Apprentissage comportemental en continu de dispositifs ambiants par systèmes multi-agents adaptatifs

Valérian Guivarch, Valérie Camps, André Péninou, Pierre Glize

Institut de Recherche en Informatique de Toulouse
Université Paul Sabatier - Toulouse III, France
{prénom.nom}@irit.fr

RÉSUMÉ. *Nous proposons d'étendre la résolution émergente de problème au cadre des systèmes ambiants, au travers de la conception du système multi-agent adaptatif* Amadeus. *L'objectif d'*Amadeus *est d'apprendre le comportement correct à attribuer à un système ambiant. Cet apprentissage passe par l'observation des différentes actions des utilisateurs, afin d'être capable de progressivement réaliser ces actions à leur place. Pour être réellement adaptatif,* Amadeus *réalise un apprentissage en continu sans connaissance a priori ; il s'adapte à l'apparition/disparition de dispositifs et est capable de modifier son comportement en cours de fonctionnement en cas d'évolution du comportement utilisateur.* Amadeus *a été évalué par simulation sur plusieurs scénarios, et a montré sa capacité à apprendre localement le comportement correct à attribuer à un dispositif par l'observation des actions utilisateur.*

ABSTRACT. *We propose to extend the emergent problem solving to the ambient system field, through the design of the adaptive multi-agent system* Amadeus. *The goal of* Amadeus *is to learn the correct behavior to assign to an ambient system. This learning is based on the observation of users actions, in order to be able to progressively perform these actions on their behalf. To be truly adaptive,* Amadeus *performs a continuous learning without any a priori knowledge; it adapts itself to the appearance/disappearance of devices and it is able to modify its behaviour at runtime if there is a change in the users behavior. Amadeus has been evaluated by simulation through several scenarios, and has demonstrated its capability to learn in a local way the correct behavior to assign to a device by observing users actions.*

MOTS-CLÉS : *systèmes multi-agents adaptatifs, apprentissage, systèmes ambiants.*

KEYWORDS: *adaptive multi-Agent systems, learning, ambient systems.*

1. Introduction

Nos travaux s'inscrivent dans le domaine des systèmes ambiants et de l'informatique ubiquitaire. Contrairement à l'informatique classique, où un utilisateur interagit consciemment avec un unique dispositif, l'informatique ubiquitaire passe par l'utilisation d'un grand nombre de dispositifs hétérogènes. Ces dispositifs sont distribués dans l'environnement physique des utilisateurs et certains d'entre eux peuvent apparaître ou disparaître en cours de fonctionnement. Parmi les exemples d'applications, nous pouvons citer la domotique, avec les habitations s'adaptant aux préférences de ses habitants (Goumopoulos *et al*., 2008), ou encore le tourisme (FitzGerald *et al*., 2013).

Etant donnée la complexité de tels systèmes, il devient de plus en plus difficile, voire impossible, d'énumérer dès la phase de spécification la totalité des cas d'utilisation auxquels ils peuvent être confrontés, et par conséquent de déterminer le comportement correct à attribuer à ces systèmes pour la totalité des situations qu'ils sont susceptibles de rencontrer. Pour répondre à ce besoin, une des solutions possibles est de rendre ces systèmes capables de s'adapter au contexte de leurs utilisateurs ; nous parlons alors de systèmes *context aware* (Coutaz *et al*., 2005). Cette sensibilité au contexte doit prendre en compte la dynamique des systèmes ambiants ; il s'agit alors d'être capable de s'adapter (*i*) à l'ouverture du système, en étant capable de gérer l'apparition de nouveaux dispositifs et la disparition de dispositifs présents, et (*ii*) aux changements de comportements des utilisateurs.

L'objectif de ce papier est de proposer un aperçu général du système multi-agent adaptatif *Amadeus* destiné à apprendre le comportement à attribuer à un système ambiant. Après avoir dressé un rapide état de l'art du domaine en section 2, les principes généraux de fonctionnement d'*Amadeus* sont présentés en section 3. Les sections 4, 5 et 6 montrent respectivement ses capacités d'apprentissage et d'ouverture, ses capacités d'adaptation à l'évolution des comportements utilisateurs et la distribution de ses données. Chaque section est complétée par une évaluation d'*Amadeus* réalisée à l'aide d'un simulateur. La section 7 conclut par quelques perspectives.

2. État de l'art

2.1. Systèmes sensibles au contexte

Inspirés notamment des définitions fournies par (Dey, Abowd, 2000), (Strassner *et al*., 2009) et (Brézillon, 2010), nous avons défini le contexte comme "l'ensemble des informations extérieures à l'activité d'une entité, décrivant l'environnement tel qu'elle le perçoit et ayant un impact sur l'exécution d'une ou plusieurs tâches liées à son activité", et la sensibilité au contexte comme "la capacité à percevoir, interpréter et utiliser les différentes informations relatives au contexte courant pour adapter dynamiquement sa fonctionnalité" (Korkea-Aho, 2000).

Pour notre étude, nous nous sommes intéressés aux travaux aptes à rendre un système ambiant sensible au contexte. L'objectif de tels systèmes est d'adapter dynamiquement sa fonctionnalité selon son contexte courant. Nous avons alors étudié les propriétés requises pour de tels systèmes, puis étudié les solutions (partielles ou complètes) proposées par la littérature.

2.1.1. Propriétés requises pour la sensibilisation au contexte en système ambiant

Pour permettre à un système ambiant d'être sensible au contexte, certaines propriétés doivent être respectées. Plusieurs d'entre elles ont été proposées par (Dey *et al.*, 2001),(Hofer *et al.*, 2003),(Euzenat *et al.*, 2008), (Reignier, 2010). Nous présentons ici celles qui nous ont semblé les plus pertinentes :

– **Généricité** : un système est dit générique si sa représentation des données contextuelles lui permet d'intégrer n'importe quel type de données ;

– **Expressivité** : une donnée contextuelle ne se limite pas à une valeur, elle est aussi associée à une sémantique plus ou moins riche ;

– **Distribution** : un système distribué est physiquement et/ou logiquement décomposé en différents processus fonctionnant sans requérir la présence d'un processus central (un serveur, par exemple), et limitant leurs échanges aux données nécessaires et suffisantes à leur fonctionnement ;

– **Ouverture** : un système est ouvert s'il peut intégrer et exploiter de nouvelles sources de données en cours de fonctionnement sans pour autant nécessiter d'être (re)configuré et/ou de recommencer la totalité de ses traitements ;

– **Proactivité** : la proactivité d'un système désigne sa capacité à anticiper et à prendre des décisions concernant une situation, et à agir en conséquence ;

– **Confidentialité** : la confidentialité permet de garantir que les données contextuelles ne sont pas divulguées à des tiers ;

– **Explicabilité** : l'explicabilité d'un système concerne sa capacité à donner, en continu ou en cas de demande d'un utilisateur, les faits et le raisonnement associé qui expliquent son comportement.

Pour évaluer les différents systèmes de sensibilisation au contexte décrits dans la section suivante, nous nous basons sur des critères issus de ces propriétés. Chaque système est alors évalué selon qu'il respecte parfaitement (++), significativement (+), légèrement (-) ou pas du tout (- -) chacun des critères.

2.1.2. Systèmes existants pour la sensibilisation au contexte

La littérature propose un certain nombre de systèmes permettant de rendre un environnement ambiant sensible au contexte. Parmi eux, certains appartiennent à la catégorie des gestionnaires de contexte. Un gestionnaire de contexte est défini par (Rottenberg *et al.*, 2012) comme une "entité logicielle responsable de la collecte, de la gestion (traitement et filtrage) et de la présentation des informations de contexte aux applications". Il n'a donc pas pour objectif de répondre aux sept critères précédem-

ment énoncés, son fonctionnement se limitant aux aspects récupération et présentation des données contextuelles plutôt qu'aux aspects utilisation de ces données.

Le *Context Toolkit* de (Dey *et al*., 2001) est une des premières architectures de gestion du contexte proposée dans la littérature. Sa modélisation des informations contextuelles grâce à un simple modèle du type <*key-value*> lui permet d'être générique, et il dispose de mécanismes pour gérer la propriété d'ouverture, mais requiert un serveur central pour fonctionner. Le gestionnaire de contexte *Hydrogen*, proposé par (Hofer *et al*., 2003), possède des caractéristiques similaires (notamment en matière de généricité), mais favorise l'aspect distribution des données et des traitements. Le système *CoBrA* (COntext BRoker Architecture) proposé par (Chen *et al*., 2005), en revanche, favorise l'expressivité des données en les modélisant à l'aide d'une ontologie, mais nécessite aussi la présence d'un composant central pour fonctionner. Enfin, le gestionnaire de contexte SPACES de (Romero *et al*., 2010) assure une gestion des données ouverte, complètement distribuée, et utilisant une modélisation des données basée sur la classification des types média MIME, ce qui limite sensiblement la généricité des données mais leur donne une certaine expressivité.

D'autres systèmes ne se limitent pas à l'agrégation et à la présentation des données, mais sont aussi capables de réaliser des actions sur le système ambiant de façon proactive en se basant sur un comportement fourni explicitement par le concepteur. L'*Intelligent Classroom* (Franklin, Flaschbart, 1998) est un système de gestion d'une classe de cours capable d'adapter le fonctionnement des effecteurs de la classe (lampes, grand écran, etc.) en fonction de l'activité de l'enseignant. L'architecture SIM (Baek *et al*., 2007) est un framework pour la gestion et l'exploitation des données issues de capteurs en système ambiant qui inclut un agent dit "intelligent" dont l'objectif est de modifier l'état des différents effecteurs de l'environnement en fonction des données perçues. Le contexteur (Rey, 2005) est une abstraction logicielle permettant de décomposer les processus de traitements des données contextuelles depuis les capteurs jusqu'aux applications consommatrices. Enfin, le système ASK-IT (Spanoudakis, Moraitis, 2006) permet aux utilisateurs d'accéder facilement aux données et aux services qui les entourent, en adaptant notamment les services proposés en fonction des éventuels handicaps de ses utilisateurs.

Cependant, nous considérons qu'un système est réellement proactif s'il est capable d'enrichir son comportement par un processus d'apprentissage. C'est le cas du système ACHE (Mozer, Miller, 1998) qui utilise un apprentissage basé sur le Q-Learning. Le système iDorm (Hagras *et al*., 2004) et le système du projet ATRACO (Minker *et al*., 2010) appliquent un algorithme d'apprentissage basé sur la logique floue. Le système de (Tapia *et al*., 2008) utilise un mécanisme basé sur le *Case-Based Reasoning*. Le système de (Zaidenberg, 2009) utilise un mécanisme d'apprentissage par renforcement indirect. Cependant, la majorité de ces systèmes présentent des limites en matière de généricité, d'ouverture et de distribution. Même des systèmes tels que celui de (Tapia *et al*., 2008), qui propose une architecture multi-agent distribuée, attribue généralement l'apprentissage à un unique élément central.

Tableau 1. Respect des critères définis en section 2.1.1 par les différents systèmes étudiés sensibles au contexte

Auteurs	Système	Année	Généricité	Expressivité	Distribution	Ouverture	Proactivité	Confidentialité	Explicabilité
Franklin	Intelligent Classroom	1998	-	-	--	--	+	--	-
Mozer	ACHE	1998	--	+	--	-	++	--	-
Dey	Context Toolkit	2001	++	--	-	++	--	++	--
Hofer	Hydrogen	2003	+	-	+	++	--	-	--
Hagras	iDorm	2004	--	-	-	--	++	--	-
Chen	CoBrA	2005	--	++	--	++	--	+	-
Rey	Contexteur	2005	-	+	++	++	+	--	--
Spanoudakis	ASK-IT	2006	--	++	-	++	+	--	--
Baek	SIM	2007	-	+	-	--	+	--	--
Tapia	...	2008	-	-	-	--	++	--	-
Conan	SPACES	2008	-	+	++	++	+	--	--
Zaidenberg	...	2009	-	-	-	+	++	--	++
Minker	ATRACO	2010	-	++	-	++	++	++	+

L'étude de ces différents systèmes par rapport à notre grille d'analyse (tableau 1) a permis de mettre en évidence un certain nombre de constats. En particulier, nous pouvons remarquer que les systèmes capables d'enrichir leur comportement par un processus d'apprentissage sont fortement pénalisés en matière de distribution et d'ouverture, ce qui semble les rendre difficilement capables de réagir à la réelle dynamique des systèmes ambiants. Partant de ce constat, nous nous sommes intéressés aux techniques classiques d'apprentissage pour tenter de comprendre ces limites.

2.2. Apprentissage et adaptation à la dynamique environnementale

2.2.1. Techniques d'apprentissage

Un algorithme d'apprentissage est défini par (Cornuéjols, Miclet, 2011) comme un algorithme dont "*l'objectif est de construire un modèle de la réalité à partir de données*". Parmi ces algorithmes, nous trouvons les techniques d'apprentissage supervisé et les techniques d'apprentissage par renforcement.

L'apprentissage supervisé (Kotsiantis, 2007) cherche à construire un modèle à partir d'un ensemble de jeux de données, chacun d'eux étant associé à une certaine sortie. Le modèle construit permet alors d'attribuer la sortie correcte à tout nouveau jeu de données. Le plus souvent, les algorithmes fonctionnent en mode *hors ligne*. L'apprentissage se fait alors en deux phases : l'algorithme (*i*) utilise un ensemble de jeux de données pour construire son modèle, puis (*ii*) utilise ce modèle sans le modifier. De tels algorithmes sont intéressants pour notre problème car ils se basent sur des observations du passé afin de déterminer la sortie correcte ; cette propriété les rend adaptés à un apprentissage visant à faire une imitation d'un comportement. En revanche, ils présentent des limites en cas d'évolution de la fonctionnalité à apprendre ou en cas de changement des dispositifs présents (*i.e.* des données en entrée de l'algorithme) car cela nécessite de recommencer l'apprentissage pour les intégrer.

L'apprentissage par renforcement (Sutton, Barto, 1998) fonctionne au travers de cycles essai/erreur. Il propose une action dans une situation donnée, puis observe le *feedback* explicite reçu en retour pour établir si son action était correcte ou pas, et construit ainsi progressivement son modèle. L'utilisation de tels algorithmes n'implique donc pas de séparation entre la phase d'apprentissage et d'utilisation. En revanche, ils requièrent un *feedback* explicite de leur environnement pour réaliser leur apprentissage. Leur fonctionnement entraîne la réalisation d'actions erronées, le temps qu'un modèle correct soit construit, ce qui, dans le cadre d'un système ambiant, peut engendrer une forte gêne pour les utilisateurs.

Notre objectif est de disposer d'une solution proposant les avantages de ces deux approches, à savoir se baser sur des observations sans séparation entre la phase d'apprentissage et la phase d'utilisation. Pour obtenir ce résultat, nous nous sommes intéressés à la résolution émergente de problèmes.

2.2.2. Résolution émergente de problèmes

L'approche par AMAS (*Adaptive Multi-Agent System)* développée dans l'équipe SMAC [1] permet la conception d'un système multi-agent adaptatif pour résoudre des problèmes complexes nécessitant de l'adaptation, cette adaptation étant alors obtenue par auto-organisation. Cette approche est préconisée pour résoudre des problèmes pour lesquels il n'existe pas *a priori* d'algorithme classique capable d'y répondre, pour lesquels la fonctionnalité recherchée n'est qu'incomplètement spécifiée, et pour lesquels l'environnement du système à concevoir est dynamique, inaccessible et continu (Russell, Norvig, 1995). Ces caractéristiques la rendent particulièrement intéressante pour notre problème. La conception d'un AMAS suit une approche *bottom up* ; le concepteur commence par définir les fonctions locales des différents agents composant le système (chaque agent possédant son propre objectif et ses propres connaissances), puis les interactions entre les différents agents de sorte à collectivement produire une fonctionnalité collective. Selon cette approche, pour s'assurer que cette fonctionnalité soit adéquate (du point de vue d'un observateur extérieur au système connaissant sa finalité), chaque agent doit entretenir des activités coopératives de son point de vue local, avec les agents qu'il connaît et l'environnement du système (Georgé *et al*., 2011).

Une définition générale de la coopération (Picard, 2004) la caractérise comme le juste milieu entre l'altruisme et l'égoïsme. Un agent coopératif cherche donc à atteindre aussi bien ses objectifs qu'à aider les autres agents à atteindre les leurs. L'approche par AMAS considère qu'un agent est en situation coopérative dès lors qu'il respecte les trois métarègles suivantes (à instancier en fonction du problème à résoudre) : (i) tout signal perçu par l'agent est compris par lui sans ambiguïté, (ii) toute information provenant de ses perceptions est utile à son raisonnement, et (iii) son raisonnement lui permet d'effectuer des actions utiles aux autres et à l'environnement.

1. http://www.irit.fr/-Equipe-SMAC

Si une de ces métarègles n'est pas respectée, l'agent est face à une situation est appelée "Situation Non Coopérative" (SNC). Sept SNC génériques ont été mises en évidence à partir des 3 métarègles précédentes : (i) *Incompréhension* : l'agent perçoit un signal qu'il ne comprend pas ; (ii) *Ambiguïté* : l'agent attribue plusieurs interprétations possibles à un signal perçu ; (iii) *Incompétence* : l'agent n'est pas capable d'exploiter le signal perçu lors de son raisonnement ; (iv) *Improductivité* : l'agent ne peut tirer aucune conclusion à partir du signal perçu ; (v) *Concurrence* : l'agent croit que son action et celle d'un autre agent vont aboutir au même résultat ; (vi) *Conflit* : l'agent croit que la réalisation de son action est incompatible et avec celle d'un autre agent (elle empêche cet agent de réaliser sa propre action) ; et enfin (vii) *Inutilité* : l'agent croit que son action ne va ni le rapprocher de son but, ni aider un autre agent.

(Capera, 2005) définit le comportement d'un agent coopératif comme étant constitué de deux comportements distincts : (i) un comportement *nominal*, décrivant sa fonctionnalité au sein du système et les actions qu'il réalise pour atteindre ses objectifs indépendamment des SNC qu'il est susceptible de rencontrer, et (ii) un comportement *coopératif* qui lui permet de prévenir, détecter et résoudre les SNC qu'il est susceptible de rencontrer. Le comportement coopératif des agents est à l'origine de l'auto-adaptation du système : au fur et à mesure que les différentes SNC sont résolues ou évitées, le système tend vers l'adéquation fonctionnelle (sa fonctionnalité est considérée satisfaisante pour un observateur extérieur).

Lorsqu'un agent se retrouve face à une SNC, il dispose de trois mécanismes pour revenir dans un état coopératif, et tendre ainsi vers l'adéquation fonctionnelle : (i) l'ajustement (*tuning*) : il va modifier uniquement les paramètres de sa fonction nominale ; (ii) la réorganisation : il va modifier ses liens avec les agents de son voisinage ; (iii) l'évolution : il crée d'autres agents ou bien se supprime lui-même.

Pour concevoir *Amadeus*, nous avons utilisé la méthode ADELFE (Bonjean *et al.*, 2013) qui fait office de guide pour la conception d'AMAS.

3. Principes d'*Amadeus*

L'objectif du système multi-agent adaptatif *Amadeus* est d'apprendre un comportement correct à attribuer à un système ambiant de sorte à satisfaire les utilisateurs en considérant les deux hypothèses suivantes : (*i*) le comportement à apprendre concerne les activités récurrentes (observées plusieurs fois) des utilisateurs ; nous ne nous intéressons pas aux cas particuliers ou à risque, impliquant leur sécurité ; (*ii*) le système ne peut réaliser une action à la place d'un utilisateur que si cette action a été préalablement observée (contrairement par exemple à ce que ferait un algorithme d'apprentissage par renforcement). Le domaine des systèmes ambiants impose aussi un certain nombre de contraintes comme la nécessité de fonctionner avec un grand nombre de données perceptibles qui peuvent se rajouter ou se supprimer en cours de fonctionnement. Pour répondre à ces contraintes, nous proposons d'étendre la résolution émergente par AMAS au domaine de l'apprentissage. Il s'agit alors de concevoir une technique d'apprentissage :

– par observation *en ligne* des activités utilisateur : *Amadeus* enrichit ses connaissances sur les actions qu'il peut réaliser au fur et à mesure que les utilisateurs agissent dans leur environnement, et il réalise progressivement ces actions à leur place au fur et à mesure de son apprentissage ;

– ouverte et distribuée : le comportement à attribuer à chaque dispositif est appris de façon locale à chaque dispositif car le nombre de dispositifs peut évoluer en cours de fonctionnement ;

– sans connaissance *a priori* : l'apprentissage d'*Amadeus* se fait sans profil utilisateur ni présupposé sémantique. Les données perçues sont alors caractérisées uniquement par leur valeur numérique associée à un identifiant unique, sans aucune information supplémentaire.

3.1. Présentation générale

L'application de la méthode ADELFE à notre problème nous a permis d'aboutir à une architecture composée de deux niveaux (figure 1) :

– le niveau inférieur est local à chaque dispositif. Il s'agit d'un AMAS *dispositif* dont le but est d'apprendre le comportement à attribuer à son dispositif. Il se compose de 5 types d'agents :

a) Un agent *capteur* qui est en charge de la propagation pertinente des données en provenance de son capteur et à destination des différents AMAS *dispositif* (voir section 6.1) ;

b) Un agent *effecteur* qui réalise le même traitement que l'agent *capteur* (propager les données représentant l'état de l'effecteur) mais qui est aussi capable de modifier l'état de l'effecteur qui lui est associé (voir section 6.1) ;

c) Un agent *donnée* qui est en charge de l'évaluation locale de l'utilité de la donnée dont il est responsable pour les agents *contexte* de son AMAS *dispositif* (voir section 6.2) ;

d) Un agent *contrôleur* qui décide quelle action réaliser sur son effecteur en fonction du contexte courant (voir section 4.2) ;

e) Un agent *contexte* qui propose une action particulière à réaliser dans un contexte spécifique (voir section 4.1).

– le niveau supérieur est constitué de l'ensemble des AMAS *dispositif* qui forment alors *Amadeus*, un AMAS d'AMAS *dispositif* capable d'apprendre un comportement pour le système ambiant. Au sein de cet AMAS, les différents AMAS *dispositif* coopèrent entre eux pour s'échanger les données pertinentes à leur fonctionnement respectif (Guivarch, 2014).

L'initialisation d'un AMAS *dispositif* induit la création d'agents *capteur*/*effecteur* pour chaque capteur/effecteur dans le dispositif. Cette création est réalisée explicitement par le concepteur. La création d'un agent *effecteur* entraîne automatiquement la création de l'agent *contrôleur* en charge du contrôle de cet effecteur. Par la suite,

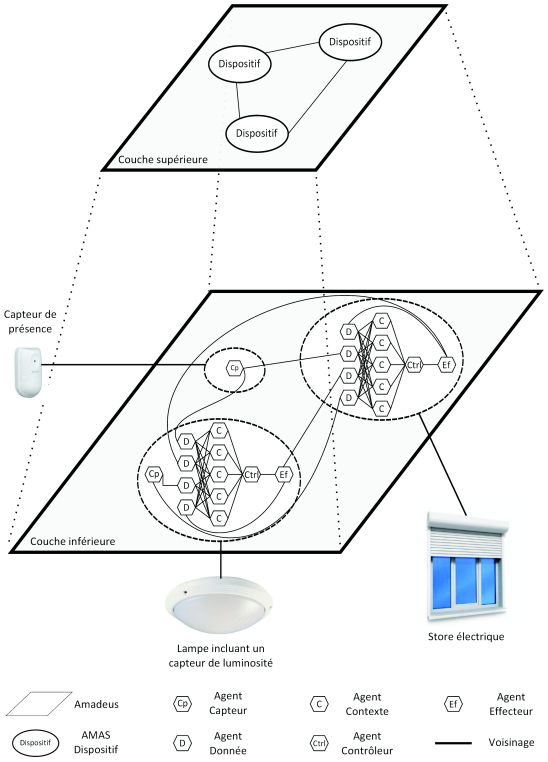

*Figure 1. Architecture générale d'*Amadeus

la création des agents *donnée* et *contexte* se fait en cours de fonctionnement, comme décrit en sections 6.1 et 4.2.3.

3.2. Exemple d'adaptation selon notre approche

Pour faciliter la compréhension des objectifs et du fonctionnement des différents agents, nous présentons le fonctionnement global simplifié d'un AMAS *dispositif*, de la mise à jour d'une valeur perçue depuis un capteur, jusqu'à la décision de l'action à réaliser. Ce fonctionnement est illustré (figure 2) à l'aide d'un exemple du contrôle d'un radiateur (par un AMAS *dispositif* associé à ce radiateur) :

1. L'agent *capteur* perçoit un changement de valeur depuis le capteur physique (exemple : capteur de température) ;
2. Cet agent *capteur* envoie la nouvelle donnée à l'agent *donnée* qui lui est associé (voir section 6.1) ;
3. L'agent *donnée* envoie sa nouvelle valeur aux agents *contexte* locaux (quatre dans cet exemple) pour qui cette donnée est utile (voir section 6.2) ;

4. a) Un agent *contexte*, devenu valide grâce à la mise à jour, propose de réaliser une action (par exemple, il propose de mettre le radiateur à 1) ;
   b) Un autre agent *contexte*, devenu lui aussi valide, propose de réaliser une autre action (par exemple, il propose de mettre le radiateur à 2) (voir section 4.1) ;
5. a) L'agent *contrôleur* choisit un agent *contexte* parmi les 2 valides en fonction des deux paramètres que sont la confiance et le gain estimé. Le premier agent *contexte* est supposé ici sélectionné ;
   b) L'agent *contrôleur* déselectionne l'agent *contexte* sélectionné précédent (qui proposait par exemple de maintenir le radiateur à 0) (voir section 4.2) ;
6. L'agent *contrôleur* envoie la proposition d'action de l'agent sélectionné à l'agent *effecteur* (associé au radiateur dans notre exemple) ;
7. L'agent *effecteur* réalise l'action demandée par l'agent *contrôleur* (dans l'exemple, mettre le radiateur à 1).

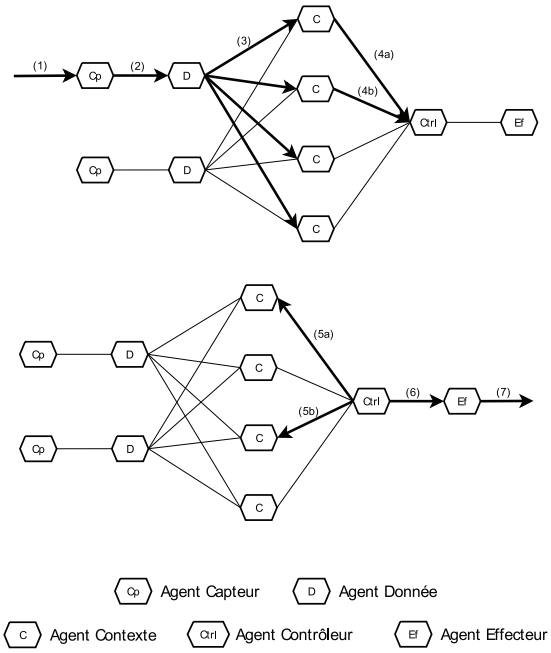

*Figure 2. Exemple de fonctionnement d'*Amadeus *sur un cas simple*

Les trois prochaines sections présentent les trois grandes propriétés d'*Amadeus*, à savoir (*i*) sa capacité d'apprentissage et d'ouverture, (*ii*) sa capacité à faire évoluer la fonctionnalité qu'il a préalablement apprise, et enfin (*iii*) sa capacité à distribuer de façon efficace les données entre AMAS *dispositifs*.

4. Apprentissage et ouverture dans *Amadeus*

Nous présentons ici le fonctionnement des agents *contexte* et *contrôleur* qui permettent à un AMAS *dispositif* d'apprendre le comportement à attribuer à un dispositif.

4.1. Agent contexte

Le processus d'apprentissage réalisé par un AMAS *dispositif* résulte principalement du fonctionnement des agents *contexte*. L'objectif d'un agent *contexte* est de proposer une action sur un effecteur présent sur son dispositif dans une situation particulière, chaque agent *contexte* étant associé à un unique agent *contrôleur*. Cet agent représente une connaissance sur une action passée d'un utilisateur dans une situation donnée, cette connaissance permettant à *Amadeus* d'imiter le comportement observé de cet utilisateur en réalisant cette même action si une situation similaire se présente.

La création d'un agent *contexte* fait suite au mécanisme d'évolution (section 2.2.2) survenant dans deux cas (voir section 4.2.3) : (*i*) lorsqu'une action est réalisée par un utilisateur, l'agent *contexte* propose alors la même modification d'état de l'effecteur que celle réalisée par l'utilisateur ; (*ii*) lorsqu'aucune action n'est réalisée, l'agent *contexte* propose alors de maintenir l'état de l'effecteur dans son état courant. Un agent *contexte* possède les représentations du monde suivantes :

– une représentation de l'action à réaliser sur l'effecteur (l'état à lui attribuer) ;

– une représentation de la situation dans laquelle la réalisation de son action semble pertinente, appelée situation de validité ;

– un niveau de confiance qui représente la confiance qu'il accorde à sa proposition d'action lorsqu'il l'envoie à l'agent *contrôleur* (voir section 5.2.1) ;

– un gain estimé, qui est une valeur numérique représentant une croyance sur l'intérêt de réaliser son action plutôt que celle d'un autre agent (voir section 5.2.3) ;

Pour représenter sa situation de validité, un agent *contexte* observe, à sa création, les valeurs courantes de chaque donnée perceptible. Il crée alors autour de la valeur de chaque donnée perçue d une *plage de validité* Ξ_d sous la forme d'un AVRT (voir section 5.2.2). Chaque plage de validité est une plage de valeurs qui possède deux états : *valide* si la valeur courante de la donnée appartient à la plage de valeurs, *invalide* sinon. Un agent *contexte* est lui-même *valide* si toutes ses plages de validité sont valides ; la situation courante est alors similaire à celle décrite par l'agent *contexte*, c'est pourquoi il est pertinent pour celui-ci de proposer son action. Il envoie alors sa proposition d'action à l'agent *contrôleur* qui lui est associé. En revanche, si au moins une de ses plages de validité est invalide, l'agent *contexte* est lui-même *invalide* : la situation courante n'est pas pertinente pour qu'il propose son action. Un agent *contexte* valide est alors susceptible d'être sélectionné par son agent *contrôleur*.

L'ouverture est mise en oeuvre dans *Amadeus* (*i*) lorsqu'une nouvelle donnée est perçue par un agent *contexte*. Il crée alors une nouvelle plage de validité pour cette donnée, en initialisant les bornes de cette plage de sorte à ce qu'elle recouvre l'en-

semble des valeurs que peut prendre cette nouvelle donnée ; (*ii*) lorsqu'une des données perçues par un agent *contexte* disparaît, celui-ci arrête d'en tenir compte pour déterminer s'il est valide ou pas.

La proposition d'action d'un agent *contexte* se compose de trois éléments : (*i*) l'action elle-même, autrement dit l'état à affecter à un effecteur, (*ii*) le gain estimé η qu'il attribue à la réalisation de cette action, et (*ii*) le niveau de confiance Cf qu'il accorde à sa proposition. Les différents ajustements qui mènent un agent *contexte* à modifier la valeur de son gain estimé et de sa confiance sont décrits en section 5.

4.2. Agent contrôleur

Un agent *contrôleur* est associé à chaque effecteur présent sur un dispositif. Son objectif est de choisir l'action à réaliser sur cet effecteur dans chaque situation, en fonction des propositions d'actions envoyées par ses agents *contexte*.

Le choix de la meilleure proposition d'action envoyée par les agents *contexte* se fait par résolution des différents SNC (voir section 2.2.2) qui apparaissent entre ces propositions (Guivarch *et al.*, 2012). Nous donnons ici deux exemples de SNC susceptibles d'apparaître, ainsi que le processus de résolution associé. Le comportement nominal d'un agent *contrôleur* consiste, à chaque cycle, à résoudre toutes les SNC entre ces différentes propositions d'actions jusqu'à choisir la meilleure, et sélectionner l'agent *contexte* qui la propose.

4.2.1. Deux agents contexte *proposent deux actions différentes*

Deux agents *contexte* C_1 et C_2 peuvent se retrouver simultanément valides alors qu'ils proposent deux actions A_1 et A_2 différentes. Ces deux agents *contexte* sont alors face à une SNC de conflit : tous les deux pensent contribuer à la même situation (satisfaire les utilisateurs) au travers d'actions différentes.

Cette SNC est détectée par l'agent *contrôleur* associé aux agents *contexte* en conflit, lorsqu'il reçoit leurs propositions. Sa résolution se fait par observation des gains estimés η_1 et η_2 associés aux propositions d'actions de C_1 et C_2 (voir section 5.2.3). L'agent *contrôleur* sélectionne l'agent *contexte* proposant le gain estimé le plus élevé.

4.2.2. Deux agents contexte *proposent deux actions identiques*

Un agent *contrôleur* peut recevoir deux propositions d'actions identiques issues de deux agents *contexte* C_1 et C_2. En effet, le fait que les agents *contexte* tentent de généraliser leurs connaissances en élargissant leurs plages de validités, entraîne des situations où leurs plages de valeurs ont des intersections non nulles. Cette situation est perçue comme une SNC de concurrence si ces deux agents accompagnent leurs propositions d'actions d'une valeur de gain estimé différente. Ces deux agents se retrouvent alors face à une SNC de concurrence.

La détection de cette SNC est réalisée par l'agent *contrôleur* associé aux agents *contexte* en concurrence, lorsqu'il reçoit leurs propositions. Pour résoudre cette SNC,

nous avons intégré aux agents *contexte* la notion de confiance. Plus un agent *contexte* possède une confiance élevée, plus il croit que le gain estimé qu'il associe à sa proposition d'action est représentative de la pertinence à réaliser l'action qu'il propose à l'agent *contrôleur* (voir section 5.2.1). L'agent *contrôleur* sélectionne alors l'agent *contexte* dont le niveau de confiance est le plus élevé.

4.2.3. Aucun agent contexte *valide correct*

L'objectif de l'agent *contrôleur* est de réaliser des actions à la place des utilisateurs. Si un des utilisateurs réalise une action alors que cette action ne faisait pas partie des propositions d'actions déjà perçues par l'agent *contrôleur*, l'agent *contrôleur* est face à une SNC d'incompétence : il n'est pas capable de réaliser les actions à la place des utilisateurs au vu des propositions d'actions qu'il perçoit. La résolution de cette SNC passe par la création d'un nouvel agent *contexte* par l'agent *contrôleur*, qui lui fournit la description de la situation dans laquelle l'utilisateur a réalisé son action (autrement dit, la liste de l'état de chaque donnée perçu par l'AMAS *dispositif*), ainsi que la description de l'action elle-même. Les différents éléments composant ce nouvel agent *contexte* sont initialisés comme suit : (i) l'action, représentée par une valeur numérique représentant l'état à attribuer à l'effecteur ; (ii) les plages de valeurs, modélisées sous la forme d'AVRT (section 5.2.2), qui sont initialisées à partir des valeurs des différentes données au moment de la réalisation de l'action par l'utilisateur ; (iii) le niveau de confiance, initialisé à 0.5 ; (iv) le gain estimé, initialisé d'après le processus décrit en section 5.2.3.

Les mécanismes d'évolution menant à la création des agents *contexte* ainsi que les raisonnements des agents *contrôleur* pour choisir l'action la plus appropriée parmi les propositions d'actions des agents *contexte*, permettent à un AMAS *dispositif* d'apprendre un comportement pour le dispositif associé grâce à un apprentissage *en ligne* basé uniquement sur des observations d'actions utilisateurs.

4.3. Évaluation des capacités d'apprentissage et d'ouverture

L'objectif de cette première évaluation est de montrer la capacité d'*Amadeus* à apprendre un comportement (non évolutif) utilisateur, tout en étant capable d'intégrer de nouvelles données en cours de fonctionnement.

4.3.1. Cadre d'étude

Nous avons développé un simulateur pour l'ensemble des évaluations réalisées. Il permet de concevoir un environnement dans lequel le concepteur insère des dispositifs et dans lequel nous ajoutons un ou plusieurs utilisateurs avec des préférences et des règles comportementales associées. La même procédure a été suivie pour chaque étude présentée : nous avons d'abord étudié les résultats obtenus en l'absence d'*Amadeus*, puis ceux obtenus avec *Amadeus* ajouté à la simulation. Pour chaque étude, 20 simulations différentes ont été réalisées afin d'obtenir des résultats plus pertinents.

4.3.2. *Protocole*

Nous simulons un utilisateur ayant comportement non évolutif placé durant 50 jours dans un appartement dont l'une des pièces est dotée de différents dispositifs : un capteur de luminosité, un capteur de présence, un capteur d'état (ouvert/fermé) fixé à la porte, une lampe et un store électrique. Un AMAS *dispositif* est associé à chaque dispositif dès le début de la simulation, excepté celui du store électrique qui est ajouté à mi simulation pour évaluer la capacité d'ouverture d'*Amadeus*.

4.3.3. *Résultats*

Le nombre d'actions réalisées par l'utilisateur sur la lampe au cours des différentes simulations en l'absence d'*Amadeus* est représenté en figure 3. En moyenne, l'utilisateur réalise 88 actions par jour (66 au minimum, 114 au maximum). Le nombre moyen d'actions par jour réalisées par l'utilisateur et par *Amadeus* une fois *Amadeus* ajouté aux simulations est visible en figure 4.

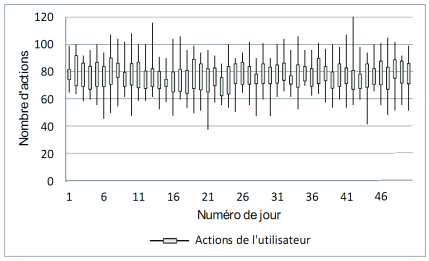

Figure 3. Nombre d'actions sur la lampe sans Amadeus

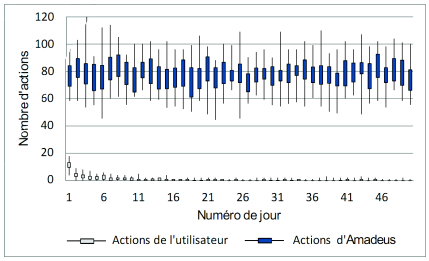

Figure 4. Nombre d'actions sur la lampe avec Amadeus

Dès le premier jour, l'utilisateur ne réalise plus que 13,5 % des actions qu'il aurait réalisées en l'absence d'*Amadeus*. Sur les 10 premiers jours, il n'en réalise plus que 3,6 %. Sur les 15 jours suivants, il n'en réalise que 0,7 %. Lorsque du jour 26, un AMAS *dispositif* est associé au store électrique, les agents *contexte* de l'AMAS *dispositif* associé à la lampe peuvent percevoir aussi l'état du store. Cette donnée n'a que peu d'effet sur le fonctionnement de la lampe, mais connaître son état permet à *Amadeus* d'affiner son contrôle sur la lampe. Sur la moitié de la simulation, l'utilisateur ne réalise donc plus que 0,4 % des actions qu'il aurait réalisées en l'absence d'*Amadeus*.

L'utilisateur réalise en moyenne 1,9 action par jour sur le store électrique en l'absence d'*Amadeus*. Sur les 10 jours suivant l'association du store avec un AMAS *dispositif*, l'utilisateur ne réalise plus que 4,7 % des actions qu'il aurait réalisées sans *Amadeus*, puis seulement 10,5 % sur les 15 jours suivants (figures 5 et 6).

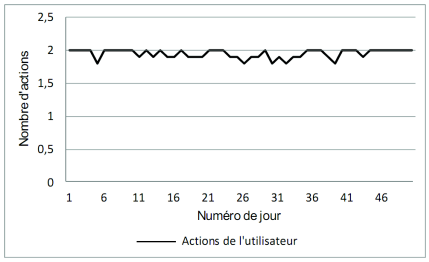

Figure 5. Nombre d'actions sur le store sans Amadeus

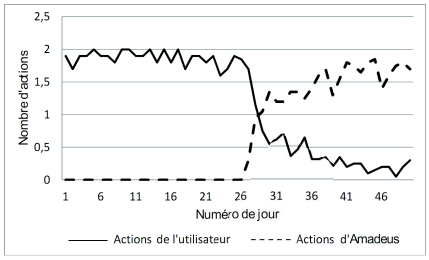

Figure 6. Nombre d'actions sur le store avec Amadeus

4.3.4. Discussion

Cette étude montre les capacités d'*Amadeus* à apprendre progressivement un comportement correct pour différents dispositifs par observation des actions de l'utilisateur. Cet apprentissage est local à chaque dispositif et permet l'ajout de données dans les perceptions d'un AMAS *dispositif* si un autre AMAS *dispositif* est intégré au système.

5. Changement de la fonctionnalité apprise

Nous nous intéressons à présent à la capacité d'*Amadeus* à modifier dynamiquement sa fonctionnalité, dès lors que le comportement évolutif des utilisateurs rend inadaptée la fonctionnalité apprise.

5.1. Détection des contradictions

L'adaptation d'*Amadeus* aux changements de comportement des utilisateurs se fait sans profil utilisateur et sans requérir de retour explicite des utilisateurs afin de ne pas gêner leur activité. Pour cela, nous rendons *Amadeus* capable d'introspection : il réalise des actions grâce à ses connaissances (ses agents *contexte*), puis observe si ces actions sont acceptées ou contredites par l'utilisateur. Détecter si une action réalisée par *Amadeus* sur un effecteur est acceptée ou contredite est à la charge de l'agent *contrôleur* associé à cet effecteur. Supposons une action A_1 réalisée par un agent *contexte* menant le système d'une situation S_1 à une situation S_2 ; par la suite, un utilisateur réalise une action A_2 menant le système d'une situation S_3 à la situation S_4. L'agent *contrôleur* doit alors déterminer si l'action A_2 réalisée par un utilisateur suite à l'action A_1 (ex : l'utilisateur éteint la lumière après qu'*Amadeus* l'ait allumée) est une contradiction ou une action indépendante. Nous posons qu'une action A_2 est une contradiction de l'action A_1 si elle vérifie les deux conditions suivantes : (*i*) A_2 est réalisée après A_1 sans qu'aucune autre action du système ou d'un utilisateur n'ait

été réalisée entre-temps sur ce même effecteur ; (*ii*) la situation S_1 dans laquelle A_1 a été réalisée est "similaire" à la situation S_4 qui suit la réalisation de A_2 (voir figure 7).

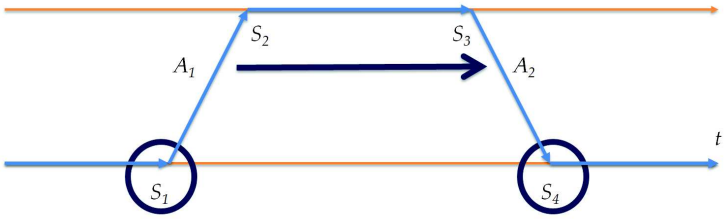

Figure 7. Illustration d'une contradiction perçue par un agent contrôleur

La similarité entre deux situations est évaluée par l'agent *contrôleur* en fonction des agents *contexte* valides. Lorsque l'action A_1 est réalisée, l'agent *contrôleur* enregistre la liste des agents *contexte* valides dans cette situation A_1 (avant la réalisation de l'action A_1). Lorsque l'action suivante A_2 est réalisée, l'agent *contrôleur* compare la liste des agents *contexte* alors valides dans la situation S_4 (après la réalisation de l'action A_2) avec celle précédemment mémorisée. Si ces deux listes sont identiques, l'agent *contrôleur* considère que les deux situations S_1 et S_4 sont similaires. Si ces deux conditions sont respectées, l'agent *contrôleur* conclut à une contradiction.

5.2. Adaptation aux contradictions

Lorsqu'un agent *contrôleur* détecte une contradiction, il prévient l'agent *contexte* à l'origine de l'action contredite qui doit alors s'adapter en conséquence. Pour cela, il va tout d'abord ajuster son niveau de confiance, puis tenter de s'améliorer. Il peut alors, soit exclure la situation S_1 des situations dans lesquelles il se considère comme valide, soit diminuer son gain estimé η afin qu'un autre agent *contexte* proposant une action différente soit sélectionné à sa place si la situation S_1 se présente à nouveau. Les sections suivantes précisent les différents ajustements réalisés par l'agent *contexte*. Plus de détails sont donnés dans (Guivarch *et al*., 2014).

5.2.1. Ajustement du niveau de confiance

La confiance d'un agent *contexte* est modélisée par une valeur numérique comprise entre 0 et 1 (0 signifiant une absence totale de confiance, et 1 une certitude totale). A la création d'un agent *contexte*, sa confiance est initialisée à 0,5. Celle-ci augmente lorsque l'agent *contexte* est sélectionné et que l'action qu'il a proposée n'est pas contredite. En revanche, elle diminue si l'agent *contexte* sélectionné est contredit.

Plus précisément, le calcul de la nouvelle confiance $T_{new} = T*(1-\lambda)+R*\lambda$ d'un agent *contexte* se base sur sa précédente confiance T et sur le retour R qu'il a reçu. Ce calcul est paramétré par une valeur réelle $\lambda \in [0;1]$ qui représente l'impact du retour sur le calcul du nouveau niveau de confiance. En l'absence de retour explicite des utilisateurs, l'agent *contrôleur* considère que le retour R est de 1 (augmentation

de la confiance) s'il n'y a pas eu de contradiction, ou de 0 (diminution de la confiance) s'il y en a eu une.

Le paramètre λ représente donc la dynamique supposée de l'environnement dans lequel est situé l'agent : plus sa valeur est importante, plus le niveau de confiance évolue rapidement en fonction des retours. Avec un paramètre λ fixé à 0,1, le dernier retour reçu n'influence donc que de 10 % la valeur du niveau de confiance. Ainsi, un agent *contexte* accorde plus d'importance au niveau de confiance qu'il possède qu'au retour qu'il a reçu, ce qui limite l'impact des éventuelles actions non pertinentes des utilisateurs. En revanche, si un retour est envoyé plusieurs fois successivement, il finit par modifier significativement le niveau de confiance de l'agent.

5.2.2. Exclusion de la situation incorrecte

En cas de contradiction, la première solution pour qu'un agent *contexte* améliore son comportement consiste à faire en sorte que, si la même situation S_1 se présente, l'agent *contexte* contredit ne soit pas valide afin qu'il ne soumette pas à nouveau sa proposition d'action dans une situation similaire. Il s'agit donc d'exclure la situation S_1 des plages de validité de l'agent *contexte*. Une situation étant représentée par l'ensemble des états courant de chaque donnée perçue, il suffit d'exclure la valeur courante d'au moins une plage de validité pour que la situation soit elle-même exclue.

Ces plages de validité sont modélisées sous la forme d'un AVRT (*Adaptive Value Range Tracker*, ou Traqueur de Plage de Valeurs Adaptatif), qui est une version étendue de l'AVT (*Adaptive Value Tracker*, ou Traqueur de Valeurs Adaptatif) proposé par (Lemouzy, 2011). L'objectif d'un AVT est de trouver la valeur d'une variable dynamique dans un espace donné, alors que celui d'un AVRT est de rechercher les bornes d'une plage de valeurs (voir figure 8) comprise entre deux valeurs v_{min} et v_{max}. Il se compose pour cela de deux AVTs, l'AVT$_{min}$ et l'AVT$_{max}$ représentant respectivement les valeurs minimales et maximales de la plage de valeurs.

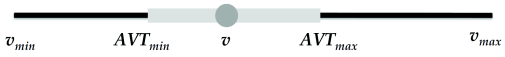

Figure 8. Représentation d'une plage de validité

L'AVRT propose un certain nombre de mécanismes, notamment un mécanisme d'exclusion de valeur qui, étant donnée une valeur v appartenant à la plage de valeur de l'AVRT, va tendre à exclure v. Ce mécanisme observe la borne la plus proche, et envoie un signal "plus grand" à l'AVT$_{min}$ si la borne min est la plus proche, ou un signal "plus petit" à l'AVT$_{max}$ si la borne max est la plus proche. Cette exclusion n'est pas directe : selon la proximité de la valeur v avec une des bornes de la plage de valeur, l'AVRT peut réussir à l'exclure du premier coup après modification d'une de ses bornes, ou bien uniquement tendre à l'exclure sans modifier suffisamment sa borne pour que v se retrouve en dehors de la plage de valeur. L'AVRT propose alors un second mécanisme qui, considérant cette même valeur v, sera capable de dire si l'AVRT est capable de l'exclure directement grâce à son mécanisme d'exclusion ou pas.

Lors d'une contradiction, s'il existe au moins une plage de validité chez l'agent *contexte* pour laquelle le mécanisme d'exclusion de l'AVRT est applicable, cette solution est suffisante pour résoudre la SNC. Elle n'implique qu'un mécanisme de tuning, car elle n'entraîne de modifications que dans l'état interne de l'agent *contexte*. L'agent *contexte* applique alors le mécanisme d'exclusion sur chaque plage de validité pour lequel une exclusion directe est possible. Dans la mesure où cette solution est applicable, elle sera préférée à la seconde solution, cette-ci impliquant un mécanisme de réorganisation.

5.2.3. Modification du gain estimé

La seconde méthode dont dispose un agent *contexte* contredit cherchant à améliorer son comportement consiste à modifier son gain estimé. Lorsque l'agent *contrôleur* détecte une contradiction, il envoie un message à l'agent *contexte* contredit, dans lequel il lui précise la liste des agents *contexte* valides en même temps que lui. Vu que son action s'est avérée inappropriée, l'agent *contexte* contredit considère qu'il aurait mieux valu que l'agent *contrôleur* sélectionne un autre agent *contexte* à sa place. Il va donc tenter d'ajuster son gain estimé, afin que celui-ci soit inférieur au gain estimé des autres agents *contexte* valides en même temps que lui.

Cependant, le gain estimé d'un agent *contexte* n'est pas une valeur numérique représentative en soi, mais plutôt une valeur relative entre plusieurs agents *contexte* permettant d'établir des relations d'ordre. Chaque agent *contexte* C possède une liste d'agents *contexte* Sup_C qui lui sont supérieurs, ainsi qu'une liste d'agents *contexte* Inf_C qui lui sont inférieurs. La formule 1 représente les effets d'une relation d'ordre entre deux agents *contexte* C_i et C_j sur leur gain estimé $\eta(C_i)$ et $\eta(C_j)$.

$$\forall C_i, C_j, C_i < C_j \Rightarrow \eta(C_i) < \eta(C_j) \tag{1}$$

De plus, la relation d'ordre entre les deux agents C_i et C_j est transitive, et est définie par la formule 2.

$$\forall C_i, C_j, C_i < C_j \Rightarrow \left| \begin{array}{l} C_j \in Sup_{C_i} \text{ et } C_i \in Inf_{C_j} \\ \\ \exists C_n \text{ tel que } C_n \in Sup_{C_i} \text{ et} \\ C_i \in Inf_{C_n} \text{ et } C_i < C_n < C_j \end{array} \right. \tag{2}$$

Pour ajuster son gain estimé, un agent *contexte* C contredit intègre l'ensemble des agents *contexte* qui étaient valides en même temps que lui à sa liste d'agents supérieurs Sup_C. Cet ajout entraîne automatiquement la mise à jour de sa valeur de gain estimé, qui devient alors inférieure à celles de tous les agents *contexte* valides en même temps que lui lors de sa sélection. Ainsi, si cet agent *contexte* se retrouve valide dans la même situation que celle où il a proposé son action contredite, il est assuré (sauf changement ultérieur dans les préférences des utilisateurs) qu'au moins un autre agent *contexte* appartenant à Sup_C proposera une action accompagnée d'un meilleur gain estimé que le sien, et sera donc sélectionné à sa place. Cet ajustement automatique

est le résultat d'un mécanisme d'auto-ordonnancement intégré aux agents *contexte*, détaillé notamment dans (Guivarch *et al*., 2014). La contradiction d'un agent *contexte* mène donc cet agent à ajouter des agents *contexte* dans sa liste d'agents supérieurs, puis à coopérer avec ces agents pour modifier leur gain estimé tout en minimisant ces modifications, puis à propager éventuellement ces ajustements d'agents en agents jusqu'à résolution de la totalité des conflits.

5.3. Évaluation des capacités d'apprentissage d'une fonction dynamique

La capacité de l'agent *contrôleur* à détecter les contradictions ainsi que les deux mécanismes des agents *contexte* pour s'adapter à ces contradictions, permettent à *Amadeus* d'adapter en continu sa fonctionnalité. L'objectif de cette évaluation est d'observer cette capacité d'évolution suite aux changements de comportement des utilisateurs.

5.3.1. Cadre d'étude

Pour observer la capacité d'*Amadeus* à faire évoluer sa fonctionnalité en cours de fonctionnement suite aux changements de comportement des utilisateurs, nous nous focalisons ici sur une étude basée sur une simulation de 50 jours impliquant 2 utilisateurs qui évoluent dans le même appartement que pour l'évaluation précédente.

5.3.2. Protocole

Le 1er utilisateur possède un comportement constant tandis que le 2nd utilisateur possède un comportement légèrement différent de celui du 1er utilisateur durant la première moitié des simulations puis un comportement radicalement différent durant la seconde moitié des simulations. Nous modifions donc le capteur, qui peut identifier quel utilisateur est présent ou pas, et donc transmettre deux données distinctes pour notifier si chaque utilisateur est présent ou pas. Puis nous associons un AMAS *dispositif* à chaque dispositif dès le début des simulations.

5.3.3. Résultats

En moyenne, les utilisateurs réalisent 120 actions par jour sur la première moitié des simulations, et 210 actions par jour sur la seconde moitié (figure 9).

En ajoutant *Amadeus*, sur les 10 premiers jours, les utilisateurs ne réalisent plus que 6,8 % des actions qu'ils auraient réalisées en l'absence d'*Amadeus*, puis 1,9 % sur les 15 jours suivants (figure 10). Lorsque le comportement du second utilisateur change radicalement (jour 26), le nombre d'actions utilisateurs augmente à nouveau, les amenant à réaliser 3,7 % des actions qu'ils auraient réalisées en l'absence d'*Amadeus*. Ce nombre diminue ensuite et atteint 1,7 % sur les 15 derniers jours des simulations grâce aux capacités d'*Amadeus* à faire évoluer sa fonctionnalité apprise.

Nous pouvons observer le faible nombre de contradictions réalisées par le 1er utilisateur (0,04 par jour en moyenne sur les 25 premiers jours puis 0,01 par jour en

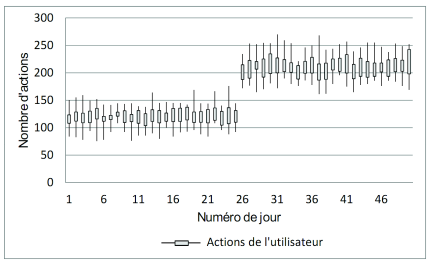

Figure 9. Nombre d'actions sur l'ensemble des dispositifs sans Amadeus

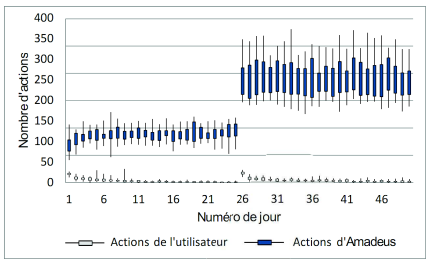

Figure 10. Nombre d'actions sur l'ensemble des dispositifs avec Amadeus

moyenne sur les 25 jours suivants), ces contradictions n'étant dues qu'à quelques actions erronées ponctuelles d'*Amadeus* (figure 11).

Le 2nd utilisateur obtient un résultat similaire pour la première moitié des simulations avec 0,06 % des actions réalisées en moyenne par jour, mais des résultats très différents sur la seconde moitié de la simulation avec en moyenne 1,94 contradictions par jour (figure 12).

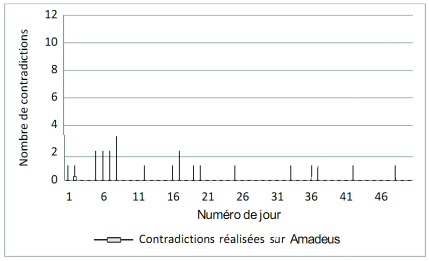

Figure 11. Nombre de contradictions réalisées par le 1er utilisateur

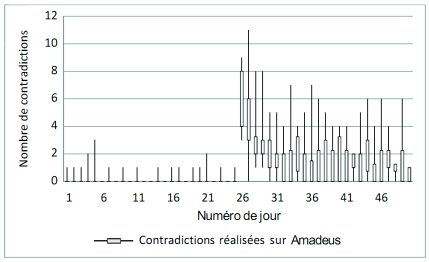

Figure 12. Nombre de contradictions réalisées par le 2nd utilisateur

5.3.4. Discussion

Cette étude montre les capacités d'*Amadeus* à permettre l'évolution de la fonctionnalité préalablement apprise sans qu'il soit nécessaire de réinitialiser son apprentissage. Cette évolution se fait (*i*) par un renforcement des comportements corrects (augmentation de la confiance des agents *contexte* non contredits), (*ii*) par un désapprentissage des comportements incorrects (perte de confiance, ajustement et réorganisation des agents *contexte*) et (*iii*) par intégration des nouveaux comportements (création de nouveaux agents *contexte*).

6. Distribution des données

Nous présentons ici la gestion des données distribuées entre les différents AMAS *dispositif*. Celle-ci résulte du fonctionnement des agents *capteur* et *effecteur* d'une part qui s'assurent de la transmission des données entre AMAS *dispositif*, et des agents *donnée* d'autre part qui s'assurent de l'utilité de leur donnée pour les agents *contexte* de leur voisinage.

6.1. Agents capteur *et* effecteur

Nous avons présenté les traitements permettant à chaque AMAS *dispositif* de réaliser l'apprentissage du comportement à attribuer à son dispositif. Cependant, les données nécessaires pour réaliser cet apprentissage se limitent rarement à celles issues des capteurs et effecteurs de ce même dispositif. Les différents AMAS *dispositif* doivent donc s'échanger les informations qui leur sont utiles. Il n'existe cependant pas d'agents *dispositif* à proprement parler, ce traitement est donc instancié aux niveaux des agents composant les AMAS *dispositif*.

Les agents *capteur* et *effecteur*, respectivement associés à chaque capteur/effecteur d'un dispositif, permettent ces échanges. Ils envoient leur mise à jour aux agents *donnée* des différents AMAS *dispositif* grâce à un mécanisme coopératif appelé la communication spontanée : chaque agent *capteur*/*effecteur* mémorise la dernière mise à jour envoyée et envoie spontanément toute nouvelle mise à jour aux agents *donnée* avec qui il partage cette donnée. Les agents *donnée* associés n'ont donc pas besoin de requérir de mise à jour (Camps, Gleizes, 1995). Ce mécanisme permet de prévenir l'apparition de SNC d'inutilité en cas d'envoi de mises à jour inutiles, et de SNC d'improductivité en cas de mises à jour pertinentes non transmises.

Au niveau d'un AMAS *dispositif*, en cas de réception de données depuis un agent *capteur* (ou *effecteur*) inconnu, un nouvel agent *donnée* est automatiquement créé pour représenter cette nouvelle donnée. La création de ce nouvel agent *donnée* permet alors aux agents *contexte* de percevoir cette nouvelle donnée.

Pour autant, si ces agents permettent les échanges suffisants entre AMAS *dispositif*, ils n'assurent pas que ces échanges se limitent aux seules données requises. Nous proposons donc un mécanisme de filtrage des données inutiles fonctionnant sans connaissance *a priori* et en parallèle du processus d'apprentissage d'*Amadeus*.

6.2. Agent donnée

Il est parfois possible que la connaissance de la valeur d'une donnée pour un agent *contexte* soit inutile, voire dégrade son fonctionnement. En effet, si une situation ayant précédemment mené à la réalisation d'une action par un utilisateur (et donc à la création d'un agent *contexte*) se présente à nouveau, mais avec seulement l'état d'une donnée inutile différent, l'agent *contexte* interprètera à tort qu'il s'agit d'une situation différente. Par exemple, si un agent *contexte* considère qu'il faut allumer la lumière

quand un utilisateur entre dans la pièce, un changement du niveau d'humidité perçue par un capteur d'humidité (donnée inutile) amène cet agent à se considérer dans une nouvelle situation, et donc à ne pas proposer son action.

Une SNC apparaît donc lorsqu'un agent *donnée* envoie sa valeur à un agent *contexte* qui n'en a pas besoin. Il s'agit d'une SNC de conflit entre l'agent *donnée*, dont l'objectif est de transmettre sa donnée aux agents *contexte* qui en ont besoin, et l'agent *contexte* qui peut se retrouver à se croire invalide à tort du fait de la présence de cette donnée inutile. Sa résolution s'appuie sur un processus local impliquant l'agent *donnée* et les différents agents *contexte* de son voisinage, afin d'évaluer non pas l'inutilité de la donnée, mais plutôt son utilité. L'objectif recherché est alors d'estimer le niveau d'utilité de la donnée en fonction des différents effecteurs, pour ensuite considérer comme inutile une donnée présentant un niveau d'utilité trop faible vis-à-vis d'un effecteur, et ainsi prévenir l'apparition de cette SNC (Guivarch *et al*., 2013).

6.2.1. Génération de signaux d'utilité

L'invalidité d'un agent *contexte* peut-être justifiée par deux causes : (i) il se peut qu'il soit invalide tout simplement parce que la situation courante n'est pas la même que celle de sa création ; (ii) il se peut aussi qu'une donnée inutile appartienne à la description de sa situation et, possédant une valeur différente de celle qu'elle avait au moment de la création de l'agent *contexte*, amène celui-ci à croire à tort qu'il est invalide. Pour savoir si une donnée qui l'invalide est inutile, un agent *contexte* doit savoir dans laquelle de ces deux situations il se trouve.

Considérons deux agents *contexte* C_v et C_i. Dans une situation donnée, l'agent C_v est valide, alors que l'agent C_i est invalide. L'agent C_v envoie alors sa proposition d'action à l'agent *contrôleur* qui le sélectionne et dit à l'agent *effecteur* associé de réaliser l'action proposée par C_v. De son côté, l'agent C_i observe son état interne afin d'évaluer s'il a eu raison de ne pas envoyer sa proposition d'action, autrement dit s'il a bien fait de ne pas être valide en même temps que l'agent C_v.

Si nous considérons que l'agent C_v n'est pas contredit suite à son action, nous pouvons considérer alors que l'action proposée par C_v était la bonne action à réaliser. A présent, si l'agent *contexte* C_i proposait une action différente de celle de C_v avec un gain estimé $\eta(C_i)$ supérieur au gain estimé $\eta(C_v)$, alors l'agent C_i sait qu'il a eu raison d'être invalide. En effet, s'il ne l'avait pas été, il aurait envoyé sa proposition d'action qui, différente de celle de C_v avec un meilleur gain estimé, aurait été sélectionnée à la place de celle de C_v. Or, la proposition de C_v étant considérée comme la proposition correcte en l'absence de contradiction, celle de C_i aurait probablement été incorrecte. En résumé, si nous considérons un agent *contexte* C_v valide, sélectionné et non contredit, alors un agent *contexte* C_i invalide sait qu'il a eu raison d'être invalide s'il respecte les conditions décrites par la formule 3.

$$Action(C_v) \neq Action(C_i) \; et \; \eta(C_v) < \eta(C_i) \tag{3}$$

Considérons l'ensemble des données D perçues par les agents *contexte* C_v et C_i, et le sous-ensemble $D_{inv} \subset D$ des données vis-à-vis duquel l'agent *contexte* C_i est invalide. Lorsque l'agent *contexte* C_i détermine qu'il a eu raison d'être invalide dans une situation particulière, il sait que c'est grâce aux données de D_{inv}. Cependant, s'il est certain qu'au moins une des données de D_{inv} est utile pour décrire la situation courante, il est possible qu'une autre donnée de D_{inv} soit une donnée inutile, mais vis-à-vis duquel l'agent *contexte* est invalide par simple coïncidence dans la situation courante. N'étant pas capable de différencier les données vraiment utiles de celles qui ne le sont pas, l'agent C_i envoie un "signal d'utilité" à chacun des agents *donnée* invalides appartenant à D_{inv} afin de les informer de leur potentielle utilité.

6.2.2. Traitement des signaux d'utilité

Considérons deux agents *donnée* D_u et D_i, D_u étant associé à une donnée utile et D_i étant associé à une donnée inutile vis-à-vis de la réalisation d'une action A. Le principal souci dans la détection de l'inutilité de la donnée D_i est que, parfois, sa valeur peut sembler utile à un agent *contexte*. Les agents D_u et D_i sont donc tous les deux susceptibles de recevoir des signaux d'utilité de la part d'agents *contexte*.

Cependant, lorsque la donnée de l'agent D_i est jugée utile, ce n'est qu'un *hasard* ; la réception d'un signal d'utilité par l'agent D_i est donc indépendante de la valeur de la donnée de D_i. Quelle que soit l'action A que propose l'agent *contexte* à l'origine du signal d'utilité reçu, les lois de distribution de la valeur de la donnée de D_i sont les mêmes (il n'y a pas de corrélation entre la valeur de D_i et l'action proposée par les agents *contexte* ayant envoyé des signaux d'utilité à D_i). C'est pourquoi, si un agent *contexte* envoyant un signal d'utilité lui associe son action, alors la valeur de la donnée de l'agent D_i à la réception de ce signal d'utilité suit la même fonction de densité théorique quelle que soit l'action associée au signal d'utilité.

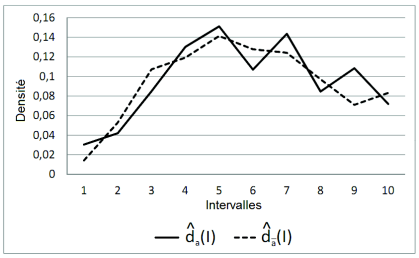

Figure 13. Illustration de la similarité entre les fonctions de densité d'une donnée inutile

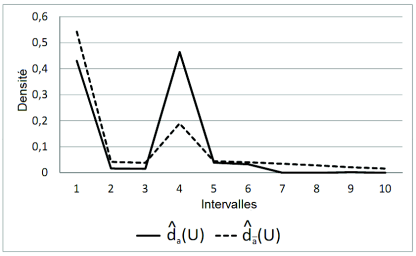

Figure 14. Illustration de la différence entre les fonctions de densité d'une donnée utile

En revanche, la valeur de la donnée représentée par l'agent D_u étant utile, il y a une corrélation entre sa valeur et la génération des signaux d'utilité. Ainsi, si la donnée de l'agent D_u est utile à la réalisation de l'action A, il en résulte une différence entre

les fonctions de densité que suit la valeur de cette donnée à la réception d'un signal d'utilité selon que ce signal soit associé à l'action A ou pas.

Au fur et à mesure que des signaux d'utilité sont reçus par un agent *donnée* D, celui-ci construit pour toute action A deux fonctions de densité empirique $\hat{d}_A(D)$ et $\hat{d}_{\overline{A}}(D)$, la première représentant sa valeur courante à la réception de signaux d'utilité associés à l'action A, et la seconde sa valeur courante à la réception de signaux d'utilité associés à tout autre action que A. Les figures 13 et 14 illustrent la différence entre les fonctions de densité pour un agent *donnée* utile et inutile.

Pour tout agent *donnée* D, la comparaison entre deux fonctions de densité $\hat{d}_A(D)$ et $\hat{d}_{\overline{A}}(D)$ renvoie une valeur représentant la distance entre ces deux fonctions, et est réalisée à travers le calcul de la distance du *chi-square* (Greenwood, Nikulin, 1996) donnée par la formule 4.

$$\delta\left(\hat{d}_A(D), \hat{d}_{\overline{A}}(D)\right) = \sum_{Valeur\ i} \frac{(\hat{d}_A(D)(i) - \hat{d}_{\overline{A}}(D)(i))^2}{\hat{d}_A(D)(i)} \tag{4}$$

L'utilisation de la distance du *chi-square* pour comparer deux fonctions de densité permet d'obtenir une valeur statistiquement significative. En effet, si nous supposons qu'une donnée est inutile, alors nous pouvons dire que $\delta\left(\hat{d}_{\overline{A}}(D), \hat{d}_A(D)\right)$ suit une loi de *chi-square* (Baillargeon, 1982), ce qui signifie que la loi suivie par la fonction de densité $\hat{d}_{\overline{A}}(D))$ est la même que celle suivie par la fonction de densité $\hat{d}_A(D))$. Par conséquent, dès lors que la valeur renvoyée par la comparaison des fonctions de densité pour une action A devient suffisamment faible (en dessous d'un seuil τ fixé empiriquement), l'agent *donnée* va se considérer comme inutile vis-à-vis de cette action A. Il en informe alors l'ensemble des agents *contexte* proposant cette action, et cesse alors de leur envoyer spontanément des mises à jour concernant cette valeur. Étant donné que cette donnée est devenue inutile pour eux, ces agents *contexte* suppriment la plage de validité associée à cette donnée.

Au final, le traitement de détection et de résolution de la SNC d'inutilité d'un agent *donnée* vis-à-vis d'agents *contexte* permet de réaliser un filtrage des données inutiles en cours de fonctionnement et sans connaissance *a priori*. Un prétraitement peut éventuellement être réalisé en amont du système grâce à des méthodes exploitant par exemple la sémantique des données pour accélérer ou simplifier le processus de filtrage, mais notre objectif est de permettre à *Amadeus* de pouvoir fonctionner même en l'absence de sémantique associée aux données. Notre filtrage des données inutiles assure le respect du critère de généricité de notre système, tout prétraitement ne permettant alors que d'optimiser les performances d'*Amadeus* sans être pour autant nécessaire à son fonctionnement.

6.3. Évaluation des capacités de filtrage des données inutiles

L'objectif de cette dernière évaluation est d'observer la capacité d'*Amadeus* à réaliser son apprentissage en présence de données inutiles.

6.3.1. Cadre d'étude

Nous considérons 2 utilisateurs aux comportements non évolutifs situés dans un appartement composé de 4 pièces, 1 capteur de luminosité et 1 capteur de présence par pièce (sans distinction des utilisateurs, car ils possèdent les mêmes règles de comportement), 1 capteur d'état pour chacune des 6 portes, 1 capteur de température extérieure, 1 lampe et 1 radiateur (3 états) par pièce. Chaque AMAS *dispositif* perçoit un total de 23 données dont seules 4 ou 5 sont utiles à son fonctionnement. De plus, le seuil τ pour le filtrage des données inutiles est fixé à 0,01.

6.3.2. Résultats

Le nombre moyen d'actions réalisées chaque jour par l'ensemble des utilisateurs sur la totalité des 8 dispositifs dotés d'un effecteur (lampes et chauffages) est égal en moyenne à 711 (il varie de 602 à 835) sur l'ensemble des simulations (figure 15).

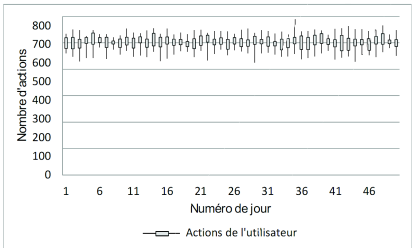

Figure 15. Nombre d'actions sur l'ensemble des dispositifs sans Amadeus

La figure 16 représente le nombre moyen d'actions par jour réalisées par les utilisateurs et par *Amadeus* sans doter les agents *donnée* de mécanisme de filtrage des données inutiles. Sur la totalité des simulations, les utilisateurs réalisent en moyenne 19,7 % des actions qu'ils auraient réalisées en l'absence d'*Amadeus*. Durant les 10 derniers jours des simulations, les utilisateurs réalisent toujours 13,6 %. En revanche, lorsqu'*Amadeus* est doté de son mécanisme de filtrage des données inutiles (figure 17), les utilisateurs ne réalisent en moyenne que 5,9 % des actions qu'ils auraient réalisées en l'absence d'*Amadeus* sur la totalité des simulations. Durant les 10 derniers jours des simulations, ils n'en réalisent plus que 1,9 %. A la fin du dixième jour nous obtenons un taux moyen de 38,4 % de données inutiles filtrées. Sur l'ensemble des simulations, 71,6 % des données inutiles sont filtrées et non 100 %, mais nous obtenons un taux d'erreur de 0 % (données utiles filtrées à tort).

6.3.3. Discussion

Cette étude montre une forte amélioration de l'apprentissage réalisé par *Amadeus* en présence de données inutiles dès lors que nous le dotons d'un mécanisme pour les filtrer. La principale limite de ce filtrage est qu'il est basé sur l'utilisation d'un seuil τ fixé empiriquement. Une valeur suffisamment stricte de ce paramètre τ permet d'éviter des filtrages incorrects. Cependant, en cas de filtrage à tort dû à une valeur de τ trop laxiste, le fonctionnement actuel d'*Amadeus* ne permet pas de détecter cette erreur

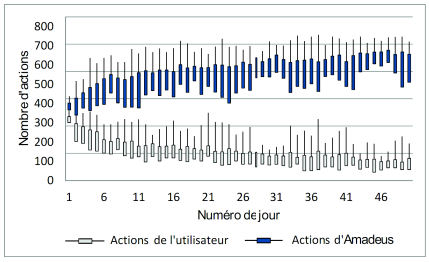

Figure 16. Nombre d'actions sur l'ensemble des dispositifs avec Amadeus *sans filtrage*

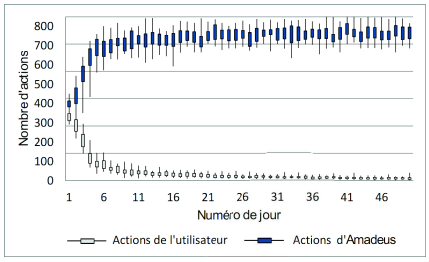

Figure 17. Nombre d'actions sur l'ensemble des dispositifs avec Amadeus *avec filtrage*

et de réintégrer la donnée filtrée, ce qui constitue une seconde limite forte qui sera discutée en conclusion. En revanche, ce mécanisme de filtrage présente l'avantage de fonctionner sans connaissance *a priori* et en parallèle du processus d'apprentissage.

7. Conclusion

Ce papier présente *Amadeus*, un AMAS d'AMAS permettant d'apprendre un comportement correct à attribuer à un système ambiant par observation des actions des utilisateurs. Au fur et à mesure que ces derniers réalisent des actions, *Amadeus* enrichit ses connaissances sur les actions qu'il peut réaliser, puis réussit progressivement à les réaliser à leur place. Cet apprentissage, réalisé en continu, est capable d'évoluer en cas de changement des comportements des utilisateurs ou survenant dans l'environnement. Il est réalisé sans connaissance *a priori* (ni profil des utilisateurs, ni sémantique associée aux données perçues), et localement à chaque dispositif, grâce à une distribution des données et des traitements ne nécessitant aucun composant central. La conception d'*Amadeus* a notamment impliqué la conception d'outils génériques : l'AVRT utilisé pour modéliser les plages de validité (section 5.2.2), le mécanisme d'auto-ordonnancement utilisé par les agents *contexte* (section 5.2.3), ou encore le mécanisme de filtrage des données inutiles (section 6.2).

Outre sa propriété de proactivité, *Amadeus* possède la propriété de généricité grâce à son exploitation des données sous forme numérique. Cette généricité est obtenue aux dépens de la propriété d'expressivité, aucune sémantique n'étant associée aux données perçues. Cependant, il est toujours possible d'ajouter de la sémantique aux données afin de réaliser des prétraitements améliorant les capacités d'*Amadeus*, mais nous considérons qu'il est important qu'il soit capable de fonctionner en l'absence de telles informations. *Amadeus* est également ouvert et distribué, mais il ne gère pas la confidentialité des données qu'il utilise. Une gestion locale des données apporte un début de réponse en ne rendant pas nécessaire la centralisation de la totalité des données. L'explicabilité d'*Amadeus* constitue une de nos perspectives ; il s'agit de concevoir

un outil capable d'extraire *a posteriori* de l'ensemble des agents *contexte* une base de règles expliquant son fonctionnement. Si nous reprenons les systèmes présentés en section 2.1.2, nous pouvons considérer que *Amadeus* est capable d'enrichir l'utilisation d'un gestionnaire de contexte ouvert et distribué, tel que SPACES (Romero *et al*., 2010), en y rajoutant l'aspect adaptation.

Actuellement, des travaux sont réalisés pour mettre en place des protocoles de comparaison de notre système avec des algorithmes d'apprentissage classique. De plus, le système *Amadeus* est en cours d'application dans le cadre d'une nouvelle thèse Cifre en collaboration avec Sogeti High Tech portant sur l'apprentissage d'un comportement correct pour un robot (Verstaevel *et al*., 2015), ainsi que dans le cadre du projet ANR INCOME [2] pour la détection de situation. Ces nouveaux cadres d'application permettent d'envisager différentes améliorations, telles que la prise en compte de données sémantiques, l'utilisation de données à partir non seulement de sa valeur mais aussi de sa vitesse (et donc indirectement des situations passées). Notre principale perspective concerne notamment l'amélioration de la sélection des données utiles en permettant aux agents d'*Amadeus* de détecter l'absence de données utiles nécessaires à leur fonctionnement. Un tel processus pourrait s'enrichir de l'utilisation d'un système complémentaire, tel que celui présenté par (Olaru, 2011), pour rechercher les données pertinentes manquantes parmi les données inexploitées jusqu'à présent. Ce processus serait notamment indispensable pour résoudre la principale faiblesse du filtrage des données inutiles, à savoir l'incapacité d'*Amadeus* à réintégrer une donnée filtrée si elle a été filtrée à tort, ou si les changements des comportements utilisateur ont rendu utile une donnée précédemment inutile. Enfin, nous envisageons de réaliser des évaluations supplémentaires après amélioration du simulateur utilisé, puis à plus long terme dans le cadre d'un système ambiant réel.

Bibliographie

Baek S.-H., Choi E.-C., Huh J.-D. (2007). Design of information management model for sensor based context-aware service in ubiquitous home. In *International conference on convergence information technology*, p. 1040–1047. IEEE.

Baillargeon G. (1982). *Introduction à l'inférence statistique: méthodes d'échantillonnage, estimation, tests d'hypothèses, corrélation linéaire, droite de régression et test du khi-deux avec applications diverses* (Les Editions SMG éd.).

Bonjean N., Mefteh W., Gleizes M.-P., Maurel C., Migeon F. (2013). Adelfe 2.0. In *Handbook on agent-oriented design processes.* Springer.

Brézillon P. (2010). *Modélisation et management des contextes*. Rapport technique n° MCI. Laboratoire d'Informatique de Paris 6.

Camps V., Gleizes M.-P. (1995). Analysis of altruistic behaviors in multi-agent systems. In *Int. Worshop on Decentralized Intelligent and Multi-agent systems, Poland,* , p. 93–103. Don Wydawnictw Naukowych S. C.

2. http://www.irit.fr/-INCOME

Capera D. (2005). *Systèmes multi-agents adaptatifs pour la résolution de problèmes: Application à la conception de mécanismes*. Thèse de doctorat, Université Paul Sabatier.

Chen H., Finin T., Joshi A. (2005). *Semantic web in the context broker architecture*. Rapport technique. DTIC Document.

Cornuéjols A., Miclet L. (2011). *Apprentissage artificiel*. Eyrolles.

Coutaz J., Crowley J. L., Dobson S., Garlan D. (2005). Context is key. *Communications of the ACM*, p. 49–53.

Dey A. K., Abowd G. (2000). Towards a better understanding of context and context-awareness. In *Chi 2000 workshop on the what, who, where, when, and how of context-awareness*, p. 304–307. Springer.

Dey A. K., Abowd G. D., Salber D. (2001). A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *HCI*, p. 97–166.

Euzenat J., Pierson J., Ramparany F. (2008). Dynamic context management for pervasive applications. *The Knowledge Engineering Review*, p. 21–49.

FitzGerald E., Taylor C., Craven M. (2013). To the castle! a comparison of two audio guides to enable public discovery of historical events. *Personal and Ubiquitous Computing*.

Franklin D., Flaschbart J. (1998). All gadget and no representation makes jack a dull environment. In *Proceedings of the aaai 1998 spring symposium on intelligent environments.*

Georgé J.-P., Gleizes M.-P., Camps V. (2011). Cooperation. In *Self-organising software*, p. 193–226. Springer.

Goumopoulos C., Kameas A., Hagras H., Callaghan V., Gardner M., Minker W. *et al*. (2008). Atraco: Adaptive and trusted ambient ecologies. In *Sasow*, p. 96–101.

Greenwood P. E., Nikulin M. S. (1996). *A guide to chi-squared testing*. Wiley.

Guivarch V. (2014). *Prise en compte de la dynamique du contexte pour les systèmes ambiants par systèmes multi-agents adaptatifs*. Thèse de doctorat, Univ. de Toulouse.

Guivarch V., Camps V., Péninou A. (2012). Context awareness in ambient systems by an adaptive multi-agent approach. In *Int. joint conference on ambient intelligence, pisa.*

Guivarch V., Camps V., Péninou A., Glize P. (2014). Self-adaptation of a learnt behaviour by detecting and by managing user's implicit contradictions (regular paper). In and (Ed.), *International Conference on Intelligent Agent Technology (IAT), Warsaw, Poland.*

Guivarch V., Camps V., Péninou A., Stuker S. (2013). Self-organizing mas by dynamic filtering of useless data: evaluation in the ambient domain. In *Paams, salamanca*, p. 110–121.

Hagras H., Callaghan V., Colley M., Clarke G., Pounds-Cornish A., Duman H. (2004). Creating an ambient-intelligence environment using embedded agents. *Intelligent Systems, IEEE*, vol. 19, n° 6, p. 12–20.

Hofer T., Schwinger W., Pichler M., Leonhartsberger G., Altmann J., Retschitzegger W. (2003). Context-awareness on mobile devices-the hydrogen approach. In *Hicss'03*, p. 10–19.

Korkea-Aho M. (2000). *Context-aware applications survey.* Consulté sur http://www.cse.tkk.fi/fi/opinnot/T-110.5190/2000/applications/context-aware.html

Kotsiantis S. B. (2007). Supervised machine learning: a review of classification techniques. *Informatica*.

Lemouzy S. (2011). *Systèmes interactifs auto-adaptatifs par systèmes multi-agents auto-organisateurs: application à la personnalisation de l'accès à l'information*. Thèse de doctorat, Université de Toulouse.

Minker W., Heinroth T., Kameas A., Hagras H., Helvert J. van, Bellik Y. *et al*. (2010). *D05 - project final report*. Public Deliverable. The ATRACO Project.

Mozer M. C., Miller D. (1998). Parsing the stream of time: The value of event-based segmentation in a complex real-world control problem. In *Adaptive processing of sequences and data structures*, p. 370–388. Springer.

Olaru A. (2011). *Un système multi-agent sensible au contexte pour les environnements d'intelligence ambiante*. Thèse de doctorat, Université Pierre et Marie Curie.

Picard G. (2004). *Méthodologie de développement de sma adaptatifs et conception de logiciels à fonctionnalité émergente*. Thèse de doctorat, Université Paul Sabatier.

Reignier P. (2010). *Intelligence ambiante pro-active: de la spécification à l'implémentation*. Thèse de doctorat, Université Joseph-Fourier-Grenoble I.

Rey G. (2005). *Contexte en interaction homme-machine: le contexteur.* Thèse de doctorat, Université Joseph Fourier, Grenoble.

Romero D., Rouvoy R., Seinturier L., Chabridon S., Conan D., Pessemier N. (2010). Enabling context-aware web services: a middleware approach for ubiquitous environments. *Enabling Context-Aware Web Services: Methods, Architectures, and Technologies*, p. 113–135.

Rottenberg S., Leriche S., Lecocq C., Taconet C. *et al*. (2012). Vers une définition d'un système réparti multi-échelle. In *8èmes journées francophones mobilité et ubiquité*, p. 178–183.

Russell S. J., Norvig P. (1995). *Artificial intelligence: a modern approach, 3rd edition*. Prentice Hall, Person Education Inc.

Spanoudakis N. I., Moraitis P. (2006). Agent based architecture in an ambient intelligence context. In *Proceedings of the 4th european workshop on multi-agent systems*, p. 1–12.

Strassner J., Meer S., O Sullivan D., Dobson S. (2009). The use of context-aware policies and ontologies to facilitate business-aware network management. *Journal of Network and Systems Management*, vol. 17, n° 3, p. 255–284.

Sutton R. S., Barto A. G. (1998). *Reinf. learning: An introduction*. Cambridge Univ Press.

Tapia D. I., Bajo J., Sanchez J. M., Corchado J. M. (2008). An ambient intelligence based multi-agent architecture. In *Developing ambient intelligence*, p. 68–78. Springer.

Verstaevel N., Régis C., Guivarch V., Gleizes M.-P., Robert F. (2015). Extreme sensitive robotic a context-aware ubiquitous learning (short paper). In *International conference on agents and artificial intelligence (icaart), lisbonne.*

Zaidenberg S. (2009). *Apprentissage par renforcement de modeles de contexte pour l'informatique ambiante*. Thèse de doctorat non publiée, Grenoble-INPG.

Annexe : présentation du simulateur de système ambiant

Notre simulateur de système ambiant permet de concevoir un environnement virtuel dans lequel le concepteur peut rajouter des dispositifs et des utilisateurs virtuels. Actuellement, le simulateur permet d'établir la topologie de l'environnement (nombre de pièces, position des portes et des fenêtres, etc.), ainsi que d'ajouter des capteurs (capteurs de luminosité, de température, de présence et d'état ouvert/fermé pour les volets, portes ou fenêtres) et des effecteurs (ex : lampes allumées/éteintes, chauffages éteints/à puissance moyenne/à puissance forte). Le simulateur gère ensuite les échanges thermiques et lumineux à l'intérieur du monde virtuel en fonction du temps et de l'état des différents dispositifs. Au cours d'une journée, la température et la luminosité extérieure (qui influencent les luminosités et les températures des pièces de la simulation) évoluent progressivement entre un minimum et un maximum.

Nous avons intégré des utilisateurs virtuels à notre simulation. Pour nos études, ce comportement doit être régulier, mais non basé sur un scénario préétabli. Les utilisateurs virtuels doivent donc être dotés d'un comportement général décrivant leurs activités, associé à des préférences sur leur environnement. Ces préférences doivent être accompagnées de la liste des actions à réaliser pour les satisfaire. Le comportement d'un utilisateur est donc décrit par des règles de préférence et des règles de comportement, les règles de préférence étant appliquées en priorité par rapport aux règles de comportement. Chaque règle est composée d'une liste de conditions et d'actions.

Lorsque la simulation débute, le simulateur regarde d'abord les conditions des règles de préférence de chaque utilisateur, et conserve alors les règles pour lesquelles les conditions sont respectées. Il va alors réaliser en priorité les actions associées à ces préférences. Par exemple, si un utilisateur préfère avoir la luminosité au dessus d'un certain seuil, alors cette préférence aura pour condition que la luminosité de la pièce courante est en dessous de ce seuil, et aura pour action le fait d'allumer la lampe.

Chaque fois qu'un utilisateur virtuel termine une action, le simulateur vérifie ses règles de préférence. Dès lors qu'elles sont toutes respectées et que l'utilisateur n'a plus d'actions à réaliser, le simulateur s'intéresse aux règles de comportement de cet utilisateur. Ces règles sont aussi composées de conditions à respecter pour être valides, et d'actions à réaliser le cas échéant. Ce sont ces règles qui permettent de décrire notamment les déplacements des utilisateurs. De plus, la réalisation d'une action peut générer d'autres actions (se déplacer générant la succession d'actions "se déplacer d'un pas" ainsi que les actions d'ouverture et de fermeture des portes).

L'ensemble des règles de comportement et de préférence est décrit par un fichier XML. Il est aussi possible de paramétrer à nouveau les règles des utilisateurs avec un autre fichier XML en cours de simulation, pour modifier les comportements et préférences des utilisateurs.La génération des déplacements d'un utilisateur est réalisée à partir d'un générateur de nombres pseudo-aléatoires qui, étant donnée une valeur initiale appelée *seed*, génère une suite de valeurs aléatoires. L'utilisation d'une *seed* identique permet de regénérer exactement la même suite de valeurs aléatoires, permettant ainsi de relancer plusieurs fois la même simulation.