Considérons l'activité collective qui consiste à pérenniser l'espèce. L'objectif collectif est d'éviter la famine. Cet objectif implique des organisations particulières. Des contraintes sociales particulières s'imposent par conséquent aux prédateurs. Nous nous proposons d'utiliser notre modèle d'architecture pour simuler les comportements des prédateurs. Pour cela, nous identifions les fonctionnalités pour chaque brique de l'architecture. Nous présentons ensuite l'organisation au niveau du code avant de finir avec le code de l'agent qui y en résulte.

6.4.2 Spécification de l'architecture globale

Nous montrons dans cette partie comment sont définies les fonctions correspondant aux briques de l'architecture.

Modalités opératoires

Nous partons de l'objectif collectif pour définir les modalités opératoires visant à atteindre l'objectif.

Ici l'objectif collectif est d'éviter la famine sur le long terme. Nous définissons deux règles pour atteindre l'objectif. La première consiste à ne pas gaspiller les proies. C'est-à-dire, ne pas chasser lorsque le niveau de nourriture du chasseur a déjà atteint un certain seuil. La deuxième revient à inhiber la fonction de reproduction lorsque les proies sont rares. Ces deux règles constituent notre modalité opératoire.

Avec ces deux règles nous constatons que la première, anti-gaspillage, dépend du niveau de satiété du prédateur. Ce paramètre n'est pas partagé entre les autres prédateurs. Chaque prédateur possède sa propre valeur de ce paramètre. Le comportement "se nourrir" qui est relatif à ce règle est donc géré indépendamment du collectif. C'est un comportement individuel.

La deuxième règle, qui est la limitation de la reproduction, dépend de la rareté des proies. Ce paramètre concerne tous les prédateurs. La valeur de ce paramètre évolue en fonction des comportements des prédateurs. Cette valeur est partagée entre tous les prédateurs. Ce qui fait que le comportement, "se reproduire" qui modifie ce paramètre est un comportement collectif. Son exécution dépend du collectif.

Au niveau du code de l'agent prédateur, ces deux modalités vont être décrites chacune dans la classe *Regulation*. Toutefois, la deuxième règle est à vocation collective. Ce qui implique que la classe qui la décrit doit implémenter l'interface sociale correspondante dans le package *Modalities*. Cette classe est une classe fille de la classe *Regulation*.

Environnement fonctionnel et Interface d'interaction

L'environnement fonctionnel représente l'évolution des activités par rapport à cet objectif. Les comportements des prédateurs sont influencés par la disponibilité des proies. Ce qui implique que l'environnement fonctionnel que nous observons est donc caractérisé par la disponibilité des proies par rapport aux prédateurs. Cette disponibilité définit son état. Nous avons donc ici besoin d'un support partagé entre les prédateurs pour représenter cet état de disponibilité des proies.

La solution la plus simple consiste à compter le nombre de proies et de partager l'information entre les prédateurs. Cependant comme notre prédateur n'a qu'une vision locale de l'activité, il ne peut pas fournir le nombre de proies. Ensuite, nous n'avons pas une visibilité sur le nombre de proies.

La solution que nous mettons en place consiste alors à faire participer directement les prédateurs pour signaler la rareté des proies. Nous proposons un environnement de "stress" qui regroupe l'état d'anxiété du prédateur lorsqu'il ne trouve pas de proie. C'est la preuve que les proies sont rares. En fonction de l'expérience de chacun dans la recherche de proie, l'état de l'environnement va être modifié. De cette manière, la mise à jour de l'état de l'environnement est distribuée entre les prédateurs. Par la suite avant de valider le comportement "se nourrir", l'état de cet environnement d'anxiété sera consulté.

En ce qui concerne les interfaces d'interaction, elles sont définies par rapport aux moyens permettant aux composants de s'informer de l'état de l'environnement et de le modifier. Chaque participant à l'activité collective a besoin d'accéder à cette information puisque c'est celle-ci qui détermine ses comportements.

Nous ajoutons donc ces moyens d'interaction à nos prédateurs. L'outil d'écoute est ici un capteur de niveau d'anxiété qui sera ajouté au *Sensor* du prédateur. Ce nouveau capteur implémentera l'interface *Supervision*. On ajoute également au prédateur un outil d'écriture. Il s'agit d'un émetteur d'anxiété qui implémentera l'interface *Transaction*.

Unités de production

L'unité de production est définie par l'ensemble des comportements en relation avec l'objectif.

Dans notre exemple, les comportements des prédateurs sont : se déplacer, se nourrir une fois qu'une proie a été détectée, se reposer, et se reproduire une fois qu'un partenaire a été identifié. Les comportements en relation directe avec l'objectif collectif sont *se nourrir*, et *se reproduire*. Le comportement *se nourrir* exploite directement les proies, tandis que le comportement *se reproduire* augmente le nombre de concurrents aux proies.

Forts de ces spécifications, nous pouvons représenter maintenant les différentes fonctionnalités pour chaque unité de code de l'agent dans le tableau 6.1. Les éléments précédés du

COMPORTEMENT	REGULATION
- Se déplacer - Se nourrir - Se reposer + Se reproduire	- Anti-gaspillage + Anti-surpopulation
CAPTEUR	**ACTUATEUR**
- Détecteur de proie + Détecteur d'anxiété + Détecteur de partenaire	- Dispositif de déplacement + Emétteur d'anxiété + Dispositif de reproduction

TABLE 6.1 Fonctionnalités relatives à chaque unité de code de l'agent prédateur

signe - correspondent aux éléments de l'agent prédateur qui n'interviennent pas directement dans l'activité collective. Les éléments mis à disposition pour le collectifs sont précédés du signe +. Dans le code ces éléments individuels et collectifs sont déclarés respectivement par *Private* et *Public*.

6.4.3 Synthèse

A travers cette section, nous montrons comment sont définies les différentes classes à partir des spécifications d'une activité collective. Cette étape de spécification de chacune des briques de l'architecture est une étape nécessaire avant toute implémentation. Grâce à la souplesse du code de l'agent, et à sa modularité, l'agent possède une représentation dans chacune des unités collectives.

6.5 Modèle prédateur proie 2

Nous présentons dans cette section un autre exemple d'implémentation du modèle prédateur proie. Cette fois-ci nous utilisons notre plateforme développée par notre équipe de recherche. Nous aborderons davantage ici la structuration du code par rapport aux fonctionnalités attendues.

6.5.1 La plateforme SKUAD

Notre équipe de recherche travaille sur le développement d'une plateforme sur laquelle nous menons notre implémentation. Cette plateforme, appelée SKUAD [3] est une librairie logicielle pour la création des applications manipulant des capteurs ou des objets connectés en réseau. Les ressources et les tutoriels sont accessibles ici [4]. Le développement de SKUAD ne fait pas partie de la thèse. Nous avons seulement participé à un ensemble de réflexion sur

3. Software Kit for Ubiquitous Agent Development
4. ou via ce lien http://skuad.onover.top/

sa spécification au cours de la thèse. Nous le présentons ici pour plus de compréhensions à l'implémentation que nous proposons.

La particularité de la librairie SKUAD qui nous intéresse particulièrement est la facilité de création d'un agent ambiant. La librairie offre des méthodes pour le pilotage de l'agent (start, stop,..), pour l'exécution de son comportement, et surtout pour la gestion des équipements physiques ou virtuels associés à cet agent. La notion de "*device*" représente ces équipements physiques ou virtuels. La création d'un agent est possible grâce à un serveur d'exécution d'agent ubiquitaire, représentée par une classe appelée *ServerAU* (Serveur d'Agent Ubuquitaire). Un autre point clé de la plateforme est sa souplesse pour faire tourner des simulations. Une des fonctions proposées qui nous semble pertinente est la création d'environnement virtuel servant de cadre d'exécution d'une simulation.

6.5.2 Structuration des environnements

C'est avec la plateforme SKUAD que nous proposons maintenant une implémentation du modèle proie-prédateurs. Nous gardons les mêmes descriptions de l'activité collective décrites précédemment dans la section 6.4.1. Nous nous appuyons ici sur les environnements virtuels proposés par la plateforme pour mieux structurer le déroulement de l'activité.

Avec le même scenario, notre idée ici c'est de gérer chaque thématique de l'activité collective dans un environnement virtuel distinct. Nous identifions trois principales fonctionnalités : le déplacement pour la recherche de proie, la recherche de partenaire pour la reproduction, et la transmission du niveau de stress pour signaler l'insuffisance de proies. Nous décidons alors d'associer à chaque fonction un environnement virtuel. Pour une meilleure visibilité de la simulation, nous utilisons un environnement graphique de grille pour représenter ces environnements. Le déplacement de l'agent s'effectue alors dans les cases de la grille. Nous représentons avec la figure 6.10 la répartition en différents environnements des activités de l'agent avec ses différentes capteurs (à gauche) et actuateurs (à droite) pour chaque environnement.

L'environnement de recherche de proie

Cet environnement est représenté par le rectangle en bas sur la figure 6.10. Nous y gérons deux fonctions : le déplacement et la recherche de proie. L'évolution de l'état d'anxiété et donc de la reproduction est conditionnée par l'issue de la recherche de proie. Ce qui fait que cet environnement sert de base pour les deux autres. Cette recherche de proie inclut l'action de se reposer qui est tout simplement l'inactivation de la fonction de déplacement.

Nous avons ici un environnement physique. Notre agent prédateur y enregistre ses interfaces physiques d'interaction : un détecteur de proie comme capteur, et un dispositif de déplacement comme actuateur. La recherche de nourriture est soumise à une contrainte sociale anti-gaspillage. Cependant comme nous l'avons vu, cette règle dépend d'un paramètre

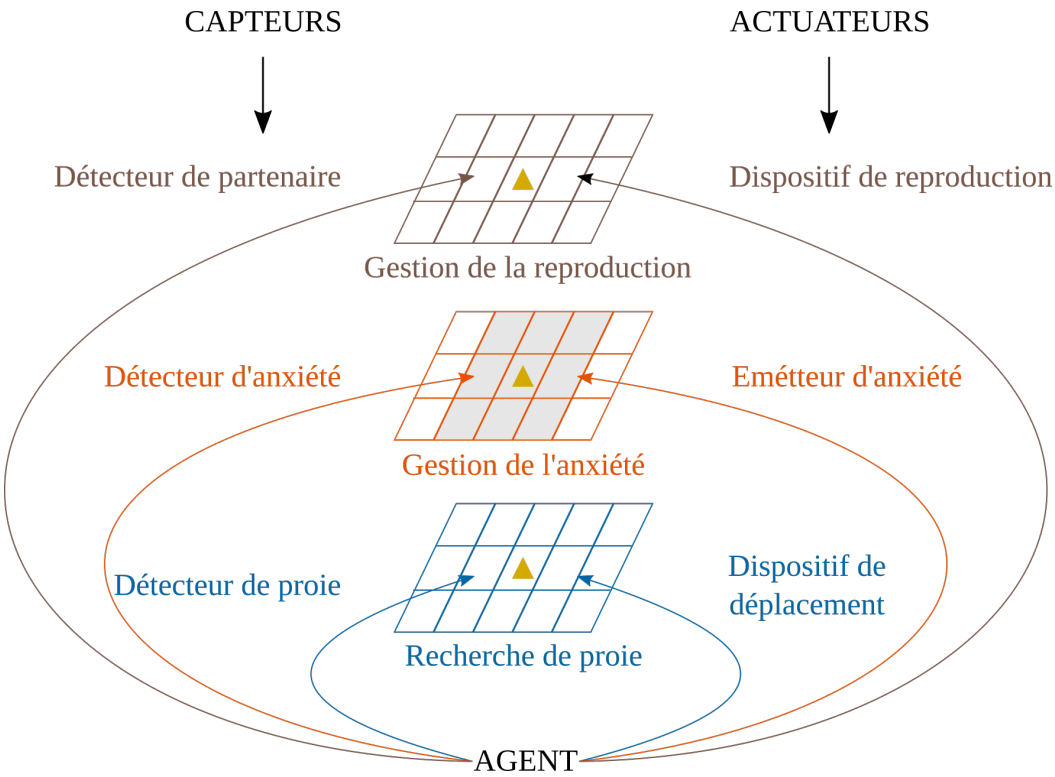


FIGURE 6.10 Répartition des activités en trois environnements avec les interfaces de l'agent

propre à chaque prédateur qui est le niveau de satiété. Cela signifie que cette activité est un ensemble d'activités individuelles en parallèle entre les prédateurs. Quand le prédateur atteint un certain niveau de nourriture, il va chercher à activer ses moyens de reproduction. Ce souhait d'activation est encore soumis à l'état de l'environnement d'anxiété.

L'environnement de gestion de l'anxiété

Cet environnement concrétise l'état de l'évolution de la recherche de proie, donc de la disponibilité des proies. Il est représenté par le rectangle au milieu sur la figure 6.10. Le paramètre clé de la modalité opératoire est visible sur cet environnement. Chaque prédateur qui n'a pas pu trouver une proie pendant une certaine durée diffuse son anxiété dans cet environnement. Si l'expérience est répétée par un autre prédateur, l'état de cet environnement va inhiber l'activation des dispositifs de reproduction pour tous les prédateurs. Nous représentons le niveau d'anxiété d'un prédateur par des éléments graphiques qui entourent son premier voisinage de Moore [5], comme nous pouvons voir sur les cases grisées sur la figure 6.10.

Cet environnement possède deux états exclusifs. Son état est *favorable à la reproduction* quand il y a au plus un prédateur en état d'anxiété. A partir d'un deuxième prédateur,

5. Les huits cases adjacentes à la case où le prédateur se trouve

l'expérience est répétée donc l'état de l'environnement passe en *défavorable à la reproduction*. Nous représentons son état avec un booléen que les prédateurs consultent avant d'activer ou pas ses dispositifs de reproduction. Une fois qu'une proie a été détectée, le niveau d'anxiété du prédateur redevient nul. Toutefois, l'environnement ne retrouve son état favorable qu'une fois qu'il y a au plus un prédateur en état d'anxiété.

Les interfaces de l'agent prédateur dans cet environnement sont un détecteur et un émetteur d'anxiété. Lorsque cet environnement est en état favorable à la reproduction, et que le niveau de nourriture requis est vérifié, l'agent peut maintenant activer ses moyens de reproduction.

L'environnement de gestion de la reproduction

Le rectangle du haut sur la figure 6.10 correspond à cet environnement. Il est inaccessible au prédateur tant que son niveau de nourriture individuel, et l'état d'anxiété social ne le permettent. L'accès à cet environnement se traduit par l'activation de ses moyens de reproduction : un détecteur de partenaire et un moyen de production. Sur la figure, le prédateur se trouve dans un état d'anxiété. Ce qui signifie que ses interfaces de reproduction ne sont pas activées. Une fois qu'un partenaire a été détecté, la fonction de reproduction est réalisée. Cet environnement notifie l'environnement de recherche de proie. Un nouvel agent prédateur y est ajouté. Les moyens de reproduction des deux partenaires redeviennent inactifs.

Les activités dans les deux derniers environnements sont collectives. Elles se basent sur l'interaction entre les prédateurs. Les 3 environnements sont liés. Pour cela, nous proposons un "observeur" dans chaque environnement pour faire passer les informations. Cette structuration donne une gestion plus ou moins indépendante de chaque fonction, dans le but d'une réutilisation dans d'autres activités. Elle met également en évidence le contrôle de l'applicabilité du comportement de reproduction par l'environnement de gestion d'anxiété.

6.5.3 Synthèse

Nous proposons dans cette section une implémentation basée sur la répartition en plusieurs environnements. Une telle structuration donne une gestion séparée des différentes fonctionnalités, tout en leur permettant de les faire interagir. L'environnement d'anxiété représente la modalité opératoire collective. Si nous supprimons cet environnement, l'activité collective des prédateurs ne passe plus par une vérification certes mais peut toujours fonctionner sans blocage. C'est-à-dire que la fonction de reproduction est activée dès que le niveau d'énergie nécessaire est atteint sans prise en compte des contraintes sociales. Dans le meilleur des cas où les proies sont abondantes nous pouvons donc nous passer facilement de l'environnement d'anxiété. Par contre, lorsque ce n'est plus le cas, l'environnement d'an-

xiété trouve toute sa pertinence pour gérer l'exploitation des proies. Dans ce contexte, nous remarquons qu'avec ou sans les modalités opératoires, le code de comportement de l'agent reste le même. Nous en tirons donc deux conclusions :

— La modalité opératoire peut être retirée à tout moment sans entraîner des blocages lorsque l'activité collective se trouve dans le meilleur des cas. Cela montre la souplesse du modèle

— L'adaptabilité de l'activité collective ne réside pas dans le code de comportement, parce que ce code ne change pas dans les deux situations. L'adaptabilité vient de l'ajustement des actions futures en fonction de l'état courant de l'environnement

6.6 Modèle véhicule-borne

Nous proposons dans cette section un exemple d'implémentation pour l'exemple de rechargement de véhicules électriques que nous avons vu dans le chapitre 5 précédent. La description de chacune des fonctionnalités attendues pour le véhicule et pour la borne dans la partie 6.6.1 nous donne les éléments nécessaires pour structurer le code correspondant dans la partie 6.6.2.

6.6.1 Identification des fonctionnalités des composants

Nous décrivons dans la section 5.2 précédente l'exemple du chargement du véhicule électrique. Pour rappel, la recharge de véhicule électrique est constituée par deux composants principaux. Il s'agit des voitures électriques et de la borne de recharge. Le véhicule a comme objectif de faire le plein d'énergie. Tandis que la borne a comme objectif de recharger chaque véhicule qui la sollicite. L'activité collective entre ces deux composants consiste à optimiser le processus de chargement en limitant le temps d'attente. Cet objectif collectif implique pour la borne un chargement à 80% lorsque le nombre de véhicules en attente atteint un certain seuil k, et un rejet de la demande d'ajout de nouveau véhicule sur la liste d'attente lorsque le nombre de véhicules atteint un autre seuil n, n>k. Du côté du véhicule, le chargement à 80% l'oblige à être économe, tandis que le rejet de la demande d'enregistrement active le choix d'un autre créneau. La figure 6.11 nous montre pour chaque composant les activités s'opérant dans chaque composant pour cette activité collective.

A gauche, nous avons le diagramme d'activités du véhicule et à droite le diagramme d'activités de la borne. Chaque rectangle représente une fonction. Les activités correspondant à chaque composant ne sont pas dissociées. La suite des activités de l'un dépend des activités précédentes de l'autre. Chaque composant est autonome mais leur décision dépend des contraintes sociales. Les conditions correspondant aux modalités opératoires sont représentées explicitement sur les nœuds de décision (losange en orange). Pour le véhicule, ses activités dépendent des deux conditions. La première condition est la validation de son

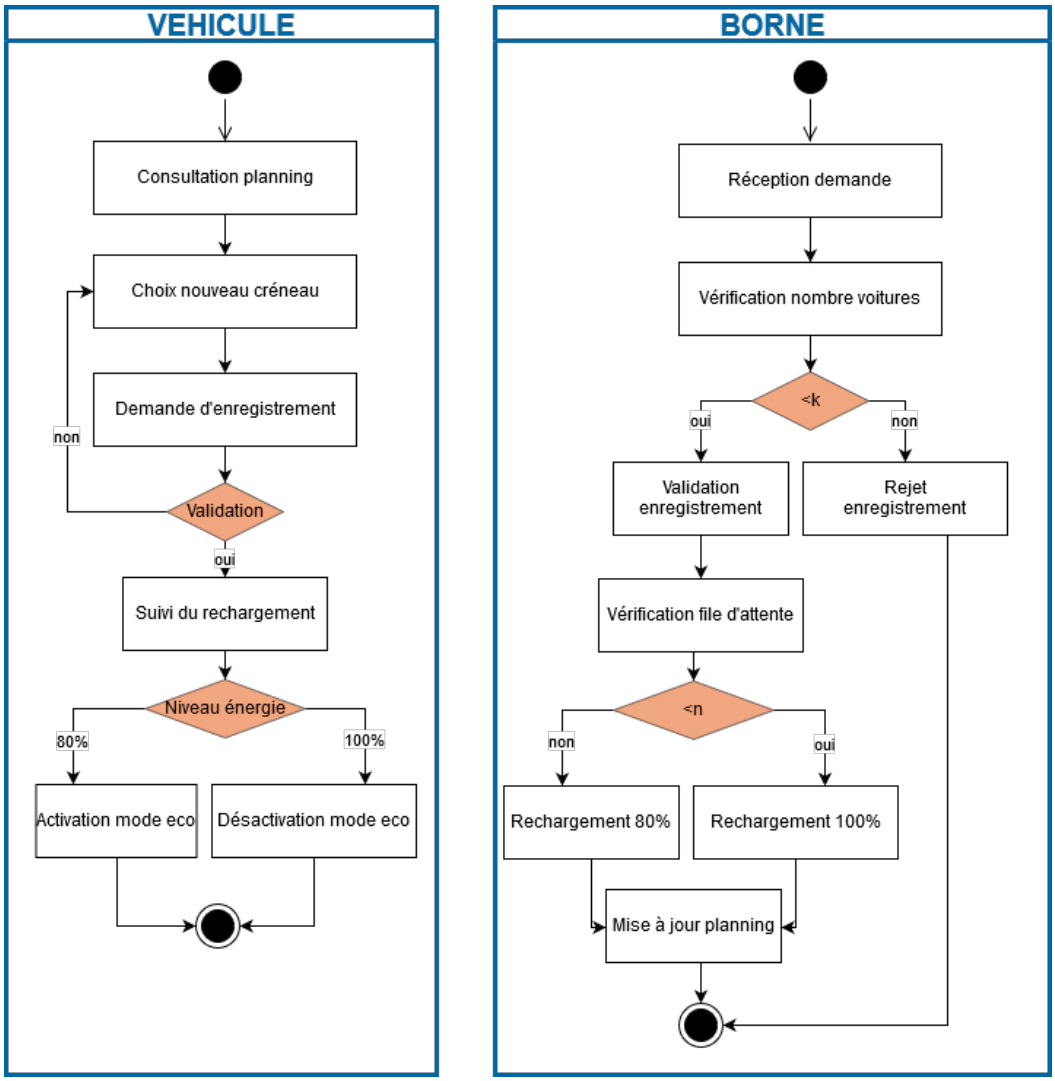


FIGURE 6.11 Diagramme d'activités du véhicule (à gauche) et diagramme d'activités de la borne (à droite)

enregistrement sur le planning. La deuxième condition est le niveau d'énergie reçu. Pour la borne, les deux conditions d'évolution de ses activités reposent sur la valeur du nombre de véhicules sur la file d'attente. La première condition correspond à une valeur au delà de laquelle l'enregistrement du véhicule sur le planning est rejeté. La deuxième condition correspond à une autre valeur au delà de laquelle le mode de chargement change. Avec ces fonctionnalités identifiées, nous pouvons maintenant définir comment les organiser pour mieux les coder.

6.6.2 Répartition en différentes unités de gestion

En analysant les nœuds de condition dans le diagramme de la figure 6.11 nous constatons qu'ils sont définis par deux paramètres : soit le nombre de véhicule sur le planning, soit le niveau d'énergie rechargé. Pour mieux gérer les activités liées à ces paramètres, nous proposons de répartir les fonctions relatives à ces paramètres dans des unités de gestion séparées. Pour cela, nous créons une unité de gestion de planning pour le suivi du nombre de véhicules, une unité de gestion de rechargement pour juger du mode de rechargement et une unité de gestions des règles pour conditionner l'évolution de ces deux aspects à gérer.

Gestion du planning

Le planning de la borne est mis à disposition pour que le véhicule puisse consulter et s'enregistrer. La borne met ensuite à jour ce planning lorsqu'un véhicule a été rechargé. Les deux composants interagissent chacun avec ce planning. Ils ont tous les deux un accès en écriture et en lecture. L'évolution de la suite de l'activité dépend de l'état de ce planning. Ce dernier constitue donc l'environnement fonctionnel. Cette unité de gestion constitue donc l'unité d'interaction avec le planning. Le diagramme de classes sur la figure 6.12 nous montre la gestion du planning. Nous retrouvons pour chaque composant les interfaces d'interaction *Supervision* et *Transaction*. Chacun des véhicule et borne l'implémente. Nous gérons ensuite dans une classe à part l'état du planning qui traduit l'état de l'environnement. Tout objet de la classe *Planning* possède obligatoirement un et un seul objet de la classe *PlanningStatut*.

Gestion de rechargement et des règles

Le rechargement constitue l'activité principale entre la borne et le véhicule. Cette unité de gestion regroupe toutes les fonctions relatives au chargement proprement dit. Il s'agit pour le véhicule de l'activation ou non de mode économique en fonction du niveau d'énergie. Pour la borne, il s'agit du rechargement à 80% ou à 100% en fonction du nombre de véhicules. Nous retrouvons ainsi les comportements qui constituent l'unité de production. De même, nous retrouvons les modalités opératoires dans ces conditions. La figure 6.13 nous montre le diagramme de classes correspondant au chargement et aux différentes modalités.

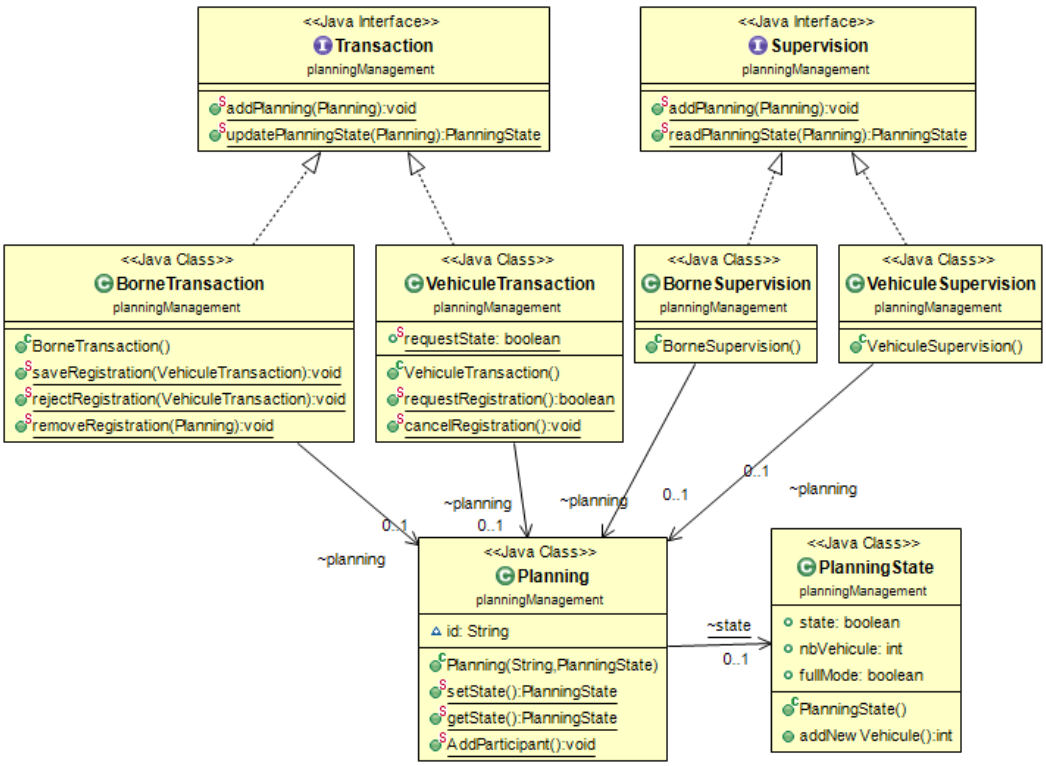


FIGURE 6.12 Diagramme de classes pour la gestion du planning

Nous retrouvons sur l'en-tête de chaque classe le package correspondant à l'unite auquelle la classe appartient : *modalities* pour les règles, *planningManagement* pour le planning, et *charging* pour le rechargement. Nous pouvons voir directement sur le diagramme que chaque unité de production (*BorneRecharging* et *VehiculeRecharging*) sont liées à une unité de règle (respectivement *BorneRule* et *VehiculeRule*). Ces deux dernières classes implémentent l'interface *Rules*. Les trois classes au milieu sont les mêmes que nous avons vu sur la figure 6.12. Nous les reprenons ici dans la figure 6.13 pour montrer les relations entre les différentes classes. Nous remarquons que les unités de production (*BorneRecharging* et *VehiculeRecharging*) n'accèdent pas directement à l'environnement (Planning). Le lien entre ces unités et celles des modalités opératoires indique la demande de vérification des règles avant l'application d'un comportement.

Le type de retour de la valeur retournée par chaque méthode des règles est booléen. Cette valeur est par défaut égale à *true*. C'est uniquement lorsqu'une condition n'est pas vérifiée qu'elle devient *false*. Ce qui implique que dans les meilleurs cas où nous n'avons pas besoin de modalités opératoires (ressources illimités, ou faible fréquence de véhicule pour se recharger) nous pouvons laisser cette valeur par défaut et sans modifier tout le reste du code.

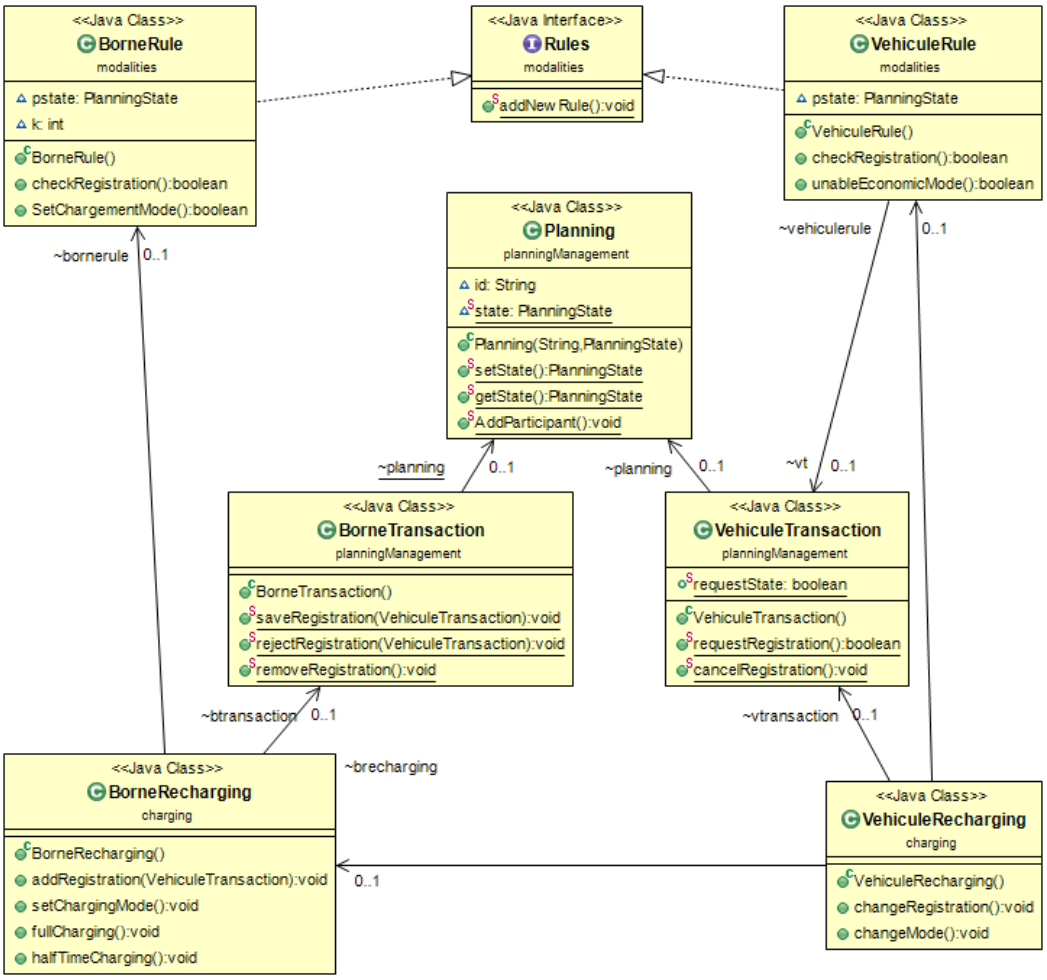


FIGURE 6.13 Diagramme de classes pour la gestion du chargement incluant les modalités

6.6.3 Synthèse

Dans cette section nous proposons une implémentation du modèle véhicule borne. Contrairement à l'exemple des proies-prédateurs précédents, nous avons ici deux types d'agents différents ayant chacun leur organisation interne. Cet exemple est caractérisé par la diversité des composants. En implémentant notre modèle uniforme, nous parvenons à les faire collaborer de manière uniforme. Nous avons représenté ici le véhicule comme étant un agent. Le but est d'avoir une version en simulation étant donné l'impossibilité de réaliser l'expérimentation à l'échelle d'une borne et d'un véhicule en système ambiant. Toutefois, le modèle est utilisable dans cette condition réelle dans la mesure où les unités liées au véhicule sont gérées par une application mobile.

6.7 Conclusion

Nous montrons à travers ce chapitre le passage de notre modèle conceptuel au code informatique exécutable. Pour cela, nous présentons des exemples d'utilisation. Nous simulons le déplacement d'un robot agricole pour illustrer la flexibilité du niveau local dans la section 6.1. L'importance de l'interaction dans le collectif nous conduit à la conception d'une plateforme d'interaction sociale dans la section 6.2. Fort des éléments présentés dans ces sections, nous construisons les bases de notre objectif principal qui est l'optimisation des activités collectives. Dans ce contexte, nous décrivons les fonctionnalités requises pour chaque brique de l'architecture dans la section 6.3. Nous l'illustrons ensuite avec des exemples d'applications dans les sections suivantes.

Nous tirons deux conclusions majeures de cette implémentation. Le premier point concerne l'algorithme d'adaptabilité. Nous constatons de manière concrète que la flexibilité au niveau local, et l'adaptabilité au niveau global, ne résident pas dans le code de comportement lorsque l'on s'intéresse à un environnement ambiant. Ces propriétés résultent surtout de la capacité du niveau considéré à vérifier l'applicabilité de ce comportement avant son exécution et de sa capacité à ajuster ses comportements futurs. Le deuxième point essentiel constaté est l'uniformité et la compatibilité des deux modèles. Le modèle a montré son efficacité au niveau local, ensuite au niveau global. Toutefois, nous remarquons qu'une fois combinés dans une seule implémentation les deux modèles sont compatibles et s'imbriquent sans problème d'intégration.

Ce chapitre fournit les spécifications des codes pour le développement logiciel de notre contribution. Nous les illustrons avec notre plateforme de développement. Toutefois, les éléments présentés dans ce chapitre présentent tout aussi bien une implémentation indépendante de notre plateforme. Ce qui implique qu'ils sont réutilisables dans d'autres applications avec d'autres plateformes.

Chapitre 7

Conclusion et perspectives

Nous proposons dans cette thèse une solution simple pour rehausser le niveau d'adaptation des systèmes informatiques en milieu ambiant. Nous développons cette solution dans trois modèles complémentaires : un modèle générique constituant le pattern général, un modèle pour le niveau local et un autre pour le niveau global. La contribution consiste à prévenir les perturbations de l'éxécution face aux dynamiques du cadre opératoire. En évaluant l'applicabilité des décisions des comportements, nous pouvons inhiber celles qui conduisent à des troubles de l'exécution. Nous obtenons alors l'adaptation par ajustement suite aux résultats de l'évaluation.

Contributions

Ce travail de thèse s'inscrit dans l'optimisation des activités collectives des systèmes informatiques opérant dans un environnement ambiant. A la fois hétérogène et fortement dynamique, ce cadre applicatif est hors de contrôle. Cette propriété conduit à deux verrous (chapitre 1.2). Le premier verrou résulte d'une dynamique interne. L'hétérogénéité des composants est susceptible de rompre la cohésion de leur activité collective. Le deuxième verrou résulte de la dynamique externe. Les événements extérieurs imprévisibles perturbent le fonctionnement normal des activités collectives. Ces deux verrous rendent difficile la conception de l'architecture pilotant ces systèmes informatiques. Nous devons donc lever ces verrous pour être efficace dans notre cadre applicatif ambiant. L'état de l'art présenté dans le chapitre 2 nous montre que ces deux verrous renvoient à un problème d'adaptation interne et externe. L'analyse des différents approches (section 2.1.3), méthodes et outils (section 2.2.3) relatifs aux problèmes d'adaptation nous donne les supports nécessaires pour concevoir notre contribution visant à lever ces verrous. Cette contribution est constituée de trois parties : un modèle générique, un modèle d'architecture interne pour le niveau local, et un modèle de coordination des activités collectives pour le niveau global. Nous résumons

comme suit chacune de ces contributions.

La contribution principale de cette thèse est un modèle générique d'architecture appelé GMAS (chapitre 3). Son objectif est d'optimiser la capacité d'adaptation des agents logiciels opérant en environnement ambiant. Compte tenu des caractéristiques de ce cadre applicatif, il nous est nécessaire d'éviter les situations indésirables voire dangereuses (incohérence dans les activités collectives et perturbation du fonctionnement normal). Le modèle proposé s'inscrit ainsi dans une démarche de prévention. Dans ce contexte, le point essentiel du modèle est le mécanisme de filtrage des décisions issues des comportements avant leur exécution. Dans l'architecture classique, ces décisions de comportements accèdent directement aux moyens d'action.

Dans ce travail, dans l'optique d'une sureté de l'exécution, nous ajoutons une brique intermédiaire entre les décisions issues des comportements et les actions. Le rôle de cette brique est de vérifier si les conditions extérieures courantes sont favorables à l'exécution de ces décisions. Cette brique est appelée "Contrôle de l'applicabilité". Il s'agit d'un ensemble de règles et de conditions garantissant la sureté de l'exécution en fonction de l'état courant du cadre opératoire. Si les décisions issues des comportements vérifient les règles d'applicabilité, elles sont validées. Elles deviennent des actions concrètes se réalisant dans le cadre opératoire. Dans le cas contraire, ces décisions seront rejetées, aucune action correspondante n'aura lieu dans le cadre opératoire. Dans les deux cas, le comportement ayant émis les décisions sera notifié du devenu de ses décisions. L'adaptation résulte alors de l'ajustement des futures décisions par rétroaction suite à la validation ou non des décisions précédentes.

Fort de ces propriétés, GMAS lève les deux verrous identifiés plus tôt. Par rapport au premier verrou, le contrôle d'applicabilité se fonde sur les règles de cohésion des activités collectives. Toutes décisions allant à l'encontre de cette cohésion ne seront pas validées. Par rapport au deuxième verrou, il offre une méthode de prévention des situations néfastes. Il inhibe les décisions inopportunes de comportements par rapport à l'état courant du contexte opératoire. Ce pattern général GMAS est instancié dans deux niveaux différents. Le premier niveau est le niveau local où nous nous intéressons aux composants individuels. Le deuxième niveau est le niveau global où nous nous intéressons au collectif formé par ces composants. L'instanciation est alors spécifiée par rapport aux besoins et caractéristiques propres à chaque niveau.

L'objectif de la thèse est l'optimisation des activités collectives. Toutefois, nous nous intéressons particulièrement au niveau individuel dans cette thèse (chapitre 4). Ce qui constitue la deuxième contribution de la thèse. Cet intérêt sur l'individuel est justifié par deux raisons principales : l'équilibre avec le collectif et la gestion des ressources au niveau local. Dans la première raison, nous estimons qu'il est nécessaire d'instaurer un équilibre entre l'adaptation individuelle et l'adaptation collective. Dans la littérature, l'adaptation

individuelle est basée sur des techniques de raisonnement et d'apprentissage nécessitant un niveau cognitif conséquent. Nos composants sont miniaturisés et dotés d'une faible capacité de calcul. C'est pour cela que nous proposons à ce niveau une "flexibilité" qui correspond mieux à la réactivité de ces composants. La flexibilité traduit ici la souplesse de composant par rapport aux imprévus. La deuxième raison pour le choix de l'individuel vient du fait que même engagé dans une activité collective, un comportement reste dépendant du composant qui le porte. En effet, c'est le composant qui fournit les interfaces et les moyens pour la réalisation des actions correspondant aux comportements. Si nous souhaitons donc exploiter les comportements, nous devons tenir compte des contraintes locales auxquelles le comportement est soumis. Ces deux raisons nous montrent que le niveau local est un niveau à considérer avant de passer au niveau collectif.

Avec GMAS, nous proposons au niveau individuel un modèle d'architecture interne d'agent appelé MECA. Le contrôle d'applicabilité correspondant est une unité de régulation. Nous avons vu qu'au niveau local, nous nous intéressons à l'exploitation des comportements par rapport aux ressources nécessaires. Dans ce contexte, l'unité de régulation proposée filtre les décisions issues des comportements en fonction des règles d'usage des unités d'action et des règles de sureté de l'environnement physique dans lequel l'action correspondante se réalise. Pour cela, nous internalisons le Modèle Influence-Réaction dans l'agent. Les comportements n'ayant plus un accès direct aux actuateurs, envoient des souhaits d'actions à destination de la régulation. Ces souhaits d'actions sont les influences. C'est uniquement, lorsque la régulation valide l'influence qu'elle envoie une demande d'action à destination de l'actuateur concerné. Dans ce cas, l'action correspondante constitue la réaction externe. Tandis que le rejet d'un comportement correspond à la réaction interne. Les réactions internes et externes constituent les paramètres d'ajustement des futures influences. Avec la régulation, nous prenons en compte les évolutions de l'environnement sans qu'il soit nécessaire de les prévoir précisément dès la phase de conception.

Une fois que nous avons atteint la flexibilité individuelle, nous gérons le niveau global (chapitre 5). La proposition d'un modèle d'architecture de coordination des activités collectives constitue la troisième contribution de cette thèse. Notre niveau global n'est pas un ensemble de composants individuels regroupés avec chacun leur objectif isolé. Il s'agit d'un ensemble de composants évoluant vers un objectif commun. Les conditions nécessaires pour atteindre cet objectif commun sont l'organisation et l'interaction entre les composants. Ce sont les forces du collectif. Par conséquent, au niveau global, nous ne nous intéressons pas à la somme des contraintes de chacun des individus. En effet, tout ce qui est propre à l'individu est géré au niveau local, notamment l'interaction avec l'environnement physique. Ici nous nous intéressons sur ce qui est propre au collectif. Il s'agit des contraintes relatives à l'organisation et à l'interaction. Nous avons donc ici un environnement fonctionnel. C'est

un environnement représentant l'évolution des activités collectives par rapport à l'objectif collectif.

L'organisation du collectif est basée sur l'objectif qu'il souhaite atteindre. Pour cela, des règles sont définies pour que les actions entreprises par chacun des participants aillent dans le sens de l'objectif commun. C'est dans ce contexte que s'inscrit l'instance du contrôle d'applicabilité au niveau global. L'ensemble des règles de réalisation de l'activité globale est défini dans les modalités opératoires. Ce sont ces modalités opératoires qui constituent l'interface de filtrage des décisions des comportements. Toujours avec le même principe de GMAS, l'unité de modalités opératoires va vérifier si les décisions des comportements peuvent être appliquées compte tenu de l'état de l'environnement fonctionnel. C'est-à-dire compte tenu de l'évolution vers l'objectif collectif.

La particularité de notre modèle au niveau global est son caractère distribué. En effet, toutes ces organisations et ces modalités opératoires ne sont pas gérées par de nouveaux éléments supplémentaires. Elles sont distribuées au sein de chacun des composants. Certes, les modalités opératoires sont définies pour l'ensemble du collectif mais elles sont réparties ensuite dans chacun des composants concernés. L'uniformité du modèle GMAS fait que le niveau local possède un point d'encrage pour les modalités opératoires dans son unité de régulation. Ainsi, dans son architecture interne, le composant individuel possède des sous-unités destinées à l'activité collective. Chaque composant ajuste alors sa décision par rapport aux modalités opératoires. C'est de cette manière que l'adaptation collective émerge. Toute décision de comportement en conflit avec l'objectif collectif sera rejetée.

L'implémentation que nous menons dans le chapitre 6 montre l'efficacité du modèle dans différents cas d'usage. Nous retrouvons le même pattern de conception tant au niveau local qu'au niveau global. Ce qui confirme le caractère uniforme de notre modèle. Comme nous gérons deux niveaux différents avec le même modèle, nous confirmons également son caractère générique. Toutefois, le point essentiel qui se dégage de cette implémentation est la compatibilité du modèle lorsqu'on l'implémente à la fois au niveau local et au niveau global. Ceci vérifie notre hypothèse sur l'importance de la flexibilité individuelle pour atteindre l'objectif collectif. En effet, les éléments du code montrent que toutes les fonctionnalités sont distribuées au niveau du code de l'agent. Ces fonctionnalités incluent l'organisation et le contrôle. Par conséquent, nous concluons que le niveau local est un élément déterminant dans la performance d'un système distribué.

Avec ces trois contributions, nous proposons une solution pragmatique et légère. Par rapport à la carence d'adaptabilité dans les systèmes ambiants, nous apportons une brique pour plus de souplesse par rapport à l'imprévisibilité des changements du cadre opératoire.

Apports en méthodologie

La contribution de cette thèse se présente sur différents aspects : modèle, système et méthodologies. L'aspect modèle est au premier plan puisqu'il constitue la base de la contribution. En outre, la contribution présente également un aspect système avec les propositions d'implémentation vues dans le chapitre 6. Ce qui constitue la base du logiciel qui va supporter le modèle. Toutefois, l'ensemble de ces deux aspects rassemblés nous apporte une importante contribution en termes de méthodologies. Ces apports marquent l'originalité de notre contribution. Décrivons chacune de ces méthodologies dans ce qui suit.

Gestion de comportements

Notre contribution apporte deux méthodologies pour la gestion de comportements, d'autant plus que notre approche est orientée comportement.

La première consiste à séparer d'un côté les comportements et de l'autre côté les contraintes extérieures relatives à leur exécution. Cela favorise la réutilisation des comportements dans d'autres contextes. Pareillement, les règles relatives à ces contraintes extérieures sont réutilisables pour d'autres comportements exploitant le même environnement.

La deuxième méthodologie est une nouvelle solution pour l'*Action Selection Problem* (l'ASP). Par la prise en compte du contexte courant du cadre opératoire et l'ajustement du comportement qui s'en suit, nous apportons une nouvelle méthode de sélection d'actions qui s'ajoute aux méthodes classiques d'arbitrage et de fusion.

Nous décrivons ces deux méthodologies dans la section 3.5. Nous les montrons ensuite à travers le code de notre agent dans l'implémentation dans la section 6.3.

Prévention des erreurs

Notre contribution apporte une méthodologie pour la prévention des erreurs. La plupart des méthodes de prévention des erreurs consiste à anticiper lors de la conception les éventuelles situations possibles et de définir à l'avance les réactions du système en retour. Dans ce travail, nous partons du fait qu'une liste exhaustive des situations possibles ne peut pas toujours être réalisée 3.3.2. C'est pour cela qu'au lieu de les identifier, nous choisissons plutôt de nous focaliser sur la sureté de l'exécution et des interfaces d'action. Nous avons le contrôle sur ces éléments, ce qui n'est pas le cas pour les éventuelles situations possibles. Cela implique que nous pouvons identifier les conditions et règles garantissant cette sureté. Ce qui réduit considérablement les cas à prendre en compte. De cette manière, quelles que soient les perturbations du cadre applicatif, l'exécution demeure dans un état favorable. L'adaptation que nous obtenons est alors une prévention des situations néfastes et un ajustement de la future décision en conséquence.

Passage en multi-échelle

Notre contribution apporte une méthodologie pour monter en complexité. Notre modèle présente une uniformité pour le niveau local et le niveau global. Nous montrons leur cohésion dans la section 5.5.2 et nous l'illustrons sur la figure 5.5.2.

Grâce à cette configuration, le niveau local prévoit dans son code natif un module pour les contraintes sociales. En effet, la relation micro-macro que nous présentons sur la figure 5.2 a été définie depuis longtemps dans les SMA. Néanmoins, les modèles agents existants ne définissent pas explicitement la fonction de l'agent qui accueille ces contraintes sociales. Avec notre modèle, cette prise en considération est très explicite dès la conception de l'agent. De cette manière les deux modèles s'imbriquent parfaitement puisque l'agent prévoit déjà les contraintes sociales du niveau global.

Par ailleurs, le principal avantage de cette uniformité est la facilité pour monter en complexité. En effet, puisque le niveau local et le niveau global sont similaires, si nous changeons d'échelle, ce même niveau global peut devenir le niveau local d'un système plus complexe. Dans ce contexte il sera imbriqué dans un nouveau niveau global plus important. Nous pouvons ainsi avoir plusieurs niveaux d'échelle. Cela montre la puissance de notre contribution en termes de gestion de niveau de complexité.

Réutilisation de l'efficacité des SMA

Notre contribution apporte une méthodologie pour la réutilisation des méthodes et outils SMA pour l'environnement ambiant. Dans la section 2.1.3 nous justifions notre choix pour l'approche SMA. Une des particularités de la thèse est alors la réutilisation des méthodes de simulation SMA pour résoudre les problèmes des systèmes informatiques en environnement ambiant. A travers cette thèse, nous utilisons principalement : l'approche orientée comportement (section 2.3) caractérisée par la sélection d'actions, le Modèle-Influence-Réaction (IRM) internalisé au sein de l'agent (section 4.3), la boucle de rétroaction représentée par le suivi et l'ajustement (section 3.2.3), la distribution du contrôle (section 5.3.3), et l'interaction entre les agents (section 5.5.1). Nous ajoutons à ces méthodes la prise en compte des paramètres spécifiques du cadre applicatif ambiant. Cela peut être un ajout d'une interface d'écoute (pour la distribution du contrôle par exemple) ou la consultation de l'état courant de l'environnement dans la prise de décision (loi pour l'IRM par exemple). Par conséquent, nous affirmons que les SMA présentent des méthodes et outils pertinents et efficaces pour l'environnement ambiant. Avec cette thèse nous montrons les paramètres à considérer pour réutiliser les SMA dans un environnement non simulé.

Limites

Adaptation lente

Avec le modèle que nous proposons, le contrôle d'applicabilité (donc la régulation et les modalités opératoires) inhibe les décisions inopportunes des comportements. Les futures décisions seront ensuite ajustées en fonction du résultat précédent. Toutefois, ce résultat est juste une indication de la validation ou non de la décision. L'applicabilité n'indique pas les nouvelles décisions à prendre. C'est le comportement qui active une autre décision lorsque la précédente a été rejetée. Dans le meilleur des cas, la décision suivante est validée et le système suit son cours normal. Dans d'autres cas, le comportement va essayer un certain nombre de décisions avant d'avoir une validation. Ce cas de figure conduit donc à un ralentissement du système. La marge de temps pour trouver la bonne décision peut être significative. Dans le pire des cas, le comportement peut tourner en boucle sur toutes ses décisions possibles sans parvenir à la validation d'une d'entre elles. Dans ce contexte, la limite de notre approche est le fait qu'elle peut conduire à une situation de blocage du système. Cette situation de ralentissement ou de blocage se produit également lorsque les règles sont très restrictives. Nos perspectives proposent des solutions pour y remédier.

Adaptation réactive

Dans ce travail, nous proposons une adaptation réactive basée sur la perception de l'état du cadre applicatif. Dans ce contexte, le système évolue selon les objectifs définis initialement. Par conséquent, il est incapable de faire émerger de nouveaux objectifs. Changer d'objectif requiert un niveau cognitif pour évaluer les conditions de changement d'objectif. Le système tel qu'il est défini dans notre modèle ne supporte pas le niveau cognitif nécessaire. Ce qui fait qu'aucune évolution n'est possible. L'ensemble des actions possibles du système est connu au préalable. Ceci montre également une limite de notre proposition actuelle. Des perspectives sont proposées pour répondre à cette limitation.

Perspectives

Langage de manipulation des comportements

Nous avons choisi dans cette thèse une approche orientée comportement. Une des suites de ce travail de thèse, que nous avons déjà entamée dans [VPC], est la modélisation des comportements. En les analysant nous identifions des similarités avec les données dans une base de données [Cod90]. Chaque comportement possède différentes propriétés de la même manière qu'une donnée possède différents attributs. En incluant dans ses attributs l'agent qui le porte, chaque comportement est unique donc distinct des autres. Chaque comportement

peut être en relation avec d'autres. Les comportements forment un ensemble tout comme les données. Nous utilisons donc le modèle relationnel de données pour représenter l'ensemble des comportements existants. Cela facilite l'analyse des compétences disponibles, tout en contextualisant leurs possibilités de mobilisation. Le modèle relationnel représente les comportements avec des concepts simples (attributs, clé). Il les manipule avec des opérations simples comme la sélection et la jointure qui constituent un langage de requête.

Langage de formulation des règles

Dans notre proposition d'implémentation, les règles de régulation sont codées en Java par un code ad hoc. A ce niveau, nous pouvons améliorer la contribution en proposant un langage de formulation de règles d'applicabilité. Ici l'objectif est de fournir une méthodologie d'identification des contraintes, physiques ou fonctionnelles, conduisant à la définition des règles qui y sont associées. Par ailleurs, au niveau global, elle définit également comment les modalités opératoires sont injectées dans le niveau local. Dans GMAS, les règles sont classées par ordre d'importance : de la plus particulière à la plus générale. Le langage de formulation des règles doit alors tenir compte de cette contrainte pour hiérarchiser les règles. Pour mieux comprendre cette hiérarchie, considérant l'exemple d'un composant engagé simultanément dans différents espaces d'interaction ou dans différentes thématiques d'activités collectives. La hiérarchie des règles attendues est la priorisation d'un espace d'interaction ou d'une thématique à privilégier en cas de conflit.

Prévention des blocages

Par rapport aux problèmes de blocages précédents, nous pouvons proposer un mécanisme de prévention de l'arrêt total. Le but est d'offrir aux systèmes les moyens pour prendre des directions nouvelles quand les rejets de comportements deviennent trop répétitifs. Cela peut être par exemple, une règle de sortie de groupe lorsque les contraintes sociales empêchent le fonctionnement normal du composant. Ou bien, une règle sur un comportement de base par défaut, non bloquant, lorsqu'un certain nombre de boucles de rétroaction sans succès est atteint. Un tel mécanisme implique alors des agents plus cognitifs, capables de réaliser des fonctions spécifiques. Il s'agit de l'identification des espaces collectifs ou des règles collectives blocantes. Une fois qu'on les a identifiées, l'agent doit avoir la possibilité de sortir de ces sources de blocages.

La première est la détection des situations bloquantes. La deuxième est l'identification des règles sources du blocage. La troisième fonction revient à désactiver ces règles bloquantes. Finalement, la dernière fonction à entreprendre est le choix d'une nouvelle décision. Dans ce contexte, l'agent a besoin de plus d'informations pour enrichir la boucle de rétroaction sur laquelle s'appuie ces actions.

Adaptation cognitive

Nous avons vu précédemment les limites causées par l'adaptation réactive. Comme solution à ces limites, nous envisageons la conception d'un modèle de sur-système pour une adaptation cognitive. Cela implique forcément une capacité d'analyse et de raisonnement conséquente. Ici l'idée n'est pas de rehausser la capacité de calcul des composants. Nous avons vu que cela reste peu réalisable compte tenu de nos équipements. Le point essentiel de cette proposition réside dans la démultiplication de la couche matérielle. Le but est de faire en sorte que la totalité des calculs nécessaires puisse être distribuée sur cette couche physique. De cette manière, nous ne modifions pas la capacité de calcul de chacun des composants. Nous augmentons juste le nombre de ces équipements pour que le nombre des participants au calcul augmente. Ce qui réduit la charge de calcul revenant à chaque composant individuel. En résumé, la proposition attendue à ce niveau est de concevoir un modèle d'adaptation cognitive basé sur l'adaptation réactive. Le modèle est ensuite réparti sur les unités distribuées sans imposer aux composants une architecture informatique à forte capacité de calcul.

Bibliographie

[ABE+13] Stuart Anderson, Nicolas Bredeche, A.E Eiben, George Kampis, and Marteen Van Steen. *Adaptive Collective Systems Herding black sheep*. BookSprints for ICT research, 2013.

[AKB11] Christopher Armbrust, Lisa Kiekbusch, and Karsten Berns. Using behaviour activity sequences for motion generation and situation recognition. In *ICINCO 2011 - Proceedings of the 8th International Conference on Informatics in Control, Automation and Robotics, Volume 2, Noordwijkerhout, The Netherlands, 28 - 31 July, 2011*, pages 120–127, 2011.

[BCGP03] Carole Bernon, Valérie Camps, Marie-Pierre Gleizes, and Gauthier Picard. Tools for self-organizing applications engineering. In Giovanna Di Marzo Serugendo, Anthony Karageorgos, Omer F. Rana, and Franco Zambonelli, editors, *Engineering Self-Organising Systems, Nature-Inspired Approaches to Software Engineering [revised and extended papers presented at the Engineering Self-Organising Applications Workshop, ESOA 2003, held at AAMAS 2003 in Melbourne, Australia, in July 2003 and selected invited papers from leading researchers in self-organisation]*, volume 2977 of *Lecture Notes in Computer Science*, pages 283–298. Springer, 2003.

[BHGT12] Alberto Brunete, Miguel Hernando, Ernesto Gambao, and Jose Emilio Torres. A behaviour-based control architecture for heterogeneous modular, multi-configurable, chained micro-robots. *Robotics and Autonomous Systems*, 60(12) :1607–1624, 2012.

[Bro86] Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE J. Robotics and Automation*, 2(1) :14–23, 1986.

[BSA12] Laure Bourgois, Julien Saunier, and Jean-Michel Auberlet. Towards contextual goal-oriented perception for pedestrian simulation. In Joaquim Filipe and Ana L. N. Fred, editors, *ICAART 2012 - Proceedings of the 4th International Conference on Agents and Artificial Intelligence, Volume 2 - Agents, Vilamoura, Algarve, Portugal, 6-8 February, 2012*, pages 197–202. SciTePress, 2012.

[BSG+09] Yuriy Brun, Giovanna Di Marzo Serugendo, Cristina Gacek, Holger Giese, Holger Kienle, Marin Litoiu, Hausi Müller, Mauro Pezzè, and Mary Shaw. Engineering self-adaptive systems through feedback loops. In *Software engineering for self-adaptive systems*, pages 48–70. Springer, 2009.

[CGPM11] Federico Castanedo, Jesús García, Miguel A. Patricio, and José M. Molina. A multi-agent architecture based on the BDI model for data fusion in visual sensor networks. *Journal of Intelligent and Robotic Systems*, 62(3-4) :299–328, 2011.

[CH15] Giacomo Cabri and Emma Hart. Introduction to the special issue on collective adaptive systems. *Scalable Computing : Practice and Experience*, 16(3) :iii, 2015.

[Che11] Toly Chen. A self-adaptive agent-based fuzzy-neural scheduling system for a wafer fabrication factory. *Expert Syst. Appl.*, 38(6) :7158–7168, 2011.

[Cod90] E. F. Codd. *The Relational Model for Database Management, Version 2*. Addison-Wesley, 1990.

[CSM+15] Francesco Colace, Massimo De Santo, Vincenzo Moscato, Antonio Picariello, Fabio A. Schreiber, and Letizia Tanca. *Pervasive Systems Architecture and the Main Related Technologies*. Data-Centric Systems and Applications. Springer, 2015.

[DDF+06] Simon Dobson, Spyros G. Denazis, Antonio Fernández, Dominique Gaïti, Erol Gelenbe, Fabio Massacci, Paddy Nixon, Fabrice Saffre, Nikita Schmidt, and Franco Zambonelli. A survey of autonomic communications. *TAAS*, 1(2) :223–259, 2006.

[Fer09] Alois Ferscha. *Pervasive computing*. Springer, 2009.

[FGM03] Jacques Ferber, Olivier Gutknecht, and Fabien Michel. From agents to organizations : An organizational view of multi-agent systems. In Paolo Giorgini, Jörg P. Müller, and James Odell, editors, *Agent-Oriented Software Engineering IV, 4th International Workshop, AOSE 2003, Melbourne, Australia, July 15, 2003, Revised Papers*, volume 2935 of *Lecture Notes in Computer Science*, pages 214–230. Springer, 2003.

[FM96] Jacques Ferber and Jean-Pierre Müller. Influences and reaction : a model of situated multiagent systems. In *Proceedings of Second International Conference on Multi-Agent Systems (ICMAS-96)*, pages 72–79, 1996.

[GF00] Olivier Gutknecht and Jacques Ferber. The MADKIT agent platform architecture. In Thomas Wagner and Omer F. Rana, editors, *Infrastructure for Agents,*

Multi-Agent Systems, and Scalable Multi-Agent Systems, International Workshop on Infrastructure for Multi-Agent Systems, Barcelona, Spain, June 3-7, 2000, Revised Papers, volume 1887 of *Lecture Notes in Computer Science*, pages 48–55. Springer, 2000.

[GGR+14] Stéphane Galland, Nicolas Gaud, Sebastian Rodriguez, Flavien Balbo, Gauthier Picard, and Olivier Boissier. Contextualiser l'interaction entre agents en combinant dimensions sociale et physique au sein de l'environnement. In *Principe de Parcimonie - JFSMA 14 - Vingt-deuxièmes Journées Francophones sur les Systèmes Multi-Agents, Loriol-sur-Drôme, France, Octobre 8-10, 2014*, pages 65–74, 2014.

[GPC12] Yassine Gangat, Denis Payet, and Rémy Courdier. Another step toward reusability in agent-based simulation : Multi-behaviors & amvc. In *IEEE 24th International Conference on Tools with Artificial Intelligence, ICTAI 2012, Athens, Greece, November 7-9, 2012*, pages 1112–1119. IEEE Computer Society, 2012.

[GTC+10] Sebastian Gottifredi, Mariano Tucat, Daniel Corbata, Alejandro Javier García, and Guillermo Ricardo Simari. A BDI architecture for high level robot deliberation. volume 14, pages 74–83, 2010.

[Han06] David Hanon. *Modèle décisionnel orienté comportement fondé sur le vote : Application à la navigation d'agents autonomes en environnement simulé. (Behavior based decisional model using vote : Application to the autonomous navigation in simulated environments)*. PhD thesis, University of Valenciennes and Hainaut-Cambresis, France, 2006.

[Har08] Gail E. Harris, editor. *Decentralized control of automatic guided vehicles : applying multi-agent systems in practice*. ACM, 2008.

[Has03] Salima Hassas. *Systèmes complexes à base de Multi-Agents situés*. 2003.

[HG07] David Hanon and Emmanuelle Grislin-Le Strugeon. Sélection d'actions en environnement continu et dynamique par vote de comportements. In *Les Modèles de Comportements - JFSMA 07 - Quinzièmes Journées francophones sur les systèmes multi-agents, Carcassonne, France, October 17-19, 2007*, pages 223–232, 2007.

[HGM05] David Hanon, Emmanuelle Grislin-Le Strugeon, and René Mandiau. A behaviour based decisional model using vote. In *2005 International Conference on Computational Intelligence for Modelling Control and Automation (CIMCA) 2005,*

International Conference on Intelligent Agents, Web Technologies and Internet Commerce (IAWTIC) 2005, 28-30 November 2005, Vienna, Austria, pages 39–44. IEEE Computer Society, 2005.

[Jac95] Ferber Jacques. Les systèmes multi-agents, vers une intelligence collective. *InterEditions, Paris*, 1995.

[KC03] Jeffrey O. Kephart and David M. Chess. The vision of autonomic computing. *IEEE Computer*, 36(1) :41–50, 2003.

[KCS14] Nassim Kaldé, François Charpillet, and Olivier Simonin. Comparaison de stratégies d'exploration multi-robot classiques et interactives en environnement peuplé. In *Principe de Parcimonie - JFSMA 14 - Vingt-deuxièmes Journées Francophones sur les Systèmes Multi-Agents, Loriol-sur-Drôme, France, Octobre 8-10, 2014*, pages 75–84, 2014.

[KJM11] Venkateswara Rao Komma, Pramod K Jain, and Narinder K Mehta. An approach for agent modeling in manufacturing on jade™ reactive architecture. *The International Journal of Advanced Manufacturing Technology*, 52(9-12) :1079–1090, 2011.

[KMP11] Yoann Kubera, Philippe Mathieu, and Sébastien Picault. IODA : an interaction-oriented approach for multi-agent based simulations. *Autonomous Agents and Multi-Agent Systems*, 23(3) :303–343, 2011.

[KRV+15] Christian Krupitzer, Felix Maximilian Roth, Sebastian VanSyckel, Gregor Schiele, and Christian Becker. A survey on engineering approaches for self-adaptive systems. *Pervasive and Mobile Computing*, 17 :184–206, 2015.

[LLN13] Aleksis Liekna, Egons Lavendelis, and Agris Nikitenko. Challenges in development of real time multi-robot system using behaviour based agents. In Sigeru Omatu, José Neves, Juan M. Corchado Rodríguez, Juan F. De Paz Santana, and Sara Rodríguez-González, editors, *Distributed Computing and Artificial Intelligence - 10th International Conference, DCAI 2013, Salamanca, Spain, May 22-24, 2013*, volume 217 of *Advances in Intelligent Systems and Computing*, pages 587–595. Springer, 2013.

[MDSM12] Gildas Morvan, Daniel Dupont, Jean-Baptiste Soyez, and Rochdi Merzouki. Engineering hierarchical complex systems : An agent-based approach. the case of flexible manufacturing systems. In Theodor Borangiu, André Thomas, and Damien Trentesaux, editors, *Service Orientation in Holonic and Multi-Agent Manufacturing Control*, volume 402 of *Studies in Computational Intelligence*, pages 49–60. Springer, 2012.

[MF07] Saber Mansour and Jacques Ferber. Un modèle organisationnel pour les systèmes multi-agents ouverts déployés à grande échelle (présentation courte). In *Les Modèles de Comportements - JFSMA 07 - Quinzièmes Journées francophones sur les systèmes multi-agents, Carcassonne, France, October 17-19, 2007*, pages 107–116, 2007.

[MGF03] Fabien Michel, Abdelkader Gouaich, and Jacques Ferber. Weak interaction and strong interaction in agent based simulations. In David Hales, Bruce Edmonds, Emma Norling, and Juliette Rouchier, editors, *Multi-Agent-Based Simulation III, 4th International Workshop, MABS 2003, Melbourne, Australia, July 14th, 2003, Revised Papers*, volume 2927 of *Lecture Notes in Computer Science*, pages 43–56. Springer, 2003.

[MHdTH13] Frank D. Macías-Escrivá, Rodolfo E. Haber, Raúl M. del Toro, and Vicente Hernández. Self-adaptive systems : A survey of current approaches, research challenges and applications. *Expert Syst. Appl.*, 40(18) :7267–7279, 2013.

[Pay16] Denis Payet. Meca, 2016. http://meca.onover.top/.

[PCSR06] Denis Payet, Rémy Courdier, Nicolas Sébastien, and Tiana Ralambondrainy. Environment as support for simplification, reuse and integration of processes in spatial MAS. In *Proceedings of the 2006 IEEE International Conference on Information Reuse and Integration, IRI - 2006 : Heuristic Systems Engineering, September 16-18, 2006, Waikoloa, Hawaii, USA*, pages 127–131. IEEE Systems, Man, and Cybernetics Society, 2006.

[PDM+99] Enrico Pagello, Antonio D'Angelo, Federico Montesello, Francesco Garelli, and Carlo Ferrari. Cooperative behaviors in multi-robot systems through implicit communication. *Robotics and Autonomous Systems*, 29(1) :65–77, 1999.

[PHBG09] Gauthier Picard, Jomi Fred Hübner, Olivier Boissier, and Marie Pierre Gleizes. Réorganisation et auto-organisation dans les systèmes multi-agents [présentation courte]. In Zahia Guessoum and Salima Hassas, editors, *Systèmes Multi-Agents, Génie logiciel multi-agents - JFSMA 09 - Dix Septièmes Journées Francophones sur les Systèmes Multi-Agents, Lyon, France, October 19-21, 2009*, pages 89–98. Cepadues Editions, 2009.

[Pic14] Gauthier Picard. *Systèmes multi-agents adaptatifs : Ingénierie et utilisation dans le cadre de la résolution de problèmes. (Adaptive multi-agent systems)*. 2014.

[Pir99] Paolo Pirjanian. Behavior coordination mechanisms-state-of-the-art. Technical report, Institute for Robotics and Intelligent Systems, University of Southern California, 1999.

[Pri02] Christopher G. Prince. *Rodney A. Brooks, Cambrian Intelligence : The Early History of the New AI, Cambridge, MA : MIT Press, 1999, xii + 199 pp., $21.56 (paper), ISBN 0-262-52263-2*, volume 12. 2002.

[RA12] Lincoln S. Rocha and Rossana M. C. Andrade. Towards a formal model to reason about context-aware exception handling. In *Proceedings of the 5th International Workshop on Exception Handling, WEH 2012, Zurich, Switzerland, June 9, 2012*, pages 27–33. IEEE, 2012.

[RSF13] Eric Rohmer, Surya P. N. Singh, and Marc Freese. V-REP : A versatile and scalable robot simulation framework. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, Tokyo, Japan, November 3-7, 2013*, pages 1321–1326. IEEE, 2013.

[SF02] Olivier Sigaud and Fabien Flacher. Vers une approche dynamique de la sélection de l'action. *Approche dynamique de la cognition artificielle. Paris : Hermès*, 2002.

[SM09] European Commission Information Society and Media. Collective adaptive system. Technical report, Future and Emerging Technologies Proactive, 2009.

[ST09] Mazeiar Salehie and Ladan Tahvildari. Self-adaptive software : Landscape and research challenges. *TAAS*, 4(2) :14 :1–14 :42, 2009.

[TRGG99] Xavier Topin, Christine Regis, Marie-Pierre Gleizes, and Pierre Glize. Comportements individuels adaptatifs dans un environnement dynamique pour l'exploitation collective de ressources. *Intelligence Artificielle Située, cerveau, corps et environnement (IAS'99), Paris, France*, 25(10) :99–26, 1999.

[TTG12] Patrick Taillandier, Olivier Thérond, and Benoit Gaudou. Une architecture d'agents BDI basé sur la théorie des fonctions de croyance : application à la simulation du comportement des agriculteurs (présentation courte). In *Systèmes Multi-Agents : Ouverture, Autonomie, et Co-évolution - JFSMA 12 - Vingtièmes journées francophones sur les systèmes multi-agents, Honfleur, France, October 17-19, 2012*, pages 107–116, 2012.

[Vin14] Phan Cong Vinh. Self-adaptation in collective adaptive systems. *MONET*, 19(5) :626–633, 2014.

[VPC] Irène Velontrasina, Denis Payet, and Rémy Courdier. Modèle relationnel pour la coordination des comportements des agents , July. Poster.

[VPC16] Irene Velontrasina, Denis Payet, and Rémy Courdier. MECA : une architecture agent pour l'adaptation au niveau individuel par réutilisation du modèle influence/réaction (présentation courte). In Fabien Michel and Julien Saunier, editors, *Systèmes Multi-Agents et simulation - Vingt-quatrièmes journées francophones sur les systèmes multi-agents, JFSMA 16, Saint-Martin-du-Vivier (Rouen), France, Octobre 5-7, 2016.*, pages 161–170. Cépaduès Éditions, 2016.

[VPC19] Irene Velontrasina, Denis Payet, and Rémy Courdier. Regulation function for agent adaptation issues in ambient environment. In *Proceedings of the 11th International Conference on Computer Modeling and Simulation, ICCMS 2019. Melbourne, Australia, January 16 - 19, 2019*. ACM, 2019.

[WBH08] Danny Weyns, Nelis Boucké, and Tom Holvoet. A field-based versus a protocol-based approach for adaptive task assignment. *Autonomous Agents and Multi-Agent Systems*, 17(2) :288–319, 2008.

[Wey10] Danny Weyns. *Architecture-Based Design of Multi-Agent Systems*. Springer, 2010.

[WJ94] Michael Wooldridge and Nicholas R. Jennings. Agent theories, architectures, and languages : A survey. In Michael Wooldridge and Nicholas R. Jennings, editors, *Intelligent Agents, ECAI-94 Workshop on Agent Theories, Architectures, and Languages, Amsterdam, The Netherlands, August 8-9, 1994, Proceedings*, volume 890 of *Lecture Notes in Computer Science*, pages 1–39. Springer, 1994.

[WOO07] Danny Weyns, Andrea Omicini, and James Odell. Environment as a first class abstraction in multiagent systems. *Autonomous Agents and Multi-Agent Systems*, 14(1) :5–30, 2007.

[WU05] Yi-Chi Wang and John M. Usher. Application of reinforcement learning for agent-based production scheduling. *Eng. Appl. of AI*, 18(1) :73–82, 2005.

[ZOA+15] Franco Zambonelli, Andrea Omicini, Bernhard Anzengruber, Gabriella Castelli, Francesco L. De Angelis, Giovanna Di Marzo Serugendo, Simon A. Dobson, Jose Luis Fernandez-Marquez, Alois Ferscha, Marco Mamei, Stefano Mariani, Ambra Molesini, Sara Montagna, Jussi Nieminen, Danilo Pianini, Matteo Risoldi, Alberto Rosi, Graeme Stevenson, Mirko Viroli, and Juan Ye. Developing pervasive multi-agent systems with nature-inspired coordination. *Pervasive and Mobile Computing*, 17 :236–252, 2015.

UNIVERSITÉ DE LA RÉUNION

Pôle Recherche
Ecoles doctorales

LETTRE D'ENGAGEMENT DE NON-PLAGIAT

Je, soussigné(e) Irène VELONTRASINA en ma qualité de doctorant(e) de l'Université de La Réunion, déclare être conscient(e) que le plagiat est un acte délictueux passible de sanctions disciplinaires. Aussi, dans le respect de la propriété intellectuelle et du droit d'auteur, je m'engage à systématiquement citer mes sources, quelle qu'en soit la forme (textes, images, audiovisuel, internet), dans le cadre de la rédaction de ma thèse et de toute autre production scientifique, sachant que l'établissement est susceptible de soumettre le texte de ma thèse à un logiciel anti-plagiat.

Fait à Saint-Denis le : 26 mars 2019

Signature :

Extrait du Règlement intérieur de l'Université de La Réunion
(validé par le Conseil d'Administration en date du 11 décembre 2014)

Article 9. Protection de la propriété intellectuelle – Faux et usage de faux, contrefaçon, plagiat

L'utilisation des ressources informatiques de l'Université implique le respect de ses droits de propriété intellectuelle ainsi que ceux de ses partenaires et plus généralement, de tous tiers titulaires de tes droits.
En conséquence, chaque utilisateur doit :
- utiliser les logiciels dans les conditions de licences souscrites ;
- ne pas reproduire, copier, diffuser, modifier ou utiliser des logiciels, bases de données, pages Web, textes, images, photographies ou autres créations protégées par le droit d'auteur ou un droit privatif, sans avoir obtenu préalablement l'autorisation des titulaires de ces droits.

La contrefaçon et le faux
Conformément aux dispositions du code de la propriété intellectuelle, toute représentation ou reproduction intégrale ou partielle d'une œuvre de l'esprit faite ans le consentement de son auteur est illicite et constitue un délit pénal.
L'article 444-1 du code pénal dispose : « Constitue un faux toute altération frauduleuse de la vérité, de nature à cause un préjudice et accomplie par quelque moyen que ce soit, dans un écrit ou tout autre support d'expression de la pensée qui a pour objet ou qui peut avoir pour effet d'établir la preuve d'un droit ou d'un fait ayant des conséquences juridiques ».
L'article L335_3 du code de la propriété intellectuelle précise que : « Est également un délit de contrefaçon toute reproduction, représentation ou diffusion, par quelque moyen que ce soit, d'une œuvre de l'esprit en violation des droits de l'auteur, tels qu'ils sont définis et réglementés par la loi. Est également un délit de contrefaçon la violation de l'un des droits de l'auteur d'un logiciel (…) ».

Le plagiat est constitué par la copie, totale ou partielle d'un travail réalisé par autrui, lorsque la source empruntée n'est pas citée, quel que soit le moyen utilisé. Le plagiat constitue une violation du droit d'auteur (au sens des articles L 335-2 et L 335-3 du code de la propriété intellectuelle). Il peut être assimilé à un délit de contrefaçon. C'est aussi une faute disciplinaire, susceptible d'entraîner une sanction.
Les sources et les références utilisées dans le cadre des travaux (préparations, devoirs, mémoires, thèses, rapports de stage…) doivent être clairement citées. Des citations intégrales peuvent figurer dans les documents rendus, si elles sont assorties de leur référence (nom d'auteur, publication, date, éditeur…) et identifiées comme telles par des guillemets ou des italiques.

Les délits de contrefaçon, de plagiat et d'usage de faux peuvent donner lieu à une sanction disciplinaire indépendante de la mise en œuvre de poursuites pénales.