Learning user's behaviour with an Adaptive Multi-Agent System approach

Valérian Guivarch[1], Juan Francisco De Paz Santana[2], Javier Bajo Pérez[2], André Péninou[1], and Valérie Camps[1]

[1] Institut de Recherche en Informatique de Toulouse, University of Toulouse, France
`Firstname.Lastname@irit.fr`
[2] Department of Computer Science and Automation, University of Salamanca, Spain* {`fcofds,jbajope`}`@usal.es`

Abstract. The context-aware systems allow the adaptation of the environment, which depends on the values perceived by sensors and on the users' preferences detected from users behaviour. These system need to adapt at the users' preferences change so as to generate dynamic systems that improve their models while they are running. This work proposes a multi-agent system which allows to learn and to anticipate users' behaviour and to adapt at the change into a dynamic mode, and where storing information from the previous cases during successive execution is no more compulsory.

Keywords: multi-agent systems, classifiers, ambient intelligence

1 Introduction

Introduced by Weiser [16], ambient computing consists rather in integrating a computer system directly into the real world, than limiting it to a single computer. Such a computer system is then distributed into the environment in the form of physical devices with which the users can interact, depending on their perceptions and location. These systems are subject to a strong dynamic, with the appearance or the disappearance of many heterogeneous devices (such as measuring the users' displacement) at run-time. However, the user may be faced with situations that the developer cannot foresee. Thus, making an exhaustive list of all the situations that the system may face is not possible during the design phase.

The use of classifiers is widespread for this type of study. However, it is not so easy to adapt to users' behaviour changes. Among the traditional classifiers, there are based on decision tree [13], probabilistic models [6], function-based algorithms [14] or fuzzy models [1]. Their functioning is efficient enough, while an adjustment is not required by a new user's behaviour. Otherwise, it becomes necessary to adapt their learning to the new cases.

* This work has been supported by the MICINN TIN 2009-13839-C03-03.

This paper focuses on the learning of users' behaviour in order to make such ambient systems able to adapt to the user context. To do so, we consider the different suitable classification algorithms so as to associate any situation with the action expected by the user. The algorithms need to remember the previous cases in order to adapt to any change in their environment, such as a change of the users' preferences or the appearance/disappearance of the devices. In this way, they can reboot. In this way, they can reboot the learning from the beginning if the learning seems too much deteriorated. We are looking for a new solution which can perform this dynamic learning and which does not involve the record of any previous cases but a continuous adaptation, depending on its perception. The system is compared with different classifiers in order to evaluate its performance.

To that end, we present the approach of Adaptive Multi-Agent System (AMAS) [8], and the system *Amadeus*, designed through this approach. We begin to describe its architecture and functioning, and then we will put forward a comparative study between *Amadeus* and more conventional algorithms.

This article is divided as follows: section two describes the state of art relating to the classifier systems and to the Adaptive Multi-Agent System approach; section three presents the proposed model; section four describes respectively the results obtained and the conclusions.

2 State of the art

2.1 Classifiers

The classification algorithms include decision trees, decision rules, probabilistic models, fuzzy models, function-based algorithms and ensemble.

These algorithms/this algorithm use different processes to make the classification. The most commons ones are decision rules and trees which make possible the creation of rules that concatenate variables through operators in order to classify a determined case. Among the classical decision rules algorithms, there are RIPPER [5], One-R [10], M5 [9]. Concerning the decision tree algorithm, the most used are C4.5 or J48 [13], (those are the same algorithms, but J48 is the java implementation), and CART [3] [1] (Classification and Regression Trees). The most commons probabilistic models are Bayes [6] and the bayesian network [2], but this type of models are not very efficient for continuous variables because it is necessary to realize a discretization of the values. The fuzzy models such as K-NN (K-Nearest Neighbours) are not always effective because they require the calculation of several measures of distances, so the performance would not be good enough. An alternative algorithm which is actually very used, is the definition of functions as the Support Vector Machine (SVM) [14], this process is used to give good results with a good yield because the classification of new instances is low although the construction of the model is time-consuming. Other options are the ensembles such as Bagging [4] and Ada-Boosting [7].

2.2 Adaptive Multi-Agent System

Adaptive Multi-Agent Systems (AMAS) [8] are based on a local approach to design complex systems for which no solution is *a priori* known. This approach focuses on defining the local behaviour of agents to be truly adaptive, while ignoring the purpose of the overall system, but ensuring that the collective behaviour is the one expected, ie the system is "functionally adequate". To this end, agents must have a local cooperative behaviour. Cooperation is not limited by a simple resource sharing or simply working together, it is grounded on three local meta-rules, applied by every agent that the designer must instantiate depending on the problem to solve: *(c_{per})* : any signal received by an agent must be unambiguously understood; *(c_{dec})* : any information from its perceptions should be useful to its reasoning, *(c_{act})* : its reasoning should make it perform actions which are useful for the others and for the environment.

The AMAS approach can be described as proscriptive because agents must overall anticipate, prevent or repair Non Cooperative Situations (NCS). A NCS appears when at least one of the three previous meta-rules is not locally verified by an agent. Several generic NCS can then be highlighted: incomprehension and ambiguity if *(c_{per})* is not verified, incompetence and unproductivity if *(c_{dec})* is not respected and, finally, concurrency, conflict and uselessness where *(c_{act})* is not verified. This approach has important methodological implications: designing an AMAS is equivalent to defining and assigning rules of cooperation agents. In particular, for a given problem, the designer must 1) define the nominal behaviour of an agent, then 2) deduct the NCS which the agent may face with, and finally 3) define the actions that the agent carries out to come back in a cooperative state. This approach has helped solve several types of problems related to different areas: the real-time profiling [11], the bioprocesses control [15], etc. The capacity of the AMAS to resolve complex, dynamic and distributed problems make them ideally suited to resolve the problem of user behaviour learning in ambient system.

3 Proposed System

Based on the approach by AMAS, we designed *Amadeus*, a multi-agent system whose purpose is, for each device of the ambient system, to collect its local context and to adapt its behaviour depending of its context. For this, *Amadeus* begins with the observation and the learning of the way the user uses each device and in what context, in order to achieve any recurring action in its place. Any new user action can modify or improve this learning because the latter is continuous during system operation.

An instance of *Amadeus* is associated with each device (Figure 1). In each instance, data agents represent contextual data which are either collected from local sensors of the device or received from other instances. The user agent represents the user satisfaction depending of the user profile the contextual data of the instance. This satisfaction is defined as a value in the range [0, 1]. We present here the operation of context and *controller* agents, which ensure the device adaptation in order to maintain the high user satisfaction.

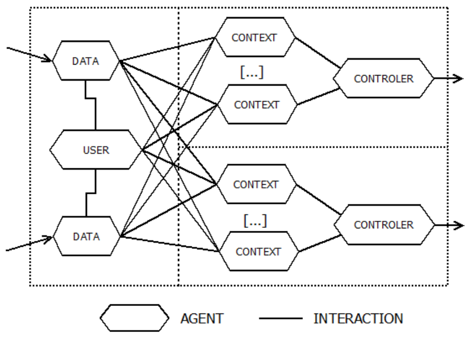

Fig. 1. General structure of the system

General Operation A *controller* agent is associated with each effector of the device. It aims to anticipate the user actions. For this, it has a set of *context* agents, which provides information on the actions it can perform and the consequences of these actions on the user's satisfaction. These *context* agents aim then to determine the effects of a particular action on the user's satisfaction, but also to identify situations where these predictions are correct. In these situations, they view themselves as being valid.

Controller agent The behaviour of *controller* agent is described trough a cycle of three phases. In the perception phase, it collects action suggestions from *context* agents. Each suggestion contains a description of the proposed action and a forecast of the impact that this action will have on the user's satisfaction, as well as a confidence level for this forecast.

In the decision phase, the *controller* agent evaluates what is the most advantageous action to make among the proposed ones in order to increase the user's satisfaction. To that end, it begins its treatment by managing conflicts or concurrences which oppose the suggestions. For example, if two *context* agents propose the same action with different forecasts, the *context* agent will ignore the less confident suggestions. Then, once in possession of pertinent suggestions, the *controller* agent evaluates what is the best action to do among those proposed by the remaining *context* agents. In other words, it assesses, based on the current situation, which forecast ensures the higher level of user satisfaction. This action may equally be a change in the status of an effector or the maintaining of this effector in its current state, because a *context* agent can propose “preserve the current state” as action.

Finally, in the action phase, it selects the *context* agent associated with the best action, and it makes this action (i.e.: it affects the new state at the effector).

Context agent A *context* agent has, for each contextual data in the input system, a values' range representing the validity range of the *context* agent. A value range is called valid if the current value of this data is included within the bounds of the values range. The *context* agent itself has a validity status, which is valid when all its data values' ranges are valid, and invalid in the other cases. The *context* agent starts its cycle by validating or invalidating its value ranges, depending on the updated values perceived. Then he does the same with its validity status, if appropriate. The bounds of the values ranges is implemented through an Adaptive Value Tracker (AVT) [12], which is a software component to find the value of a dynamic variable in a given space through successive feedbacks.

A *context* agent owns an action suggestion which is sent to the *controller* agent when it becomes valid. This suggestion includes the description of the action itself, which means the state to allocate at the effector. As well, it includes a forecast about the impact of this action on the user satisfaction, which is implemented as a value between -1 and 1. Finally, it includes a confidence about the forecast, implemented with an AVT limited between 0 and 1, and which means how much the *context* agent is certain that the proposed action will have the expected impact. When a *context* agent becomes valid, it communicates to its associated *controller* agent in order to send it its action suggestion. As a matter of fact, a *context* agent considers that the forecast which accompanies the proposed action is true only when the *context* agent is valid. The *context* agent also has a selection state. It can be selected or unselected. As I said, a *context* agent is selected if its suggestion seems to be the best for the *controller* agent. If another *context* agent is then selected, the previous selected *context* agent becomes automatically unselected. If the *context* agent is selected, it records the current level of user satisfaction. Once it is no longer selected, it makes the comparison of the effect produced on the user satisfaction with his prediction possible. Three cases can occur:

1. Forecasts match perfectly to reality: the *context* agent increases his confidence.
2. The forecast value differs from reality, but the forecast meaning is correct: the *context* agent adapts its prediction to rectify it, but nevertheless considered that the action was good enough to increase its confidence.
3. The forecast meaning is incorrect: the *context* agent first decreases confidence. However, it considers that making such a mistake means that it should not have sent this suggestion in the situation it was in. As a consequence, it modifies its input ranges so that if the same situation occurs it remains invalid.

The behaviour of a *context* agent is to propose an action for a given context with a certain forecast about the consequences of this action on the user satisfaction. Besides, it evaluates its confidence as the reliability of its suggestion. In other words, more the confidence is high, more the agent has the certainty that its proposed action will have the expected consequence. This confidence is calculated by a function that evaluates a confidence level T_{t+1} at time $_{t+1}$ based on its confidence level T_t to the previous time, a feedback F included between 0 and 1, and a parameter λ determined experimentally and which represents the impact of feedback in the calculation of the new confidence level. To increase the confidence of the *context* agent in cases 1 and 2 described above, we use a feedback close to 1, while we use a feedback close to 0 to decrease it in the third case. This function is as following:

$$T_{t+1} = T_t * \lambda + F * (1 - \lambda)$$

A *context* agent is created each time the user performs an action when no *context* agents proposed this action for this situation (including the action to "do not change the state of the effector"). This new *context* agent has initialized with this action, and its values ranges are initialized around actual values data. The forecast is based on the observed impact of these actions the first time, and the confidence is initialized at 0.5.

Finally, the learning of user behaviour is based on the interaction of *context* and *controller* agents. It is produced through the creation of *context* agents, the adaptation of values ranges and the adjusting of *context* agent confidence.

4 Case study

We made this study in order to compare the performances of *Amadeus* with more conventional learning algorithms. More specifically, we focus on the classifier algorithms Naive Bayes, J48, SVM and LMT, which are well-know techniques for this type of problems. We made this study in order to test the *Amadeus* capacity to determine, for a given situation, what the user action will be. Each situation can be considered as a case, as defined by the Case Base Reasoning approach, or as a contextual situation, as defined in *Amadeus*.

In order to make this evaluation, we designed a simulator of ambient system which makes possible the description of simulated users' simple behaviours in a virtual ambient. We define our ambient system as an apartment where we use a lamp and a shutter, both electric, in the living room. In addition to that, we add on the same room a luminosity and presence sensor.

As for the user, the simulator enables to describe the virtual users' behaviour. For now, we set the maximal numbers of users to a single user; the multi-users problems will be dealt with in a future study. We design this virtual user with a simple behaviour: he walks randomly in the apartment, making sure that the brightness is suitable when he is in the living room, but he remains energy efficient. In practice, this means that when he is in the living room, if it is too dark he will first try to illuminate the room by opening the shutter, then light up the lamp if it is not enough light. If the brightness is too high, he will first turn the lamp off if it is on, otherwise he will close the shutter. On the other hand, when he is not in the living room, he does not care about the shutter's state, but he makes sure that the lamp was efficiently turned on. This behaviour is much more than maintaining a constant average brightness in the room, because it takes into account the presence of the user in the living room, and the priority to open the shutter rather than to switch the lamp on in order to ensure that the energy is saved if it is possible. As for the user's movements, a random displacement enables to change when and how long the user is or is not present in the living room. To improve the realism of the simulation, the conditions associated to the user's actions are unclear; for example, if the theoretical condition where the user turns on the light is a luminosity above 55, then the real condition will be randomly fixed between 50 and 60. As a matter of fact, a human behaviour cannot be modeled with too precise rules of actions.

In order to compare the result produced by *Amadeus* with the result of others classifier algorithms, we generate 13 days of data. The simulator generates a simulation where the time is accelerated, with 1440 simulation cycles by virtual day, one by minute.

The first days (from 1 to 3) are allocated to the learning phase and the next 10 days are allocated to the tests on the effects of the learning. We perform this experiment in two different conditions: in the first one, the learning phase lasts only one day, and over this period we give only the first day data to various algorithms so that they make their learning; in the second condition, 3 days are used for the learning phase. Therefore, we can compare the impact of the learning time on the different systems.

In the figure 1, we can see an example of data perceived by the different learning algorithms. We can see the date from the cycle 480 until the cycle 490 of the first day (i.e. from 8:00 till 8:10 of the first day). At this moment, the user is in the living room (Presence=1), the shutter is open (Shutter=1), the light is turned on (Light=1). The user behaviour sets the max threshold of luminosity at 90, and the more the luminosity is beyond the threshold, the more the user's satisfaction decreases. As I previously explained, the threshold, which is set so that the user acts, is randomly generated in order to improve the simulation realism. Here, the generated threshold is at 26%. As a matter of fact, we can see in the cycle 485 that, whereas his satisfaction decreases until this value, the user decides to turn the light off. (User Decision=0).

The objective of our learning is to see, according to these data, what the user's decision is in order to be able to anticipate its decision. Note that the number of the day and the number of the cycle, if they are used to save the previous problem on the conventional algorithm, are not directly used to make the learning of the behaviour.

5 Results and conclusions

In order to verify *Amadeus*' adaptability, we test the system evolution through 18 720 cases (one case by minute during 13 days). We capture the data which refer to these cases and we test the success rate when new case arrives in the system. In order to verify the real adaptability from the initial step on, a validation is performed after the capture of information from the first day, in which there were 1440 pieces of data. A learning is performed, using the information of the

Day	Cycle	User Satisfaction	Luminosity	Presence	Shutter	Light	User Decision
0	480	34%	91,97	1	1	1	1
0	481	32%	92,06	1	1	1	1
0	482	31%	92,16	1	1	1	1
0	483	29%	92,26	1	1	1	1
0	484	27%	92,35	1	1	1	1
0	485	26%	92,45	1	1	1	0
0	486	100%	74,03	1	1	0	0
0	487	100%	74,11	1	1	0	0
0	488	100%	74,18	1	1	0	0
0	489	100%	74,26	1	1	0	0
0	490	100%	74,33	1	1	0	0

Table 1. Example of data, as perceived by the different learning algorithms.

first day, then as new case are coming, the system foresees the associated user's action. We perform a new training of the system only when the classification of a new element is wrong in order to integrate the new case at the used models in the classifiers.

In order to ensure that the system has adaptation and evolution capacities, we use two other variables as the user preferences change, they represent the number of the day and the case number (which is daily reset to zero). If the user preferences change, these variables refer to the case information and the system would be able to adapt itself.

In order to conduct this analysis, we study the system evolution by comparing the multi-agent system *Amadeus* with 4 different classification systems: the J48, the Naive Bayes, SVM and LMT. In the figure 2, we can see the system evolution for the estimation of the light state depending on variables defined in the case study. The axis x represents the case number, and the axis y is the number of accumulated errors. We can see that most of the algorithms have a similar behaviour until the moment where the multi-agent system becomes worst than the others. The method with high performance is LMT but the execution time is too high.

In the second experiment, we repeat the same process with 3 days as learning basis. We use the same classifiers and we obtain the results represented on figure 3. The figure 2 shows the successful rate of the SMA is better than the other methods until the user adapts a new behaviour, then the successful rate of the LMT is better than the multi-agent system but this algorithm is not efficient and the execution time is too high. The execution time to carry out the test was 5177.89 seconds and the execution time of the others methods was lower than 60 seconds.

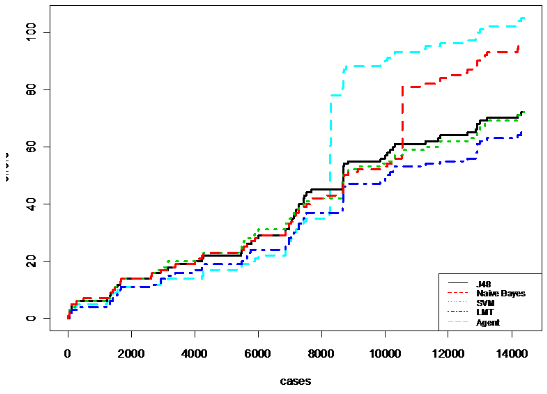

Fig. 2. Evolution of the number of errors made by the CBR with learning based on one day.

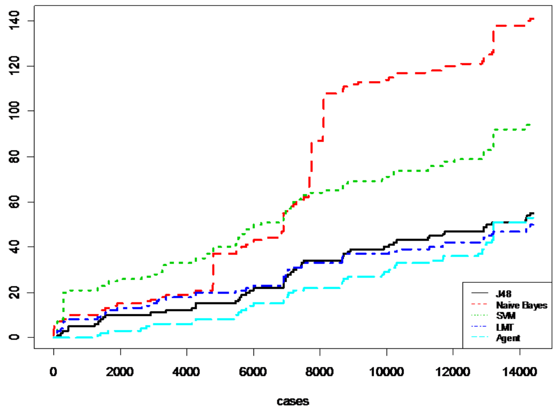

Fig. 3. Evolution of the number of errors made by the CBR with learning based on three days.

The multi-agent system has initially less generalization ability than the others classifiers, but better performances when new cases come. Moreover, *Amadeus* is able to adapt itself at new situations without it is necessary to store these new cases, facilitating the management of the necessary data for the correct functioning of the system.

References

1. D.W. Aha, D. Kibler, and M.K. Albert. Instance-based learning algorithms. *Machine learning*, 6:37–66, 1991.
2. R.R. Bouckaert. *Bayesian belief networks: from construction to inference.* Universiteit Utrecht, Faculteit Wiskunde en Informatica, 1995.
3. L. Breiman. *Classification and regression trees.* Chapman & Hall/CRC, 1984.
4. L. Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.
5. W.W. Cohen. Fast effective rule induction. *Proceedings of the Twelfth International Conference on Machine Learning, Lake Tahoe, California*, 1995.
6. R.O. Duda and P.E. Hart. *Pattern classification and scene analysis.* Wiley, 1996.
7. Y. Freund and R.E. Schapire. Experiments with a new boosting algorithm. In *Thirteenth International Conference on Machine Learning*, pages 148–156, 1996.
8. Jean-Pierre Georgé, 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, pages 193–226. Springer Berlin Heidelberg, 2011.
9. G. Holmes, M. Hall, and E. Prank. Generating rule sets from model trees. *Advanced Topics in Artificial Intelligence*, pages 1–12, 1999.
10. R.C. Holte. Very simple classification rules perform well on most commonly used datasets. *Machine learning*, 11(1):63–90, 1993.
11. S. Lemouzy, V. Camps, and P. Glize. Real time learning of behaviour features for personalised interest assessment. *Advances in Practical Applications of Agents and Multiagent Systems*, pages 5–14, 2010.
12. Sylvain Lemouzy. *Systèmes interactifs auto-adaptatifs par systèmes multi-agents auto-organisateurs: application à la personnalisation de l'accès à l'information.* Thèse de doctorat, Université Paul Sabatier, Toulouse, France, juillet 2011.
13. J.R. Quinlan. *C4. 5: programs for machine learning.* Morgan kaufmann, 1993.
14. V. Vapnik. Pattern recognition using generalized portrait method. *Automation and Remote Control*, 24:774–780, 1963.
15. S. Videau, C. Bernon, P. Glize, and J.L. Uribelarrea. Controlling bioprocesses using cooperative self-organizing agents. *Advances on Practical Applications of Agents and Multiagent Systems*, pages 141–150, 2011.
16. M. Weiser. The computer for the 21st century. *Scientific American*, 265(3):94–104, 1991.