AMADEUS: an adaptive multi-agent system to learn a user's recurring actions in ambient systems

Valérian Guivarch[a], Valérie Camps[a], André Péninou[a]

[a] Institut de Recherche en Informatique de Toulouse, Université de Toulouse

KEYWORD

Multi-agent systems
Intelligent environments
Distributed algorithms
Adaptive environments

ABSTRACT

Ambient systems are characterized by their dynamics and their huge complexity. An important issue in this field is their capability to provide a relevant behaviour in order to satisfy users involved. Multi-agent systems, because of their ability to deal with dynamic, distributed and not deterministic environments, seem to be very promising to solve adaptation problems in ambient systems. The objective of our study is to propose Amadeus, a system able to learn the user's behaviour in order to perform his recurrent actions on his behalf, independently of the ambient system in which it is applied. The originality of our contribution is to be generic and to promote a process able to learn at runtime without any prior learning phase and able to filter useful data for characterizing users' context.

1 Introduction

Ambient systems consist of a large number of heterogeneous devices distributed in the environment; some of these devices can appear or disappear at runtime, sometimes temporarily. Moreover, these systems involve many users who make actions in the environment depending on their context and their evolving preferences. Such systems require adaptation techniques to take into account the dynamics of their environment and in order to provide relevant services to users who have evolving preferences and uses. To our knowledge, existing systems only propose ad-hoc rules to tackle this dynamics but such a solution is not sufficient in real applications.

Our contribution aims at proposing a solution to tackle the problem of adaptation in ambient systems. We propose to make an ambient system able to provide a relevant behaviour, based on the perceived user's actions, in order to assist him by realizing his actions on his behalf. It is based on the use of the multi-agent paradigm. This choice is motivated by the fact that such a paradigm provides solutions to problems that evolve in dynamic, partially accessible and not deterministic environments [RUSSELL, 1995].

Thus, we propose *Amadeus*, an Adaptive Multi-Agent System that is able to learn the user's contexts while he is performing actions in order to act on his behalf in similar situations. We make the assumption that if *Amadeus* performs actions on behalf of the user, these actions may increase the user's satisfaction.

Section 2 offers a review of the use of multi-agent systems in ambient systems. The general approach we propose to solve this problem is then detailed in section 3. After a general presentation of our multi-agent system, *Amadeus*, in section 4, the explanation of its functioning is split into two parts: a description of its functioning in order to realize its learning and an explanation of its ability to filter useless data. Some results are included in these two subsections. We conclude and plan future work in section 5.

2 The use of MAS in ambient systems: positioning

In MAS approaches, context management is not directly addressed in order to develop a generic multi-agent oriented framework dedicated to only manage context data. Conversely, MAS are used to manage applications in a "vertical" axis, from low-level context data towards higher-level context data. However, we can distinguish two categories of MAS dealing with context management: those that are strongly related to their application domains, and those that are more generic because they are not related to a specific application. Among the first ones we can distinguish DALICA [CONSTANTINI et al., 2008] devoted to context management in a museum, Dujardin's system [DUJARDIN et al., 2011] that enables context management for an automatic control of a house and ASK-IT [SPANOUDAKIS et al., 2006] that allows the users to reach easily the data and services that are near them. Among the second ones we can distinguish LAICA [CABRI et al., 2005] that proposes an ambient infrastructure at the urban scale, Spatial Agents [SATOH, 2004] that is a localization-aware system, Tapia's system [TAPIA et al., 2008] that is a CBR adaptation for ambient systems, and AmbieAgents [LECH et al., 2005] that enables context-aware information sharing for mobile users.

If we consider the capture of context data, the main common point between these various multi-agents infrastructures is the agentification associated with this process. Except the data explicitly provided by the designer or the user (by completing a form for example), several agents are generally in charge of the data recovery. We can distinguish systems composed of specific agents able to capture particular kinds of data (ASK-IT) from those composed of more generic agents able to capture any kind of data (Tapia and LAICA). Once perceived by an agent, the data are made available to the system. This can be done in a centralized way, from an agent to a central server or towards a central agent (Tapia). This can be also done in a distributed way, from an agent to all other agents (LAICA and Dujardin), or towards certain specific agents (ASK-IT system).

All the MAS do not explain how data are modeled. We can distinguish those that (*i*) use an ad-hoc format (DALICA) which simplifies processings but implies a design totally chargeable to the designer, from those that (ii) use existing technologies (such as an ontology, or cell in Satoh) that implies additional processings (deduction of information, logical reasoning) to convert the perceived data.

The format used to model the data greatly influences the interpretation phase of these data. When the modeling uses an ad-hoc format (DALICA and Dujardin) the data semantics depends on its format. A more generic modeling implies an interpretation phase between the data perception and their modeling. In particular, the modeling by ontology implies the association of one semantic to every data. The interpretation phase is generally chargeable to the designer at the design time of every system, none of them being apparently capable of interpreting at runtime the semantics of the perceived data.

	Architecture			Data collection					
	Organization	Nb agents types	Cognitive/ reactive agents	Agent in responsability	Send	Modelling	Interpretation	Situations detection	Adaptation
ASK-IT	Specific	6	Cognitive	Specific	To specific agents	Ontology (no precision)	Semantics associated at the instanciation	Based on ad-hoc rules	?
DALICA	Common structure	3	Both	Specific	To a central server	Ad-hoc (POI)	Ad-hoc	Based on ad-hoc rules	Based on ad-hoc rules
LAICA	Common structure	2	Both	Generic	To all agents	?	Ad-hoc	Based on ad-hoc rules	Based on ad-hoc rules
Tapia's system	Specific	5	Both	Generic	To a central server	?	?	Based on CBR	Based on CBR
Dujardin's system	Common structure	3	Reactive	Specific	To all agents	Ad-hoc (simulator)	Ad-hoc	Based on ad-hoc rules	Based on ad-hoc rules
Satoh's system	Common structure	2	Cognitive	Specific	To a central server	cell	Ad-hocaBased on ad-hoc rules	Based on ad-hoc rules	Based on ad-hoc rules
AmbieAgents	Specific	3	Cognitive	Generic	To a central server	Ontology (OntoSense)	Semantics associated at the instanciation	Based on ad-hoc rules	Based on ad-hoc rules

Tab.1. Review of the use of multi-agent systems in ambient systems

The situations detection is, for the majority of the systems, based on ad-hoc rules. Only the Tapia's system proposes a generic situations' detection. This detection remains most of the time very close to the system goal, and only the particular situations described in advance by the designer are detectable. It is then impossible, except for the Tapia's system that uses a Case-Based Reasoning, to dynamically detect new situations.

From an architecture point of view, the existing MASs for context management differ from the type of agents they use. Some of them use cognitive agents (ASK-IT, AmbieAgents and Satoh) while Dujardin's system uses reactive agents and others conjugate the simultaneous use of cognitive and reactive agents (LAICA and Tapia). Furthermore these systems are more or less heterogeneous: some of them have architectures composed of a small number of different types of agents (2 for Satoh and LAICA) while the others propose a bigger number of types of agents (6 for ASK-IT). Finally, the architectures differ from the complexity of their organizations. Some architectures limit themselves to agents all connected together through a common structure (a software bus for LAICA, a central server for DALICA), whereas the others adopt more complex organizations where every agent communicate with other agents according to their roles (Tapia and ASK-IT). However, the various organizations remain quite static, without evolution at runtime, and do not thus seems to take into account the environmental dynamics

Finally, from the adaptation viewpoint depending on the users' context, most of these systems are based on ad-hoc rules. Only the Tapia's system possesses learning capacities, by using a CBR algorithm in order to modify its behaviour when it is faced to new situations

The main observations we can formulate regarding these systems (Tab. 1) is that no proposal explicitly addresses context management independently of any kind of application and also the weakness of existing systems in terms of generic adaptation. Most of processings are performed in an ad-hoc way, sometimes in a centralized way, whereas the ambient systems imply a lot of dynamics. To overcome this lack, we propose to investigate on the use of a generic adaptive and distributed approach [GEORGE et al., 2011] in order to study how a system can improve its functionality at run-time and especially each time an unforeseen situation appears.

3 Our MAS proposition

The objective of this study is to design a system able to adapt the behaviour of an existing ambient system in order to satisfy each user that uses it. Our contribution currently focuses on a single user (called "the user" in the rest of the paper). One of our perspectives is to extends our work to several users.

We consider that the user is satisfied if the system is able to perform his recurring actions on his behalf. As highlighted in the previous section, the use of MAS is motivated by their ability to provide dynamic and distributed solutions. Moreover, our objective is to design a system able to work on any kind of ambient system rather than on a specific one.

3.1 Requirements

The first requirement concerns the genericity of the system to propose. It has to be able to deal with any kind of device, i.e. to integrate any type of data. This can be done either directly, using a generic modeling of data, or using an expandable model of data that can be improved with some new categories of data. For our system, we use all perceived data in a numerical form, directly usable.

The second requirement deals with the distribution aspect of the system to propose. An ambient system is by nature composed of numerous and heterogeneous devices. So, interact with the totality of such a system in a decentralized way, without any central component, seems to be the more relevant solution. For our system, we propose an exclusively distributed solution, without any centralized component.

Finally, the third requirement concerns the adaptation of the system to a user's actions. An ambient system realizing actions depending on ad-hoc rules in order to satisfy the user can become inappropriate if it is not able to adapt its behaviour to the user's preferences (that may evolve) or if the ambient system changes. So, our system has to be able to dynamically adapt its behaviour at runtime. With this aim in view, we propose to use the AMAS approach [GEORGE et al., 2011]. It is an organizational approach that consists in endowing each agent with local cooperative behaviors that do not directly depend on the overall function the system has to achieve.

These three requirements in mind, we designed an adaptive multi-agent system *Amadeus*. Its objective is to observe the user's actions on his environment, and to detect regularities in his behaviour in order to perform his actions on his behalf.

3.2 Proposed approach

We consider an ambient system as a set of devices, themselves being composed of sensors and effectors. These devices are distributed in the environment, and each device is connected with some of the other devices of the ambient system. An ambient system possesses a strongly dynamic infrastructure: devices can appear (new device) or disappear (removed or defective device), the connection between devices can evolve (mobile device can be temporally perceived).

A centralized solution to make such a dynamic system context-aware seems to be inappropriate, because such a solution requires being able to perceive, at any time, the state of the entire ambient system. That is why we propose to design *Amadeus* as a distributed and generic multi-agent system that has to be instantiate for every device composing the ambient system. To avoid any confusion, in this paper, "*Amadeus*" names the set of all the *Amadeus* instances in the ambient system, whereas an instance of *Amadeus* is always explicitly named "instance of *Amadeus*" (or "the instance").

An instance of *Amadeus* is in charge of the management of the device to which it is associated. Its objective is to perceive the user's actions on the effectors of the device and to learn, based on theses perceptions, a relevant behaviour for these effectors. For this, the instance perceives the user's actions on the effectors, observes in which situations these actions were performed, and tries progressively to learn what action it has to perform in every situation. As a matter of fact, we consider that the best solution in order to learn a relevant behaviour for a device without *a priori* knowledge is to base this learning on the user's actions. The objective of our contribution thus is to perform actions on behalf of the user.

4 Amadeus

This section is devoted to the presentation of the general functioning of the MAS *Amadeus*. After a brief presentation of Amadeus, we focus on the roles of two types of agents composing the system (*controller* and *context* agents), which are in charge of the learning of a good behaviour for each effector of the device. Finally, we introduce the functioning of the *Data*, in charge of the filtering of useless perceived data.

4.1 Architecture of *Amadeus*

Figure 1 shows an instance of *Amadeus* associated to a device. An instance consists of four different types of agents: the *Data* agent dealing with the perceived data, the *User* agent dealing with the user's preferences, and the *Context* and *Controller* agents dealing with the learned behaviour for each effector of the device.

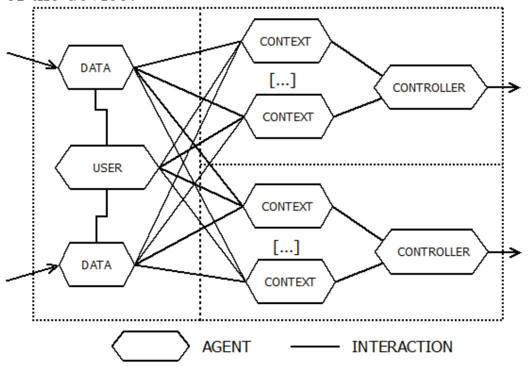


Fig. 1.Representation of an instance of *Amadeus*

The *Data* agents gather all data perceived by the instance in order to be available for the other agents of the instance. These data can result from local sensors of the device, or from other instances of *Amadeus*. Indeed, to describe the context in which one user uses a device, the locally perceived data are not sufficient, so it is necessary for the different instances of *Amadeus* to exchanges data between them.

Each instance possesses a *User* agent that is in charge of the user's preferences: based on a representation of the user's preferences (currently explicitly written in a XML file), the *User* agent can evaluate, for any situation, if the user is satisfied with regard to the state of the effectors of the device. This satisfaction level is represented by a numerical value that varies from 1 (he is very satisfied) to 0 (he is not satisfied at all). One of our main perspectives for this work is to implement a *User* agent able to learn the user's preferences by itself, without any static knowledge. Moreover, the actual solution deals with only one user, but can be applied in a similar way with many users.

Finally, a C*ontroller* agent is associated to each effector of the device. Its goal is to control the state of its effector in order to make the user's satisfaction level as high as possible. For this, it is related to a set of *Context* agents whose aim at proposing actions associated to the forecast of their effect on the user's satisfaction.

4.2 Dynamic learning of a good behaviour

The functioning of a *Controller* agent and its associated *Context* agents are strongly related. If we place ourselves in a more general point of view, these two agents can be seen as a single agent that (i) records all the user's actions and, for each of them, the situation where he/she performed it and (ii) uses these records to decide at runtime, faced to a new situation, the action that seems to be the more appropriate to be done.

Currently, the *Controller* agent decides of the most appropriate action and the set of its *Context* agents represents the recorded previous cases. However, contrary to more classical learning algorithms (such as CBR) where the previous cases are generally recorded as static knowledge, a *Context* agent represents a dynamic knowledge that adapts itself if it becomes erroneous.

The next two paragraphs are devoted to a more detailed presentation of the *Controller* agent and the *Context* agent. Next the joint behavior of these agents is detailed in order to explain how an instance of Amadeus is able to learn and to act on its effectors on behalf of the user.

4.2.1 Controller agent

A *Controller* agent is associated to each effector of a device. Its goal is to evaluate at any time what is the best action to keep the user's satisfaction level as high as possible. We define an action as being the affectation of a state to the effector. So, an action can be "to change the effector state", or "to maintain the actual effector state" (the action is to do nothing).

A *Controller* agent begins every cycle by perceiving the propositions of actions made by the *context* agents. Each proposition of action *PA* includes the action description *A*, associated with a forecast *F* on the effect of this action on the user's satisfaction level *U*, and a confidence *C* about this forecast. So, each action proposition *PA* can be expressed as "if the action *A* is performed now the user's satisfaction level will pass from *U* to *U+F* with a confidence *C*".

The *Controller* agent has then to evaluate the best proposition of action received. However, two or more action propositions may be in conflict. For example, a *Controller* agent can receive two propositions of actions *PA* and *PA'* with the same proposed action *A=A'* but with a different forecast *F≠F'*. In this case, (and in similar cases where more than two propositions of actions are in conflict), the *Controller* agent eliminates the less confident proposition(s) of action(s).

Once this preprocessing realized, the *Controller* agent has propositions of actions, each of them having only one forecast. Based on these forecasts, the *Controller* agent is then able to evaluate the better proposition of action in order to increase as much as possible (or decrease if it is not possible to do better) the user's satisfaction level.

Finally, the *Controller* agent selects the *Context* agent that sent the best proposition of action and unselects at the same time the other previously selected *Context* agents.

Even if the *Controller* agent is the agent in charge of deciding what is the best action to be performed on the effector, we can notice that it does not possess any particular knowledge; all information to take its decision are provided by *Context* agents.

4.2.2 Context agent

A *Context* agent is a representation of an action previously performed by the user in a particular situation. Indeed it considers that, if the same situation happens once again, the realization of its action will have the same effect that the first time the action was performed. A *Context* agent is then created each time the user makes an action, in order to represent this action in the associated situation. It also possesses a description of this action.

The description of its situation is not based on a semantic representation, but only on the states of the perceive data. More precisely, when a *Context* agent is created, it creates a range values around the values of the perceived data, thanks to a data structure named Adaptive Range Tracker (ART). An ART is an interval where borders are managed by two AVT

[LEMOUZY et al., 2011]. An AVT is a tool based on an AMAS that enables to approximate as best as possible the real value of a changing variable. We fixed the initial wide of a range values to 5% of the possible values for a data. Each ART has a validity status: when the actual data value is included in the range of an ART, this ART is considered as *valid*. The *Context* agent has its own validity status, and when all the ART of a *Context* agent are *valid*, this *Context* agent becomes *valid* in its turn. This means it can send its proposition of action to the *Controller* agent.

As explained in section 4.2.1, a proposition of actions consists of the value to affect to the effector, a forecast on the effect of this action on the user's satisfaction and the confidence level of this forecast. The forecast is initialized at the *Context* agent creation, and it is based on the comparison between the user's satisfaction levels before and after he performs his action. The confidence level associated with this forecast is initialized at 0.5. The figure 2 represents the internal state of a *Context* agent, including a values range for each data, its validity status, its selection status, its forecast and the confidence associated to this forecast.

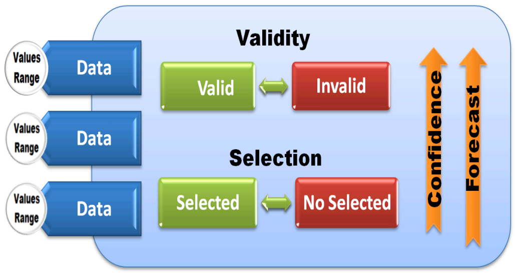


Fig. 2.Representation of the *Context* agent

The *Context* agent begins its cycle by validating its values ranges depending on perceived data updates. Then, if these updates make it *valid*, it sends its proposition of action to the *Controller* agent. If its proposition of action is the most interesting for the *Controller* agent, it is then selected. In this case, the *Context* agent records the current user's satisfaction level as perceived from the *User* agent. A *Context* agent is not a static representation of an action proposition because it has the capabilities to observe the real effect of its proposition on the user's satisfaction level. Indeed, when the *Controller* agent unselects a *Context* agent, the *Context* agent compares the previous and actual user's satisfaction levels, and evaluates if the real effect of its action fits with the reality. If it is the case, the proposition of action sent to the *Context* agent was perfectly correct; so it increases the confidence level of its forecast. When the forecast value differs from the reality, but the forecast meaning is correct (for example, the action increased the user's satisfaction level, but not as much as expected), the *Context* agent adapts its forecast to make it more correct. However, it considers that its action proposition was correct enough to increase the confidence level of its forecast. If the user's satisfaction level has increased in the opposite direction of what it expected (the user's satisfaction level has decreased whereas the *Context* agent has forecasted an increase), the *Context* agent first decreases the confidence level of its forecast. Then it adjusts its values ranges. Indeed it considers that making such an error means that it sent its proposition of action in a wrong situation (maybe the initial values ranges were too wide, or maybe the user's preferences changed).

To summarize, when the *Context* agent is selected by the *Controller* agent, it adjusts its proposition of action. If its proposition was good, it can progressively refine its action proposition: the values ranges enabling it to consider itself as *valid* become then more precise. It is then able to send its proposition of action in more correct situations with a forecast more accurate and a higher confidence. If its proposition was not good, the *Context* agent decreases the possible situations where it can send its proposition, and it decreases the confidence on the forecast on the effect of the proposition of action.

Finally, if the *Context* agent sends too much bad propositions of actions, it can be considered as inappropriate. So, if a *Context* agent possesses a confidence level equal to zero, it decides to disappear.

4.2.3 How Amadeus is learning?

This paragraph aims at explaining how the set of *Context* agents and the *Controller* agent, by working together, can make the instance of *Amadeus* able to learn a good behaviour for each effector of a device.

The first point concerns the *Context* agents' creation. These agents represent knowledge about the user's behaviour, but the *Controller* agent does not have to manage them. Every *Context* agent possesses an autonomous behavior and sends by itself its propositions when it seems appropriate to the actual situation.

Moreover, contrary to classical learning algorithms where it is necessary, at every change in the environment, to start again the learning from the beginning (and so, to record all previous cases), every *Context* agent autonomously observes in which situation its proposition of action becomes inappropriate. For that it evaluates each time it is selected, if its proposition of action has the expected effect; if it is the case, it adapts itself by adjusting its values ranges and by updating its confidence level. Finally, the suppression of a part of the effector behaviour that became incorrect is also chargeable to every *Context* agent.

4.2.4 Results

In order to evaluate the ability of *Amadeus* to learn the user's recurring actions, we established a simulation of a simple example of ambient system. Figure 3 represents an example of an apartment with a light and an electric shutter, as well as a presence sensor and a luminosity sensor.

Fig. 3.Illustration of the apartment of the study

In this simulation, the user moves in and out of the room during the day, looking for a satisfying luminosity. More precisely, if the luminosity is too low when the user is in the room, the user opens the electric shutter, and if it is not enough, he turns on the light. In the same way, with a strong luminosity, he begins to turn off the light, and eventually he closes the shutter. When the user leaves the room, it turns off the light if it was turned on, but does not care if the shutter is opened or closed when he is absent.

Figure 4 represents the number of actions performed by the user during a simulation of fifty days; the user performs on average ten actions by days.

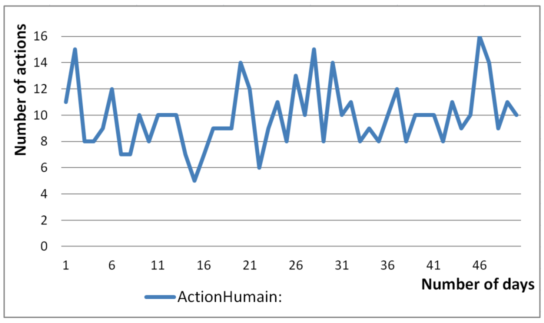


Fig. 4.Number of user's actions by day without *Amadeus*

Then, in a first study, we add an instance of *Amadeus* to every device of the ambient system. So, we have an instance of *Amadeus* in charge of learning the correct behaviour for the light, and another instance for the electric shutter.

Figure 5 represents the user's actions as well as the *Amadeus* actions. We can observe that, the first day, the user performs all the actions. Indeed, *Amadeus* begins its process without any initial knowledge (without any *Context* agent). But after the first day, some *Context* agents are created and begin to send their propositions of actions. So, considering the static user's preferences of this study, *Amadeus* performs most of the actions on behalf of the user. And progressively, as the different *Context* agents accurate their own behaviors, the user's actions go to disappear and be made by *Amadeus*, and the last user's actions allow the creation of other *Context* agents.

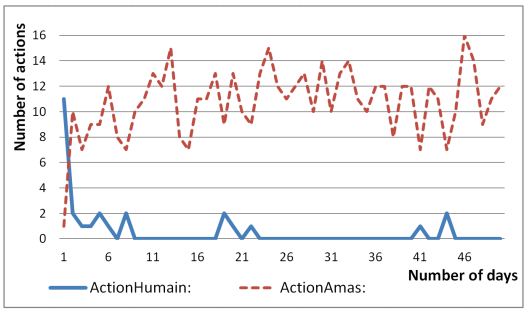


Fig. 5.Number of user's actions by day with *Amadeus*

We can notice that whereas the user performed 495 actions during the 50 days without *Amadeus*, this number decreases until 25, when *Amadeus* is added. This study shows the capability of *Amadeus* to learn a correct behaviour in order to act on behalf of the user (having static preferences).

To study now the capability of *Amadeus* to deal with a change in the user's preferences we realized the same simulation but at the 25th day, we modify the user's preferences: he is now supposed to prefer a lower level of luminosity. He turns on the light or opens the shutter later, whereas he turns off the light or closes the shutter sooner.

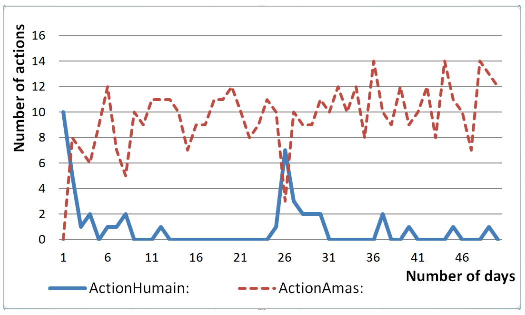


Fig. 6.Number of user's actions by day with a user's preferences change in the 25th day

Figure 6 shows that during the 25th day, the user starts again to act because the action provided by *Amadeus* is not appropriate for him. However, *Amadeus* progressively learns new actions to make its behaviour adapted to the new user's preferences. More precisely, the existing *Context* agents that became inappropriate progressively disappear whereas new *Context* agents, based on the new user's actions, are created.

We can notice many properties for this learning. It is realized exclusively in a in a local way, independently of the other instances learning, and without any initial knowledge. The data themselves are not associated with any *a priori* knowledge about its semantic, the perceived data being processed as numerical data. Moreover, each instance makes its learning at runtime: there are not two different phases with a first phase where the instance only observes what it happens, and a second phase where, after the use of a learning algorithm on the recorded data, the instance uses its learned behaviour to act on the effectors. On the contrary, an instance adapts its learned behaviour without recording perceived data, but only by adapting its learned behaviour when new perceptions arrive. So, any new action of the user can modify or improve the learning at any moment.

4.3 Filtering Data

In section 4.2 we described the ability of an instance of *Amadeus* to associate, for each occurred situation, an action with a forecast on the effect of this action on the user's level satisfaction. However, the main limit of our contribution concerns the number of data that our system has to process. Indeed, it seems easy to describe correctly the situations in which the instance of *Amadeus* can act when all perceived data are useful to describe this situation. Nevertheless, in a real ambient system, considering the large number of devices, we can consider that generally, only a part of the perceived data is really useful and that a lot of perceived data is useless.

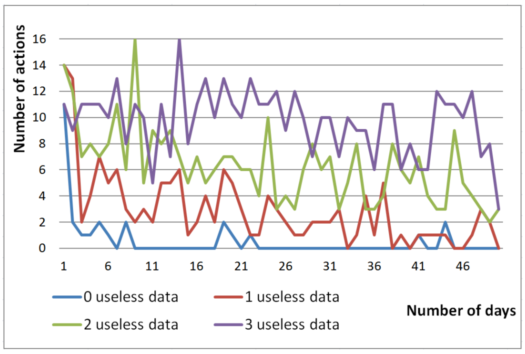
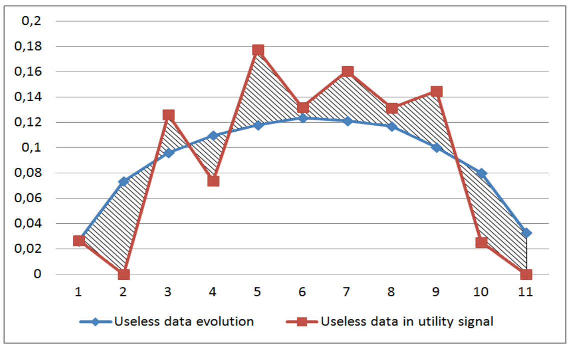


Fig.7. Number of actions performed by *Amadeus* depending on the useless data number

Figure 7 represents the number of *Amadeus* actions depending of the useless data number, the useless data values evolving in a random way. It shows a strong decrease of the *Amadeus's* performances when useless data are added to the ambient system.

To solve this problem, we propose to endow every instance of *Amadeus* with learning capabilities in order to filter at runtime the useless perceived data. Considering our data modeling only based on numerical values, we have to realize this filtering process without any semantic data. Moreover, our objective is to realize this process at runtime, without saving any data.

We propose in that sense to agentify each data. The filtering process is performed by the *Data* agents: each *Data* agent has to learn if its associated data is useless to the *Context* agents related to a *Controller* agent. For that, when an action is performed by the system, if this action is a relevant one, every *Context* agent proposing a different action observes the list of its invalid values ranges. Indeed, if its



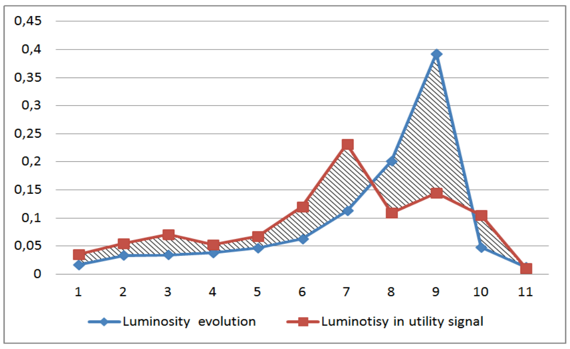


Fig. 8. Density functions for the luminosity and the useless data

proposition of action is different from the performed one, it considers that it was right to not be valid because its action would not have been relevant. So, it considers that all its invalid values ranges helped it to do not wrongly propose its action in the current situation. In this data list, it may exist useless data that are invalid in this situation, but we can be sure that at least one data of the list is useful. So, the *Context* agent sends a *utility signal* to the associated *Data* agent. This signal is in the form "I was invalid partially thanks to you and my invalid status was the right one; you have been useful for me".

To determine the uselessness of a data, we observe a difference between useful and useless data. On the one hand, the state of a useless data is independent of the performed actions, so it can have any value when it is included in a *Context* agent 's *utility signal*. On the other hand, a useful data has a specific value when it is included in a *utility signal*. So, we can distinguish useful and useless data by comparing their values at runtime with their values when they are considered as useful. Figure 8 represents the density functions of two data of the simulation: the luminosity data and a useless data. We can observe that there is a big difference between the two density functions of the luminosity data, because this piece of data is linked to the decisions of realizing actions on the light (figure 8, right part). Conversely, the two density functions of the useless data are very similar, because there is not link between the value of the data and its supposed utility (figure 8, right part).

So, it is possible, by only studying a representation of the density function of a data, to establish a level utility, based on the difference between the two density functions. This difference is calculated thanks to the statistic indicator known as the *Chi-square distance* [MILLOT, 2009].

Figure 9 shows the result of the *Amadeus* learning, once its *Data* agents are able to filter the useless data. In this experiment, we used the same simulation as the one presented in section 4.2.4, but we added three useless data whose values evolve randomly. We can then observe that, during the first days, the useless data prevent *Amadeus* realizing its learning correctly and acting on behalf of the user. However, after few cycles of simulation, *Amadeus* learned what data are useful.

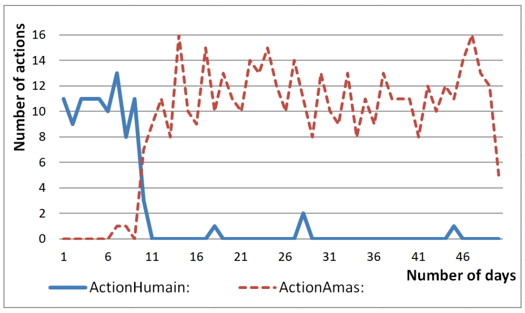


Fig. 9.Number of actions performed by *Amadeus* depending of the useless data number

5 Conclusions

This paper presents the multi-agent system *Amadeus*, which purpose is to assist the user in an ambient system. For this, it perceives the user's actions and learns, from its observations, how and in what situations it has to perform the user's actions on his behalf.

We established three requirements for our system: it has to be generic, distributed, and able to adapt its behaviour to the user's recurrent actions. Amadeus respects these three requirements. Firstly, it is generic as it manages only numerical data without any associated semantic information. Secondly, it is distributed because one instance of *Amadeus* is associated to each device of the ambient system. And finally, we have experimentally shown that it is able to adapt its behaviour to the user's actions, even if the user changes his preferences at runtime.

The main originality of this proposal is that *Amadeus* has been designed in a generic way that is independently from any kind of ambient system. The main assumption of *Amadeus* is that the "good" or "correct" behavior can be obtained from user's actions. Even if this assumption could be discussed, it is very relevant in repeated everyday life situations. Finally, the behavior of the system is learned directly at runtime, without classical learning prior phases. So, *Amadeus* is able to deal with evolving behaviour of the user, such as preferences changes.

Even if experiments are encouraging, some future works can be sketched. On the one hand, the current experiments, done on many simulations, have to be extended in more realistic situations such as apartments containing many rooms and/or many users. On the other hand, the learning process and the useless data filtering process, have to be enhanced in more general cases dealing with a great number of data.

6 References

[CABRI et al., 2005]	G. Cabri, L. Ferrari, L. Leonardi, and F. Zambonelli. *The laica project: Supporting ambient intelligence via agents and ad-hoc middleware* Enabling Technologies: Infrastructure for Collaborative Enterprise. 14th IEEE International Workshops on, IEEE, 2005
[CONSTANTINI et al., 2008]	S. Costantini, L. Mostarda, A. Tocchio, and P. Tsintza. *Dalica: Agent-based ambient intelligence for cultural-heritage scenarios* Intelligent Systems, IEEE, 23(2),2008
[DUJARDIN et al., 2011]	T. Dujardin, J. Rouillard, J.C. Routier, J.C. Tarby, et al. *Gestion intelligente d'un contexte domotique par un système multi-agents* Actes Journées Francophones sur les Systèmes Multi-Agents, 2011
[GEORGE et al., 2011]	Jean-Pierre Georgée, Marie-Pierre Gleizes, and Valérie Camps. *Cooperation* In Giovanna Di Marzo Serugendo, Marie-Pierre Gleizes, and Anthony Karageorgos, editors, Self-organising Software, Natural Computing Series,. Springer Berlin Heidelberg, 2011
[LECH et al., 2005]	T.C. Lech and L.W.M.Wienhofen. *AmbieAgents: a scalable infrastructure for mobile and context-aware information services* In Proceedings of the fourth international joint conference on Autonomous agents and multi-agent systems, ACM, 2005
[LEMOUZY et al., 2011]	S. Lemouzy, V. Camps and P. Glize. *Principles and Properties of a MAS Learning Algorithm: a Comparison with Standard Learning Algorithms Applied to Implicit Feedback Assessment* IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT 2011), Lyon, France, 2011.
[MILLOT, 2009]	Millot, G. *Comprendre et réaliser les tests statistiques à l'aide de R* Boeck université, Louvain-la-Neuve, Belgique, 1st edition, 2009.
[RUSSEL et al., 1995]	S.J. Russell, P. Norvig, J.F. Canny, J.M. Malik, and D.D. Edwards. *Articial intelligence: a modern approach* Prentice hall Englewood Cliffs, NJ, 1995
[SATOH et al., 2004]	I. Satoh. *Mobile agents for ambient intelligence* Proceedings of Massively Multi-Agent Systems, first international workshop MMAS Kyoto, Japan, December 2004
[SPANOUDAKIS et al., 2006]	N. Spanoudakis and P. Moraitis. *Agent based architecture in an ambient intelligence context* Proceedings of the 4th European Workshop on Multi-Agent Systems (EUMAS'06), Lisbon, Portugal 2006.
[TAPIA et al., 2008]	D.I. Tapia, J. Bajo, J.M. Sanchez, and J.M. Corchado. *An ambient intelligence based multi-agent architecture* Developing Ambient Intelligence, 2008.