Self-adaptation of a learnt behaviour by detecting and by managing user's implicit contradictions

Valérian Guivarch
IRIT, Toulouse, France
Valerian.Guivarch@irit.fr

Valérie Camps
IRIT, Toulouse, France
Valerie.Camps@irit.fr

André Péninou
IRIT, Toulouse, France
Andre.Peninou@irit.fr

Pierre Glize
IRIT, Toulouse, France
Pierre.Glize@irit.fr

***Abstract*—This paper tackles the issue of ambient systems adaptation to users' needs while the environment and users' preferences evolve continuously. We propose the adaptive multi-agent system Amadeus whose goal is to learn from users' actions and contexts how to perform actions on behalf of the users in similar contexts. However, considering the possible changes of users preferences, a previously learnt behaviour may become misfit. So, Amadeus must be able to observe if its actions on the system are contradicted by the users or not, without requiring any explicit feedback. The aim of this paper is to present the introspection capabilities of Amadeus in order to detect users contradictions and to self-adapt its behaviour at runtime. These mechanisms are then evaluated through a case study.**

I. Introduction

The evolution of computer science towards "Pervasive Computing" makes necessary the design of new "systems" that have to manage a large amount of information distributed among numerous devices. These systems are plunged into dynamic environments, they have evolutive configurations and functionalities and they are composed of heterogeneous devices. As a global control of such systems is impossible to implement, a solution is then to design them in an ascendant way. Thus, there is a real need to have devices and applications that are able to adapt their behaviour at runtime, as a response to the actions and needs of users that use such systems.

The aim of this paper is to contribute to the design of a system able to ensure a consistent management of an ambient system in its entirely. This system has to take into account at runtime the constraints previously enumerated, as well as the evolution of users' behaviours. More concretely we propose a proactive system, called *Amadeus*, based on the AMAS (Adaptive Multi-Agent System) approach, which is able to observe users during their activity in order to learn how to perform their actions on their behalf. For that, we assume that if a user performs an action in a given context, his action is the correct action to perform if a similar context occurs again. This paper completes a previous work [5] that presents the functional architecture of *Amadeus* in order to learn the behaviour to assign to an ambient system depending on the context. Section II describes related works regarding existing learning algorithms. Section III is devoted to the presentation of the main notions and principals of the *Amadeus* AMAS, including the contradiction issue, which is the new capability of *Amadeus* introduced in this paper. Section IV focuses on this new capability, more precisely on a mechanism implemented between agents for solving contradictions. Section V contains evaluations as well as an analysis of obtained results. We conclude and plan some future works in section VI.

II. Context and learning

The objective of our work is to design a "context-aware" system, namely a system able to adapt an ambient system depending on users' context [3] [13]. Our system has to be able to learn what is the correct behaviour to adopt in order to satisfy users. For this, our system aims to continually interact with its environment (especially users), to establish the most satisfactory functionality and to autonomously adapt using its environment feedbacks. Autonomous adaptation is defined [10] as *"the ability for a system to dynamically change, at runtime and without the intervention of an external entity, its way of acting according to the observed behaviour in the environment in response to its actions and in order to provide a service, a stable functionality through time, despite changes in the environment"*. To obtain this result, we studied the relevance of existing learning algorithms in the field of ambient systems.

Most of learning algorithms belong to the three main families of learning algorithms: supervised learning [9], unsupervised learning [1], and reinforcement learning [14]. We will not explain unsupervised learning algorithms because such algorithms allow dividing a set of data into different categories, without associating a specific class with these categories. In this paper we are interested in learning the behaviour to give to an ambient system depending on the perceived context, that is to say we want to associate any perceived situation with the action to perform in this situation. Thus, unsupervised learning algorithms are misfit to such a problem.

A. Supervised learning algorithms

In a supervised learning algorithm, the learner receives a set of inputs. Each input consists of many attributes that can take discrete or continual values. Moreover, to each of these inputs is associated a class from the finite set of possible classes. A supervised learning algorithm has a set of examples illustrating the objective to learn. From these examples, it tries to build a model that allows it to give the correct class for each input [9]. In other words, the learning algorithm considers that if its model allows it to attribute the correct class to each example, this model is close enough to the reality; consequently if the algorithm perceives a new input, it will be able to associate this input with the correct class. Artificial Neural Networks [12], Genetic Algorithms [6], Bayesian Network [7] as well as Case-Based Reasoning [8] belong to this category of algorithms.

B. Reinforcement learning

At each cycle, a reinforcement-learning algorithm perceives the current state of its environment and the list of the actions

it can perform. It chooses one of these actions, and then at the next cycle, it perceives the new environment state as well as a reward value that allows it to evaluate the quality of the action chosen in the previous state. Thus, the goal of a reinforcement-learning algorithm is to maximize the reward over time. This gain can be assessed in many ways: cumulated gain, average gain, etc. However, as such an algorithm does not have any *a priori* knowledge about the effect of its actions, it can only determine the reward that it will receive by trying this action for a given situation. This reward is not always the same at each time; so, determining what is the best action to perform for a given situation requires an exploration phase, where the algorithm has to try each action a number of times. In the exploitation phase, the algorithm exploits the acquired knowledge in order to choose the action with the best-expected gain. Finding the right balance between these two phases (exploration and exploitation) is one of the main issues of such algorithms. The model built by a reinforcement-learning algorithm is called a *politic*. It determines the action to perform depending on the current state. The optimal politic is the one that, whatever the environment state, allows to determine the action for which the expected gain is the highest. A reinforcement learning algorithm seeks to determine this optimal politic, or at least to find the closest politic. Q-learning [15] and SARSA [14] belong to this category of algorithms.

C. Discussion

In the framework of ambient systems, by combining user's actions with contextual situations indicating where/when these actions took place, it is possible to use supervised learning algorithms to create a cases base that will be used to learn the intended functionality. However, this approach has some limitations. More precisely, in ambient systems, it is necessary to be able to self-adapt to unexpected situations, such as the appearance or disappearance of devices, the user's preferences evolution, etc. Most of supervised algorithms address this issue by restarting their learning from the beginning. Due to these limitations, reinforcement-learning algorithms seem to be a good solution. Several variants have been proposed to overcome the long time they require to learn. For example the context management system proposed by Zaidenberg [16] uses an indirect reinforcement-learning algorithm. However, a learning algorithm based on a "trial/error" process does not seem to be appropriate for the context-aware control of an ambient system. Indeed, such a control requires the exploration of new solutions that may be bad solutions (inappropriate actions in such contexts). Furthermore, in such a domain, any error may disrupt the user who may finally reject the system.

Considering our very specific needs and constraints, very few learning algorithms are relevant. We are looking for an alternative to these algorithms in order to design a system for learning users' behaviour in ambient systems.

III. *Amadeus*

The AMAS approach seems to be a good alternative. It has been defined by our research team [2] [4] in order to design adaptive multi-agent systems enabling to solve complex problems that can be incompletely specified and for which an *a priori* known algorithmic solution does not exist. It considers the system as composed of parts (i.e. agents) and focuses on the local behaviour to give to these agents for making them adaptive (to their local environment) while ensuring that the collective behaviour that emerges from interactions between agents is the one expected; in that case the system is said "functionally adequate". To this end and in this approach, each agent pursuing a perceive/decide/act lifecycle must have a local cooperative behaviour. Our definition of cooperation is not a conventional one (resource sharing or the fact of working together). It is based on three local meta-rules that the designer has to instantiate depending on the problem to be solved and that have to be locally checked by every agent (for more details see [2] and [4]).

The AMAS approach also incorporates the notion of criticality, defined as the "*distance between the current situation and the local purpose of the agent*" [10]. Thus, "*the more the agent is far from its goal, the more it considers its current situation as being critical*". If we consider this notion, an agent is cooperative if it acts in order to help the most critical agent of its neighbourhood. So all agents within an AMAS tries to continuously reduce the criticality of the most critical agent (possibly itself), while avoiding another agent becoming even more critical. If an agent is found not to be able to help the most critical agent of its neighborhood, it may help other less critical agents. Thus, doing so, it hopes these agents will be able to help the most critical agent thanks to the reduction of their own criticality.

A. Presentation of Amadeus

Amadeus is an adaptive multi-agent system (AMAS) composed of several *device* AMAS (figure 1). A *device* AMAS is associated to every device of the ambient system, and the set of the *device* AMAS composes *Amadeus*. The local objective of each *device* AMAS is to determine the correct behaviour to give to its device. For this, it observes the users' actions, and learns in which situation it is possible to perform these actions on their behalf. More precisely, a *device* AMAS is composed of three types of agents:

1) A *controller* agent that is linked with every effector. It decides which action to perform on this effector depending on the current context;
2) A *context* agent that is linked with a *controller* agent. It proposes to its *controller* agent a particular action to perform in a specific situation;
3) A *data* agent that is linked with each data perceived from a sensor (local or remote). It is responsible for the propagation of the data to the *context* agents of its *device* AMAS, and for the usefulness evaluation of these data towards these *context* agents.

The goal of a *controller* agent is to decide at anytime what is the best action to perform on the effector (with which it is linked) on behalf of the user. This decision is made thanks to a set of *context* agents.

A *context* agent is created by the *controller* agent every time a user performs an action (for example to turn on the light). This *context* agent associates this action with a description of the situation in which the user has performed this action. This situation is composed of the set of perceived data values when the action is performed (example: Light=0;

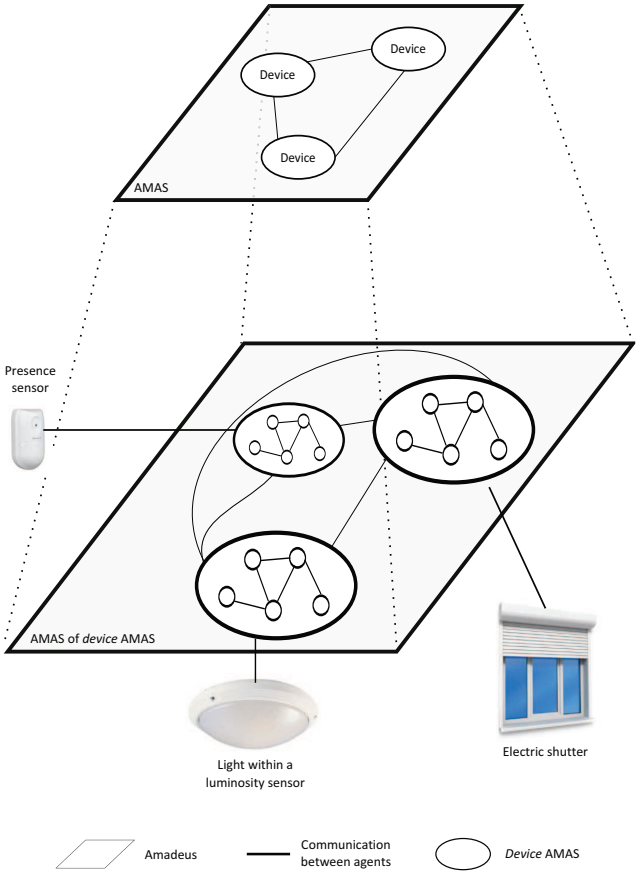

Fig. 1. *Amadeus* architecture

PresenceSensor=1; LuminositySensor=22). The action is represented by the value given to the effector (for example, 1 to turn on and 0 to turn off the light).

Every *context* agent perceives a set of data sent by *data* agents, and so, coming from local sensors situated on either the same device or on distant sensors (situated on another *device* AMAS). Each of perceived data possesses a validity status that depends on its current value compared to the situation described by the *context* agent. A data is considered as valid if it belongs to a range of values. This range represents the values interval that a piece of data may have in order to describe a situation. Thus, the *context* agent tries to establish the borders of valuable ranges for every perceived data that enables it to describe the correct situation for which its action proposition is appropriate (namely its action will have the expected effects). To do this, the *context* agent possesses, for each perceived data, an Adaptive Value Range Tracker (AVRT) that is a data structure enabling to describe a valuable interval (called "interval of validity") where the min and max borders can evolve. The value of each border is estimated with an Adaptive Value Tracker (AVT) [10], which is a tool devoted to the tuning of evolving parameters. Thus, a *context* agent considers a data as valid if its current value belongs to its associated AVRT.

A *context* agent has also its own validity status. Its status is valid if all perceived data are valid (invalid otherwise). If so, a *context* agent sends its action proposition associated with an estimated gain to the *controller* agent. The *controller* agent can then decide which action, among those proposed by all valid *context* agents, is the most appropriate to satisfy the user.

A *context* agent also possesses an estimated gain η. It is a numerical value that represents a belief on the interest to perform its action rather than another agent's action. More details on this value are given in section III-B3. In the previous versions of *Amadeus* [5], the estimated gain was a simulated value so that each *context* agent can send an explicit proposition about the interest to perform its proposed action. Such an evaluation could be performed thanks to a user profile. However, our objective is that *Amadeus* performs its learning process without any *a priori* knowledge about the users. So, we propose to give introspection capabilities to *Amadeus* agents in order to determine the actions that will satisfy the users, and those that will disturb them.

B. Contradiction issue

A *controller* agent performs on its effector an action proposed by a *context* agent because it considers that this action will satisfy the users. However, the *context* agent can be wrong, either because its action is proposed in a wrong situation or because its action is inappropriate due to the evolution of users' preferences.

The *controller* agent does not perceive any explicit feedback from users' satisfaction. However, we assume that when *Amadeus* performs an action on a given effector that does not agree with a user's preferences, this user will perform a corrective action on the same effector in order to restore a context that agrees with his preferences. Such a user's action is defined as a contradiction. A *controller* agent has to be able to evaluate if a performed action is contradicted by a user or not (if the user is satisfied or not). Indeed, if an action of *Amadeus* is contradicted by a user, we can suppose that *Amadeus* was wrong to perform this action, whereas if this action is not contradicted, *Amadeus* acted properly.

A user's action may be of two kinds: (*i*) a "normal" action performed to change the environment state, in order to fulfil the user's preferences in a specific situation (low luminosity level, etc.); (*ii*) a contradiction performed to correct a previous action made by *Amadeus*.

1) Contradiction detection: We will now explain how a *controller* agent is able to detect if a user's action is a contradiction, and how concerned *context* agents can adapt themselves to resolve this contradiction. We consider the action a performed by a *context* agent, and the next action a_u performed by a user. The goal of the *controller* agent is to determine if a_u is a contradiction of a. We consider a_u as a contradiction of a if (*i*) the action a_u is performed just after a, without any other action performed (either by the system or a user) on the same effector between a and a_u; (*ii*) the situation before a was performed is "similar" to the situation once a_u is performed. A *controller* agent only perceives situations through the action propositions sent by the *context* agents. So, from its point of view, a situation is not characterised by the current state of the perceived data, but by the set of the valid *context* agents. Therefore, if two situations S_1 and S_2 are similar, then the same *context* agents are valid in each of these

situations. So, when an action a is performed, the *controller* agent records the list of *context* agents that are valid in this situation (just before the action a is performed). When the next action a_u is performed, the *controller* agent compares the list of *context* agents currently valid (just after a_u is performed) with the previously recorded list. If these lists are identical, the *controller* agent detects that the second action a_u is a contradiction of the first action a. It sends then a contradiction signal to the *context* agent that proposed the action a.

Conversely, if *Amadeus* contradicts a user's action, we consider that the *Amadeus* action is always wrong. The user's action is always dominating. In that case, the *context* agent that proposed the action (that contradicted the user's action) receives a contradiction signal from the *controller* agent.

Finally, when an action of *Amadeus* is contradicted by another action of *Amadeus* (self-contradiction), we consider that both of the *Amadeus* actions are wrong.

2) Context *agents adaptation (first proposition):* A first solution consists in making invalid the contradicted *context* agent in the situation where it proposed its action. Thus it will not send its action proposition if the same situation occurs again. As a situation is represented by the set of states of each perceived data at a given time, it is sufficient to exclude the current data of at least one validity range in order to exclude the situation itself. The AVRT defined to model each validity range, provides a mechanism to exclude a value v from its values range. This exclusion is not instantaneous: it depends on the proximity between v and the closest boundary of the range values. The AVRT can directly exclude v by modifying one of its boundaries. It can also tend to exclude v by changing the value of its boundary but not sufficiently to make v ending outside of the values range. If there is at least one validity range in the contradicted *context* agent for which the exclusion mechanism can lead to the exclusion of v, this solution is applied in order to resolve the contradiction.

3) Context *agents adaptation (second proposition):* If the first proposition cannot be applied, another solution is to change the estimated gain of the contradicted *context* agent. When the *controller* agent sends a contradiction to a *context* agent, it completes its message by giving the list of *context* agents that were valid at the same time. Then, a contradicted *context* agent c adds these agents in its list of superior agents Sup_c (symmetrical, each new superior *context* agent c_{sup} adds c in its list of inferior agents $Inf_{c_{sup}}$). This addition causes the automatic update of the estimated gain value of c, so that this value becomes lower than the estimated gain value of all the *context* agents of the list. Thus, if c becomes valid for the same situation as the one when it proposed its contradicted action, it can be sure that at least one other *context* agent will propose another action with a bigger estimated gain, and so will be selected. The automatic update of the estimated gain value is performed thanks to a mechanism belonging to c and is explained in the next section.

IV. Ordering mechanism between *context* agents

A. Definitions and properties

The proposed ordering mechanism ensures that if a *context* agent c_i is inferior to another *context* agent c_j, then the estimated gain of c_i is lower than the estimated gain of c_j. Lec C be the set of all the *context* agents; each agent $c \in C$ has a list Sup_c of superior *context* agents and a list Inf_c of inferior *context* agents. The ordering relation between two agents c_i and c_j is transitive and is defined by the formula 1.

$$\forall c_i, c_j \in C, c_i < c_j \Rightarrow \left| \begin{array}{l} c_j \in Sup_{c_i} \text{ and } c_i \in Inf_{c_j} \\ \\ \exists c_n \in C \text{ such as } c_n \in Sup_{c_i} \text{ and} \\ c_i \in Inf_{c_n} \text{ and } c_i < c_n < c_j \end{array} \right. \quad (1)$$

The *context* agent $c \in C$ is supposed to have a list of superior *context* agents Sup_c, and a list of inferior *context* agents Inf_c. Its goal is to provide its estimated gain value $\eta(c)$. This value has to respect the constraints given by the formulae 2 and 3.

$$\forall i \in Inf_c, \eta(i) < \eta(c) \quad (2)$$

$$\forall s \in Sup_c, \eta(c) < \eta(s) \quad (3)$$

We define the "*degree of freedom*" $\delta(c_i, c_j)$ between two *context* agents c_i and c_j linked with an ordering relation, as the absolute value of the difference between their estimated gain values $\eta(c_i)$ and $\eta(c_j)$ (formula 4).

$$\delta(c_i, c_j) = |\eta(c_i) - \eta(c_j)| \quad (4)$$

$\delta(c_i, c_j)$ is the maximal value that the inferior agent can add (respectively, that the superior agent can remove) to its estimated gain) without breaking their ordering relations.

A *context* agent is also characterized by (*i*) its "superior degree of freedom" $\delta_{SUP}(c)$ that represents the degree of freedom between itself and the superior agent having the lowest estimated gain, and by (*ii*) its "inferior degree of freedom" $\delta_{INF}(c)$ that represents the degree of freedom between itself and the inferior agent having the highest estimated gain. These values can be seen as the flexibility the *context* agent c has in order to modify its estimated gain $\eta(c)$. Moreover, a *context* agent knows the estimated gain $\eta(s)$ and the superior degrees of freedom $\delta_{SUP}(s)$ of each superior agent s, as well as the estimated gain $\eta(i)$ and the inferior degrees of freedom $\delta_{SUP}(i)$ of each inferior agent i.

B. Resolution of the incoherence between two context *agents*

Let us consider a new *context* agent a_{new} added to the list of superior agents Sup_a of another *context* agent a. The agents a and a_{new} have their own estimated gain value $\eta(a)$ and $\eta(a_{new})$. Considering that $a < a_{new}$, we are supposed to have $\eta(a) < \eta(a_{new})$. If this inequality is not checked, in other words if $\eta(a) \geq \eta(a_{new})$, the agents a and a_{new} have to cooperate in order to solve this conflict.

Two solutions are possible: either a decreases its estimated gain $\eta(a)$, or a_{new} increases $\eta(a_{new})$. At every cycle, each agent has to determine if it is the most appropriate to modify its value, or if it seems more appropriate to let the other agent modify its value. To determine which agent has to act, we use the notion of criticality, defined in section III. We define the criticality of a *context* agent as its inability to modify its estimated gain value. Because *context* agents are cooperative, they seek to decrease the criticality of the most critical agent.

In our example, agent a seeks to decrease its estimated gain $\eta(a)$ under $\eta(a_{new})$. The more its ability to decrease its

estimated gain without being in conflict with other agents is low (in other words, its inferior degree of freedom $\delta_{INF}(a)$ is low), the more a is critical. In the same way, the agent a_{new} seeks to increase its estimated gain above $\eta(a)$, so the more its superior degree of freedom $\delta_{SUP}(a_{new})$ is low, the more a_{new} is critical. The least critical agent modifies its value. If it is a, a decrements the value of $\eta(a)$, whereas if it is a_{new}, a_{new} increments $\eta(a_{new})$. This resolution is made incrementally. At each step, the agents a and a_{new} determine which one is the most critical. Then this agent increments (or decrements) its value by 1. This step is repeated until the conflict is solved (so $\eta(a) < \eta(a_{new})$).

Let us consider $MaxInf_a$ the list of the inferior agents of a having the highest estimated gain of Inf_a, and $MinSup_{a_{new}}$ the list of superior agents of a having the lower estimated gain of $Sup_{a_{new}}$. To determine which agent between a and a_{new} is the most critical, a and a_{new} compare their respective degree of freedom $\delta_{INF}(a)$ and $\delta_{SUP}(a_{new})$. The one having the lower degree of freedom is the most critical. For example (figure 2), as the inferior degree of freedom $\delta_{INF}(a)$ is equal to 2 due to the agent i_2 and the degree of superior freedom $\delta_{SUP}(a_{new})$ is equal to 1 due to the agent s_1, a_{new} is the most critical agent.

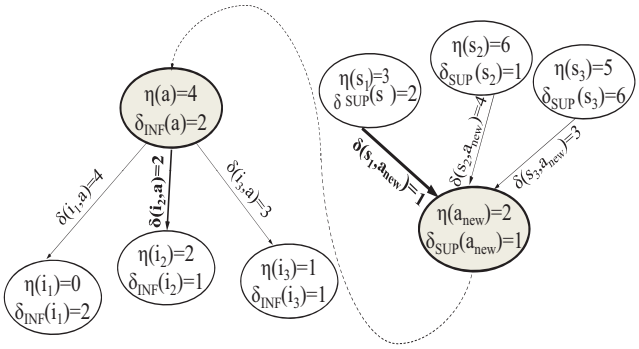

Fig. 2. Evaluation of the most critical agent by comparing $\delta_{INF}(a)$ and $\delta_{SUP}(a_{new})$

If $\delta_{INF}(a)$ and $\delta_{SUP}(a_{new})$ are equals a and a_{new} compare the number of agents that limit their respective degree of freedom (in other words, the sizes of $MaxInf_a$ and $MinSup_{a_{new}}$); the agent that has the biggest list is the most critical. For instance, in the figure 3, the agent a has an inferior degree of freedom $\delta_{INF}(a)$ equal to 2 (because of **two** inferior agents i_2 and i_3), while a_{new} has a superior degree of freedom $\delta_{SUP}(a_{new})$ equal to 2 (because of only **one** superior agent s_1). The agent a is then the most critical.

If $MaxInf_a$ and $MinSup_{a_{new}}$ have the same size the agents a and a_{new} have to determine which one is the most critical depending on the degree of freedom of the agents of $MaxInf_a$ and $MinSup_{a_{new}}$. The inferior degree of freedom $\min_{i \in MaxInf_a} \delta_{INF}(i)$ of the agent of $MaxInf_a$ having the lowest inferior degree $\delta_{INF}(i)$ is compared to the superior degree of freedom $\min_{s \in MinSup_{anew}} \delta_{SUP}(s)$ of the agent of $MinSup_{a_{new}}$ having the lowest superior degree of freedom $\delta_{SUP}(s)$. The agent having the lowest value is the most critical. For example (figure 4), the lowest inferior degree $\min_{i \in MaxInf_a} \delta_{INF}(i)$ among the agents of $MaxInf_a$ is equal to

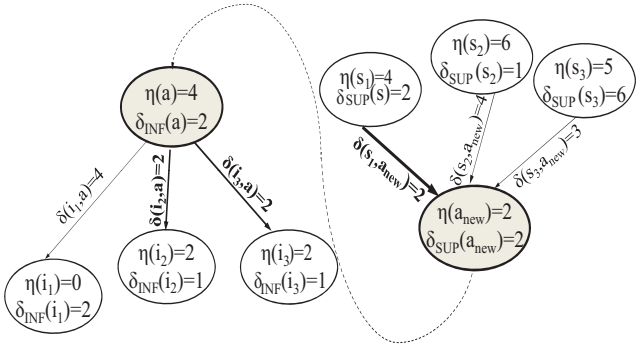

Fig. 3. Evaluation of the most critical agent by comparing $Size(MaxInf_a)$ and $Size(MinSup_{a_{new}})$

those of i_2 and i_3 (that is to say equal to 1), while the lowest superior degree $\min_{s \in MinSup_{anew}} \delta_{SUP}(s)$ among the agents of $MinSup_{a_{new}}$ is the one of s_1 (equal to 2). The agent a is then the most critical.

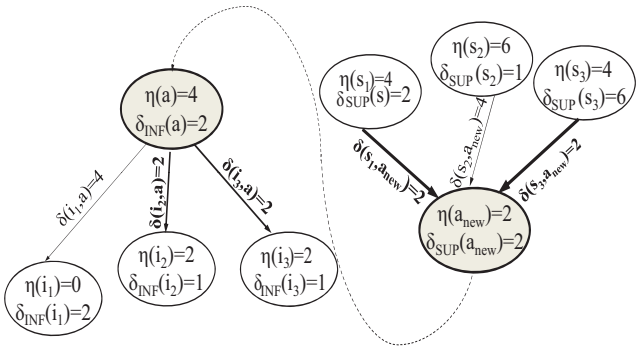

Fig. 4. Evaluation of the most critical agent by comparing $\max_{i \in Inf_a} \delta_{INF}(i)$ and $\min_{s \in Sup_{anew}} \delta_{SUP}(s)$

In case of a perfect equality of the criticality the *context* agent that started this process (the agent a in the example) modifies its estimated gain. The algorithm 1 summarizes all cases to define the most critical agents between two *self-ordered* agents (a and a_{new}).

However, the conflict resolution between two agents a and a_{new} can induce other conflicts (for example, the decrease of the estimated gain of a can bring it to have an estimated gain lower than those of one of its inferior agents), and the resolution of these new conflicts can induce other conflicts, etc. These disturbances therefore propagate step by step until all conflicts have disappeared. A last case can then appear; let us consider the resolution of the conflict between a and a_{new} that brings, by propagation, an agent a_{sup} superior to the agent a (directly or by transitivity) to be in conflict with an agent a_{inf} inferior to the agent a. The agent a_{inf} is then unable to increase its estimated gain, as the agent a_{sup} is unable to decrease its. Indeed, the agent a_{inf} cannot be lower than a and higher than a_{sup} at the same time. In this case, we consider than the oldest ordering relationship has to disappear.

Algorithm 1 Evaluation of the most critical *self_ordered* agent between a and a_{new}

If $\delta_{INF}(a) < \delta_{SUP}(a_{new})$ **Then**
  Return(a)
Else If $\delta_{INF}(a) > \delta_{SUP}(a_{new})$ **Then**
  Return(a_{new})
Else
  $MaxInf_a = \{\}$
  $\eta_{max} = (\max_{i \in Inf_a} \delta_{INF}(i))$, the maximal inferior degree of the Inf_a agents
  For all agent $i \in Inf_a$, the list of a inferior agents a **Do**
    If $\eta(i) = \eta_{max}$ **Then**
      $MaxInf_a$.Add(i)
    End If
  End For
  $MinSup_{a_{new}} = \{\}$
  $\eta_{min} = (\min_{s \in Sup_{anew}} \delta_{SUP}(s))$, the minimal superior degree of the $Sup_{a_{new}} agents$
  For all agent $s \in Sup_{a_{new}}$, the list of a_{new} superior agents **Do**
    If $\eta(s) = \eta_{min}$ **Then**
      $MinSup_{a_{new}}$.Add(s)
    End If
  End For
  If $Size(MaxInf_a) < Size(MinSup_{a_{new}})$ **Then**
    Return(a)
  Else If $Size(MaxInf_a) > Size(MinSup_{a_{new}})$ **Then**
    Return(a_{new})
  Else If $\eta_{max} < \eta_{min}$ **Then**
    Return(a)
  Else If $\eta_{max} > \eta_{min}$ **Then**
    Return(a_{new})
  Else
    Return(a) by default
  End If
End If

V. Evaluation

The proposed solution was implemented using Speadl/MAY [11], which is a tool to assemble reusable components in order to build architectures supporting the development and execution of multi-agent systems. Our solution was evaluated through a simulator allowing to simulate a virtual environment, to design a behaviour for virtual users, and then to simulate the users' actions in their environment. Each virtual user is initialized with a set of preferences (about luminosity, temperature, etc.), and his behaviour (turn on lights for example) is generated depending of these preferences and the evolution of the user's environment. In this section, we study the capability of *Amadeus* to adapt itself to the change of a user's preferences, thanks to the mechanisms previously presented.

A. User's behaviour variation

We analyse first the effect of a small variation of the user's behaviour on the *Amadeus* learning. The largest part of the previously learnt behaviour of *Amadeus* remains correct, there are just some situations where it performs wrong actions.

1) Study framework: We simulate a room of an apartment having a light and an electric shutter. A *device* AMAS is associated with each device. We add a user who randomly walks in this apartment; he performs actions in order to make the room luminosity level satisfying when he is inside the room, and to have the light turned off when he is outside the room. Each simulation lasts 50 days. We perform a perturbation (corresponding to a user's behaviour change, through a small variation of his luminosity preferences) in the *Amadeus* environment at the 25th day. Concretely, he turns on the light or opens the shutter more quickly, while he waits more time before to turn off the light or to close the shutter. However, his behaviour is quite similar once his change of preferences is performed.

2) Results: We perform a set of 20 simulations of 50 days without *Amadeus*. The actions performed by the user are represented by the figure 5 : the average number of actions performed each day by the user increases from 78.9 in the first part of the simulation to 80.9 in the second part.

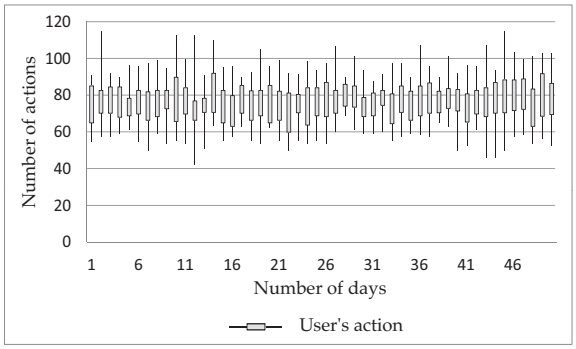

Fig. 5. Number of user's actions

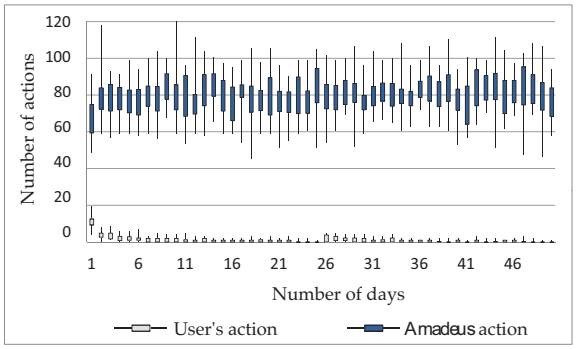

Fig. 6. Number of user and *Amadeus* actions

Now, we repeat the simulations by adding *Amadeus*. We seek to observe the effect of the user's behaviour modifications on the *Amadeus* behaviour. We represent the actions performed by the user and *Amadeus* in the figure 6. For the first half of the simulation, *Amadeus* performs on average 65.4 actions the first day versus 12.8 actions performed by the user; thus *Amadeus* performs a total of 83.6% of actions the first day. If we enlarge the comparison to the 10 first days, we obtain on average 76.4 actions by day performed by *Amadeus* versus 3.9 performed by the user, so we have a total of 95% of actions performed by *Amadeus*. Finally, on the 15 next days (which correspond

the first half of the simulation), *Amadeus* performs on average 78.6 actions by day while the user performs 0.9 action by day. With a total of 98.9% actions performed by *Amadeus* during these 15 days, we can consider that the learning is over when we change the user's preferences.

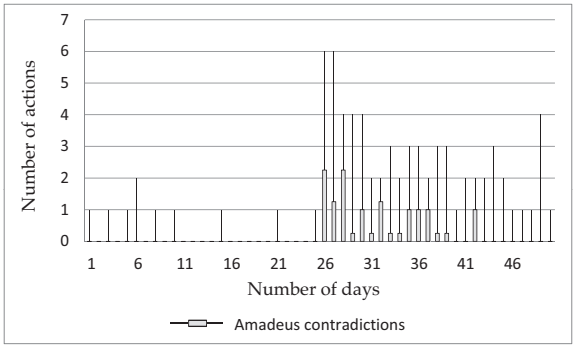

Fig. 7. Average number of *Amadeus* contradictions

When the variation of the user's behaviour occurs, we can observe a slight upturn of the user's actions. These actions allow to correct the *Amadeus* actions that became misfit to the new preferences of the user. The figure 7 shows more precisely the contradictions detected by the system through the simulations. In the first half of the simulations, the contradictions are punctual. Then, after the change of the user's preferences, the number of contradictions strongly increases, and then progressively decreases while *Amadeus* adapts its learning to the new user's behaviour. Finally, during the 10 first days of this second half of the simulation, the average number of actions performed by *Amadeus* is equal to 80.5, while the user performed on average 1.8 actions by day. If we compare these results with the number of actions performed by the user without *Amadeus* (equal to 80.3), we can observe an average increase of 2 actions by day. This increase is mostly due to contradictions, with on average 0.7 contradicted *Amadeus* actions by day (6 in the worst case) plus the user's contradictions (on average 0.7 actions by day). For the rest of the simulation (days 35 to 50), the average number of *Amadeus* actions by day is equal to 81.3 whereas the average number of user's actions by day is equal to 0.6. So, *Amadeus* performs 99.2% of the actions. Once the adaptation phase passed, *Amadeus* possesses a new behaviour that suits the new preferences of the user.

B. User's behaviour change

We analyse now the effect of a big change in the user's behaviour on the *Amadeus* learning. The goal of Amadeus is then to dynamically adapt itself to this radical change that completely invalidates the previously learnt behaviour.

1) Study framework: The framework is the same as the one of the previous study, but with a bigger variation of the user's preferences. The user wants now a very low luminosity level. He wants the light always turned off, and he keeps the shutter closed except if the luminosity level is very low (basically, he wants to stay in the dark, with the shutter opened just at the night). The objective is then to observe if *Amadeus* is able to modify completely its learning, by "unlearning" the behaviour previously learnt in order to adopt a more adequate behaviour. A set of 20 simulations without *Amadeus* is performed, and then the same 20 simulations with *Amadeus* are performed. For each of these simulations, we completely change the user's preferences at the end of the 25th day.

2) Results: The user actions without *Amadeus* are represented by the figure 8. For the first half of the simulation, the obtained results are the same as the results of the first study. For the second half of the simulation, the user decreases very strongly the number of actions he performs by day because he no longer uses the light.

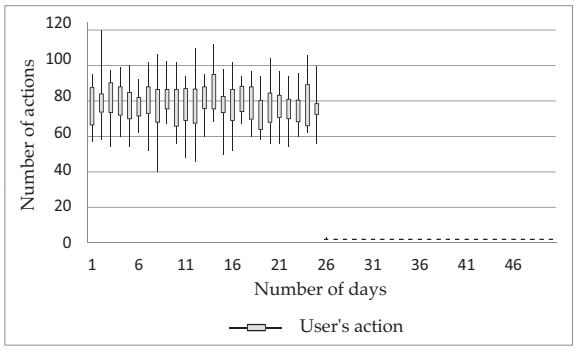

Fig. 8. Number of user's actions

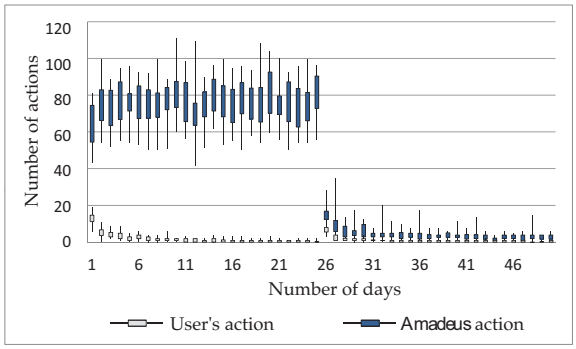

Fig. 9. Number of user and *Amadeus* actions

Amadeus is then added to simulations. The ability of *Amadeus* to learn the correct behaviour for the devices despite of the user's preferences change can be seen on figure 9. The *Amadeus* performances during the first half of the simulation are the same than in the first study. So, we can consider its learning is over when we apply our perturbation. Once the user's preferences have changed, we can observe an increase of the user's actions. These actions are composed of contradictions and other actions that represent the new user's behaviour.

The largest part of the contradictions is performed the first day (figure 10); the user corrects a big number of *Amadeus* actions. More precisely, an average number of 7,1 actions is performed by the user the 26th day. For the 10 first days of the second half of the simulation, we can observe an average number of 1,9 actions performed by the user. This number decreases until 0,4 for the 15 last days of the simulation. As the system adapts itself, the number of contradictions decreases progressively.

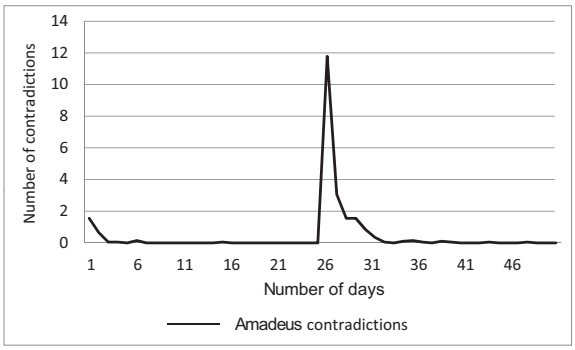

Fig. 10. Average number of *Amadeus* contradictions

C. Discussion

In the first study, the perturbation is relatively small, because we have limited the change to an increase of the luminosity level sought by the user. The user's behaviour is still more or less the same, except in some situations where the luminosity level is the old and the new threshold. By consequence, the necessary change in the behaviour learnt by *Amadeus* was small too. The largest part of the *context* agents created before the modification of the user's preferences remains still valid; only some of them became not adapted to the new preferences of the user, and so were contradicted. The *context* agents that are still correct continue to observe that their action propositions are not contradicted. Moreover, the introspection abilities of the incorrect *context* agents allow them to detect that their action propositions did not satisfy the user (when they are warned by their *controller* agent that they were corrected by the user), and so to adapt themselves. Then, we studied the effect of a big change in the user's preferences. This important change in the functionality to be learnt involves many contradictions of the *Amadeus* actions by the user. These contradictions decrease the estimated gain of the contradicted *context* agents, but also increase the estimated gain of the *context* agents that did not propose these actions. It is in this way that *Amadeus* performs its "unlearning", while it simultaneously learns the correct actions to perform.

This study shows the ability of *Amadeus* to react and adapt itself to a small and to a more important change in the functionality to learn. The originality of this capability is that it is not necessary to completely stop the *Amadeus* actions and to restart its learning from the beginning. The capability of the *context* agents to adapt themselves when they become incorrect allows *Amadeus* to modify only the part of its functionality that became incorrect without modifying the rest of its behaviour. Furthermore, experimentations with several users (having or not compatible preferences) have been performed. Even if they cannot be explained here due to lack of space, they showed that Amadeus was able to adapt a subset of its functionality in case of change of preferences of a user without disturbing the subset of its functionality for the other users.

VI. Conclusion

This paper presents an extended version of the multi-agent system *Amadeus*, devoted to the learning of users' behaviour in ambient systems. *Amadeus* can learn situations and actions performed by the user on effectors of devices in order to perform later these actions on behalf of the user. The originality of *Amadeus* relies on the fact that this learning is performed without any *a priori* knowledge. More precisely, it does not require any profile of the users, neither an explicit feedback of the users to evaluate its actions. Indeed, *Amadeus* observes the activity of users and possesses introspection capabilities in order to detect when its action does not suit the user. For that it is able to determine when a user contradicts the action it has performed. It is also able to take into account these contradictions in order to adapt its learnt behaviour to the preferences (possibly evolutive) of the user. We made some experiments that show the adaptive capabilities of *Amadeus* to take into account changes in the preferences of a user

Our main perspective is to apply *Amadeus* to a real ambient system, in order to evaluate more efficiently its abilities to dynamically learn a correct behaviour by observing actions and contradictions of users. Moreover, we are also interested in giving to *Amadeus* the capability to extract users' profiles from its knowledge. In particular, such profiles would enable to validate more efficiently the learnt behaviours.

References

[1] H. B Barlow. Unsupervised learning. *Neural computation*, 1(3):295–311, 1989.

[2] V. Camps. *Vers une théorie de l'auto-organisation dans les systèmes multi-agents baséee sur la coopération : application à la recherche d'information dans un système d'information répartie*. Thèse de doctorat, Univ. Paul Sabatier, 1998.

[3] A. K. Dey, G. D. Abowd, and D. Salber. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human–Computer Interaction*, 16:97–166, 2001.

[4] M.-P. Gleizes, V. Camps, J.-P. Georgé, and D. Capera. Engineering Systems which Generate Emergent Functionalities. In *Engineering Environment-Mediated Multiagent Systems - associated to ECCS, Dresden, Germany*, number 5049 in LNAI, pages 58–75. Springer, 2008.

[5] V. Guivarch, V. Camps, and A. Péninou. Context awareness in ambient systems by an adaptive multi-agent approach (regular paper). In *International Joint Conference on Ambient Intelligence, Pisa - Italy*, pages 129–144. Springer, novembre 2012.

[6] J. H Holland. *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence*. University of Michigan Press, 1975.

[7] F. V Jensen. *An introduction to Bayesian networks*, volume 210. University College London Press, 1996.

[8] J. L Kolodner. An introduction to case-based reasoning. *Artificial Intelligence Review*, 6(1):3–34, 1992.

[9] S. B Kotsiantis. Supervised machine learning: a review of classification techniques. *Informatica (035055596)*, 31(3), 2007.

[10] S. Lemouzy. *Systèmes interactifs auto-adaptatifs par systèmes multi-agents auto-organisateurs: application la personnalisation de l'accès l'information*. Thèse de doctorat, Univ. de Toulouse, juillet 2011.

[11] V. Noel. *Component-based Software Architectures and Multi-Agent Systems: Mutual and Complementary Contributions for Supporting Software Development*. Thèse de doctorat, Univ. de Toulouse, 2012.

[12] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 1958.

[13] Bill N. Schilit and Marvin M. Theimer. Disseminating active map information to mobile hosts. *Network, IEEE*, 8(5):22–32, 1994.

[14] R. S Sutton and A. G Barto. *Reinforcement learning: An introduction*, volume 1. Cambridge Univ Press, 1998.

[15] Christopher John Cornish Hellaby Watkins. *Learning from delayed rewards*. PhD thesis, University of Cambridge, 1989.

[16] S. Zaidenberg. *Apprentissage par renforcement de modèles de contexte pour l'informatique ambiante*. PhD thesis, INP Grenoble, 2009.