Available online at www.sciencedirect.com
SCIENCE DIRECT
Knowledge-Based Systems 19 (2006) 43–49
Knowledge-Based SYSTEMS
www.elsevier.com/locate/knosys

Personalizing user–agent interaction

Silvia Schiaffino [a,b], Analia Amandi [a,b,*]

[a] ISISTAN, Universidad Nacional del Centro Pcia. Bs. As., Campus Universitario, Paraje Arroyo Seco, Tandil 7000, Argentina
[b] CONICET, Comisión Nacional de Investigaciones Científicas y Técnicas, Buenos Aires Argentina

Received 16 July 2003; accepted 1 July 2005
Available online 10 November 2005

Abstract

Interface agents are computer programs that provide personalized assistance to users with their computer-based tasks. The interface agents developed so far have focused their attention on learning a user's preferences in a given application domain and on assisting him according to them. However, in order to personalize the interaction with users, interface agents should also learn how to best interact with each user and how to provide them assistance of the right sort at the right time. To fulfil this goal, an interface agent has to discover when the user wants a suggestion to solve a problem or deal with a given situation, when he requires only a warning about it and when he does not need any assistance at all. In this work, we propose a learning algorithm, named WoS, to tackle this problem. Our algorithm is based on the observation of a user's actions and on a user's reactions to the agent's assistance actions. The *WoS* algorithm enables an interface agent to adapt its behavior and its interaction with a user to the user's assistance requirements in each particular context.

Keywords: Interface agents; User profiling; Personalization

1. Introduction

Interface agents have become a technology widely used to provide personalized assistance to users with their computer-based tasks. Interface agents are computer programs that have the ability to learn a user's preferences and working habits, and help him to deal with one or more computer applications, improving in this way the user's productivity.

A commonly used metaphor for understanding interface agent paradigm is comparing them to a human secretary or personal assistant who is collaborating with the user in the same work environment [11]. Initially, a personal assistant is not very familiar with the habits and preferences of her employer and may not be very helpful.[1] However, the assistant becomes gradually more effective and competent as she acquires knowledge about him. This knowledge can be acquired by observing how the employer performs tasks, by asking the employer for information, by receiving explicit instructions and feedback from him, or by learning from other assistants' experience. Then, the assistant can perform tasks that were initially performed by the employer, she can suggest him the execution of tasks, she can notify the employer about situations interesting for him and warn him about problems that may arise. In the same way, an interface agent can become more competent as it interacts with a user and learns about him.

The many interface agents that have been developed have focused their attention on learning a user's preferences and working habits in a given application domain and on assisting the user according to them. However, interface agent developers have paid little attention to some key issues when assisting a user: how to best interact with each user and how to provide them assistance of the right sort at the right time.

In a mixed-initiative interaction context like the one we are considering, both user and agent can initiate a dialogue [6]. At any time, one of them might have the initiative while the other works to assist him, contributing to the interaction as required. At other times, the roles are reversed; and at other times again the participants might be working independently, interacting with each other only when specifically asked.

In this context, Horvitz [7] has pointed out some problems with the use of interface agents: poor guessing about the goals and needs of users, inadequate consideration of the costs and benefits of each agent action, poor timing of agent actions and inadequate attention to opportunities that allow a user to guide

* Corresponding author. Address: ISISTAN, Universidad Nacional del Centro Pcia. Bs. As., Campus Universitario, Paraje Arroyo Seco, Tandil 7000, Argentina.
E-mail addresses: sschia@exa.unicen.edu.ar (S. Schiaffino), amandi@exa.unicen.edu.ar (A. Amandi).

[1] For simplicity, we use 'he' for the boss and 'she' for the assistant, but we do not mean to be sexist.

0950-7051/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.knosys.2005.07.005

the invocation of agent services and to refine potentially sub-optimal results provided by the agent.

Among these problems, we have focused our attention on the interaction between users and interface agents, and we have identified several problems that have to be solved for this interaction to succeed. In this work we will address one of these problems. When an interface agent detects a problem situation or a situation in which the user might need assistance, it has to decide among warning the user about the problem and let him decide how to deal with it, suggesting him how to solve it or doing nothing. This decision will be mainly influenced by the knowledge the agent has to make a suggestion, since if it does not know what to suggest it will merely warn the user about the problem or situation. However, although the agent probably knows what to suggest, it has to learn whether the user wants it to suggest him something or not in that particular situation, or if he does not even want to be warned. In order to take the best decision, the agent must discover which assistance action the user expects in each situation that may arise.

In this work, we propose a learning algorithm, named *WoS*, to tackle the problem presented before. The goal of the algorithm is learning when to make a suggestion to a user, when to make him a warning and when to do just nothing, personalizing in this way the interaction with each user. Our algorithm is based on the observation of a user's actions, particularly on a user's reactions to the agent's actions. *WoS* enables an interface agent to adapt its behavior and its interaction with a user to the user's assistance requirements.

This work is organized as follows. Section 2 describes how an agent chooses among several assistance actions. Section 3 presents our proposed learning algorithm. Section 4 reports some experimental results. Section 5 describes some related work. Finally, Section 6 presents our conclusions and future work.

2. Decision making

When an agent is about to assist a user, it has to decide which of several possible actions to execute. An interface agent has knowledge about the application domain, about the user (in the user profile), about itself and about other agents (possibly humans) that inhabit its environment. When a situation of interest or a problem arises, the agent uses this knowledge to decide whether to do something to assist the user or not, and what to do if it decides to do something.

Several algorithms have been used to decide what an agent should do next [4,5,7,9,11]. In the most common approach, agents have three possibilities when they want to assist a user: executing a task autonomously, suggesting the user what to do, and doing nothing. These agents use two threshold values to take decisions, which are established by the user to control the agent's behavior: *do-it* threshold and *tell-me* threshold. If the confidence value associated with an agent action is smaller than the *tell-me* threshold the agent does nothing; if the confidence value is greater than the *tell-me* threshold but smaller than the *do-it* threshold, the agent makes a suggestion to the user; and if the confidence value is greater than the *do-it* threshold the agent executes the task autonomously on the user's behalf.

The confidence-based approach has several problems. The selection of an assistance action is centered on the calculus of confidence values associated with these actions. The confidence associated with a suggestion, for example, indicates the amount of times the agent made the suggestion successfully, over the total amount of times the agent made this suggestion. This calculus does not take into account whether the user wanted that type of assistance or if he preferred another one, a warning for example. Besides, thresholds are generally established by the user or by the agent developer and they have fixed values. Thus, if the agent wants to modify them according to the user's behavior, it cannot do it. Finally, the same thresholds are used for every agent action and every problem situation or situation of interest. This generalization can lead to inappropriate agent behavior since the agent is not supposed to act in the same way in every situation.

Our approach improves this decision-making process by taking into account the user's requirements regarding the agent's assistance actions. Thus, when the agent has to decide among various actions it will not only take into account the confidence values associated with them, but also how the user wants to be assisted in the particular situation the agent is dealing with.

3. WoS learning algorithm

The goal of our algorithm is learning in which contexts the user prefers a suggestion to solve the problem or deal with a situation, when he wants only a warning and when the user does not want any assistance at all. The *WoS* algorithm enables an interface agent to choose the action that is the most acceptable to the user in that particular instance of a given situation.

The input for our learning algorithm is a set of user–agent (or agent–user) interaction experiences. An interaction experience $Ex = <Sit, A, UF, E>$ is described by a situation *Sit*, by the assistance action *A* the agent executes to deal with the problem, by the user feedback *UF* obtained after assisting the user and (sometimes) by an evaluation *E* of the assistance experience (success, failure or undefined). Each situation *Sit* is described by a set of features and the values these features take, $Sit = \{(feature_i, value_i)\}$. An assistance action may be a suggestion, a warning or no action. A suggestion is described by the action suggested by the agent, the problem originating it and a justification of the proposed solution. Similarly, a warning is described by the problem situation the user is being warned about. The user feedback may be implicit if the agent has to obtain it from the user's actions (this occurs in most cases), or explicit if the user explicitly evaluates the agent's actions. If the agent has enough information, the experience can be evaluated as a success or as a failure. If it does not have enough information (e.g. no user feedback at all), the evaluation is not available.

For example, considering an agenda agent, an assistance experience could be the following. The user is scheduling

a new event: a meeting to discuss the evolution of project A with his employees Johnson and Dean. The event is being scheduled for Saturday at 10 a.m. at the user's office. The agent decides to warn the user that Mr. Dean will probably disagree about the meeting date and time because he never schedules meetings on Saturdays. In reply to this warning, the user asks the agent to suggest him another date for the event. In this example, the different parts of the assistance experience are:

- Sit = {(type, new-event), (event-type, meeting), (organizer, user), (participants, [Johnson,Dean]), (topic, project A evolution), (date, Saturday), (time, 10 a.m.), (place, user's office)}
- A = {(type, warning), (message, Dean no meetings on Saturdays)}
- UF = {(type, explicit), (action, asks for a suggestion)}
- E = {(type, failure), (certainty, 1.00)} (suggestion instead of warning)

The *WoS* algorithm uses the information of several user–agent interactions to formulate a set of hypotheses about the user's assistance requirements. These hypotheses may adopt one of the following forms: 'in situation *Sit* the user requires a warning *W*', 'in situation *Sit* the agent requires a suggestion *S* (with justification *J*)' or 'the user does not want assistance (in situation *Sit*)'. Each hypothesis has a certainty degree associated with it that indicates how sure the agent is about it.

After formulating a hypothesis, the algorithm has to validate it according to the amount of positive evidence that supports the hypothesis and the amount of negative evidence that rejects it: If a hypothesis is highly supported, it is turned into a fact that establishes the assistance action that the user requires in a given problem situation. These facts are periodically revised in order to verify if they are still valid or not. Algorithm 1 shows the main steps our algorithm involves. The following sections explain how we perform each of them.

Algorithm 1 WoS Overview

Input: A set *Ex* of user–agent interaction experiences $Ex_i = <Sit_i, A_i, UF_i, E_i>$ (where *Sit*: situation description, *A*: assistance action, *UF*: user feedback, *E*: evaluation)

Ouput: A set *F* of facts and a set *H* of hypotheses about the user's assistance requirements

1: Formulate a set of hypotheses *H* from *Ex* and compute certainty degrees
2: Validate hypotheses in *H* according to positive and negative evidence
3: Turn validated hypotheses into a set of facts *F*
4: Revise facts in *F*

3.1. Formulating hypotheses: using association rules

In our context, a hypothesis expresses the agent's belief that the user requires a certain type of assistance in a given situation. The agent formulates a hypothesis by analyzing the information it has obtained by observing the user's behavior during several assistance experiences. A hypothesis expresses that whenever situation *Sit* occurs, the user will require action *A* (*W*, *S* or *N*) with a certainty degree of *Cer*(*A*). Then, if this certainty degree is high enough the agent will execute action *A* when *Sit* occurs.

To generate hypotheses from a set of user–agent interaction experiences we use Association Rules. Association rules enable us to rapidly discover associations between different situations and the expected assistance actions. Our algorithm calls an association rule-mining module and then processes the discovered rules to formulate hypotheses. An association rule is a rule, which implies certain association relationships among a set of objects (such as they occur together or one implies the other) in a database [1]. Association discovery finds rules about items that appear together in an event (called transactions), such as a purchase transaction or a user–agent interaction experience. Given a set of transactions, where each transaction is a set of literals (called items), an association rule is an expression of the form $X \rightarrow Y$, where *X* and *Y* are two sets of attributes (called itemsets) in the given database. *X* is the antecedent of the rule and *Y* is the consequent.

Given a transaction database *D*, the problem of mining association rules is to find all association rules that satisfy: minimum support (called *minsup*) and minimum confidence (called *minconf*). The support of a rule $X \rightarrow Y$ is the probability of attributes (or attribute sets) *X* and *Y* occurring together in the same transaction. If there are *n* total transactions in the database, and *X* and *Y* occur together in *m* of them, then the support of the rule $X \rightarrow Y$ is *m*/*n*. The confidence of rule $X \rightarrow Y$ is defined as the probability of occurrence of *X* and *Y* together in all transactions in which *X* already occurs. If there are *t* transactions in which *X* occurs, and in exactly *s* of them *X* and *Y* occur together, then the confidence of the rule is *s*/*t*. *minsup* is an input parameter to the algorithm for generating association rules. It defines the support threshold, and rules that have greater support than *minsup* are the only ones generated. *minconf* is an input parameter that defines the minimum level of confidence that a rule must possess.

In this work, we are not concerned about how association rules are generated since we merely call the rule-mining algorithm (Apriori, for example) within our algorithm *WoS*.[2] In general, the values of the confidence and support thresholds are specified by the developer or the user of the association rule algorithm. In this work, determining the most appropriate values of *minconf* and *minsup* is also part of the *WoS* algorithm.

3.2. Filtering out uninteresting and redundant rules

We are interested in some particular association rules generated from a set of user–agent interaction experiences. The rules we are interested in are those association rules of the form '*problem description, assistance action* → *user feedback, evaluation*' having appropriate support and confidence values.

[2] If readers want more information about association rule-mining they should read [1,2,12].

Other combinations of items are irrelevant since we are trying to discover which 'problem situation-assistance actions' pairs have received a positive user feedback and were evaluated, in consequence, as a success.

We can use an intuitive approach [8] to select those rules we are interested in. Relevant (and also irrelevant) classes of rules can be specified with templates. Templates describe a set of rules by specifying which attributes occur in the antecedent and which attributes occur in the consequent. A template is an expression of the form: $A_1,\ldots,A_k \rightarrow A_{k+1},\ldots,A_n$, where each A_i is an attribute name, a class name, or an expression C^+ and C^*, which correspond to one or more and zero or more instances of the class C, respectively. A rule $B_1,\ldots,B_h \rightarrow B_{h+1},\ldots,B_m$ matches the pattern if the rule can be considered to be an instance of the pattern.

Once the agent has filtered out those rules that are not interesting for its purpose, it will still have many rules to process, some of them redundant or insignificant. We can then use a technique that removes those redundant and insignificant associations and then finds a special subset of the unpruned associations to form a summary of the discovered rules [10]. Many discovered associations are redundant or minor variations of others. Thus, those spurious and insignificant rules should be removed. For example, consider the following rules:

$R1$: EventType = doctor, warning → failure, ask for suggestion [sup:60%,conf:90%]

$R2$: EventType = doctor, Priority = high, warning → failure, ask for suggestion [sup:40%,conf:91%]

If we know $R1$, then $R2$ is insignificant because it gives little extra information. Its slightly higher confidence is more likely due to chance than to true correlation. It thus should be pruned. $R1$ is more general and simple. Besides, we have to analyze certain combinations of attributes in order to determine if two rules are telling us the same thing. For example, a rule containing the pair 'suggestion, failure' and another one containing the pair 'warning, success' are redundant provided that they refer to the same problem situation and they have similar confidence values. Fig. 1 shows, as an example, how a set of hypotheses is derived from a set of user–agent interactions. In this example, the assistance action executed by the agent was a warning about two overlapping events.

3.3. Turning hypotheses into facts

Once the agent has formulated a set of hypotheses it has to validate them. Our algorithm tries to prove a hypothesis by analyzing the evidence for and against it. If the hypothesis involves a warning, the evidence supporting it is composed of those assistance experiences in which the user has accepted the warning without asking for a suggestion, and those in which the user would have preferred a warning instead of a suggestion. The evidence against such a hypothesis is given by those interaction experiences in which the user requested a suggestion after the agent has warned him about a problem. If the hypothesis involves a suggestion, it is supported by those interaction experiences in which the user has requested the suggestion and those in which the user has accepted an autonomous suggestion from the agent. The evidence against this kind of hypothesis is provided by those experiences in which the user neglects suggestions.

The evidence for and against a given hypothesis can be obtained by analyzing some of the association rules generated from the interaction database. For example, if we are trying to prove a hypothesis obtained from the rule '*Situation X, warning → fbk1, success Sup:x Conf:y*', and we have another

Transaction Database

Event type 1	Host 1	Topic 1	Participants 1	Event type 2	Host 2	Topic 2	Participants 2	Feedback	Evaluation
Meeting	Me	Projects	Employees	Meeting	Me	Projects	Employees	Ask for suggestion	Failure
Meeting	Boss	Budget	Boss and workmates	Party	Mother	Birthday	Family	OK	Success
Meeting	Boss	Projects	Me and boss	Gym class	Me	None	Me	OK	Success
Appointment	Me	None	Doctor	Appointment	Me	None	Dentist	Ask for suggestion	Failure
Meeting	Me	Projects	Employees	Meeting	Me	Projects	Employees	Ask for suggestion	Failure
Meeting	Me	Projects	Employees	Meeting	Me	Projects	Employees	Ask for suggestion	Failure

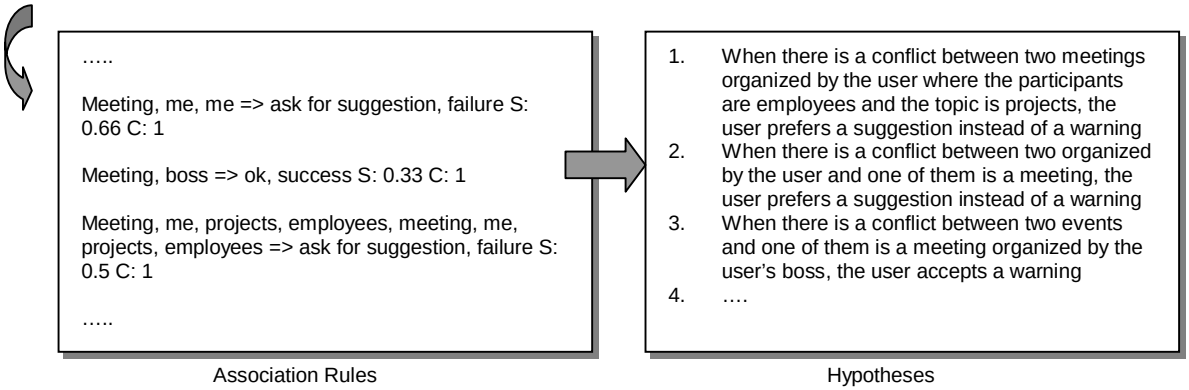

Fig. 1. Formulating hypotheses.

rule that expresses '*Situation X, warning → fbk2, failure Sup:z Conf:w*' (where *fbk*1 and *fbk*2 are feedback items) we should discard the hypothesis or at least adjust its certainty degree.

Given a rule $X \rightarrow Y$ originating a hypothesis H, a rule of the form $X \rightarrow Z$ where $Z \cap Y = \varnothing$ is considered a negative evidence, while a rule of the form $X \rightarrow Z$ where $Z \cap Y \neq \varnothing$ is considered a positive evidence. The certainty degree of a hypothesis H is computed as a function of the support of the rule originating the hypothesis and the support values of the rules considered as positive and negative evidence of H. The function we use to compute certainty degrees is shown in Eq. (1), where α, β and γ are the weights of the terms in the equation (we use $\alpha = 0.8$, $\alpha = 0.15$ and $\gamma = 0.05$), $Sup(AR)$ is the support of the rule originating H, $Sup(E^+)$ is the support of the rules being positive evidence, $Sup(E^-)$ is the support of the rules being negative evidence, $Sup(E)$ is the support value of an association rule taken as evidence (positive or negative), r is the amount of positive evidence and t is the amount of negative evidence

$$Cer(H) = \alpha Sup(AR) + \beta \frac{\sum_{k=1}^{r} Sup(E^+)}{\sum_{k=1}^{r+1} Sup(E)} - \gamma \frac{\sum_{k=1}^{t} Sup(E^-)}{\sum_{k=1}^{r+t} Sup(E)} \quad (1)$$

If the certainty degree of a hypothesis is greater than a threshold value δ the hypothesis is turned into a fact, otherwise it is discarded. The value of this threshold is currently set to 0.5.

Facts are periodically revised to determine whether they are still valid or not. Facts are validated in the same way as hypotheses are validated, i.e. considering negative and positive evidence. If a fact is found invalid, it is turned into a hypothesis again.

3.4. Determining appropriate threshold values

The value of *minsup* determines which association rules are generated and which are not. A high value will probably make us miss some important association rules. Thus, we have to determine a value for *minsup* that enables us to discover the relationships between problem situations and users' assistance requirements that we need to assist users.

In our context, a transaction database contains assistance experiences of a given type (for example those involving a warning) and related to a particular problem situation. Suppose that there are N different instances of a problem situation stored in the database and that they are equally probable. Thus, $1/N$ percent of the transactions will belong to a given problem situation. Since we want those rules showing the most frequent evaluation for a given pair of problem situation and assistance action, a *minsup* value of $1/N$ will be then appropriate. In the example in Fig. 1, the value of *minsup* is 0.25.

The value of *minconf* should be high since it indicates the probability of failing or succeeding in assisting a user in a given problem situation. We found that an appropriate value for *minconf* is 0.8. Smaller *minconf* values can be used to obtain positive and negative evidence when the algorithm is validating a particular hypothesis. Algorithm 2 presents the main steps of the *WoS* algorithm.

3.5. Decision making with WoS

When an agent detects a problem situation or a situation in which the user might need help, it looks for facts telling it what to do in that situation. If the fact implies a suggestion and the confidence on the suggestion is high enough, the agent will make the suggestion. However, if the fact implies a warning or the confidence on the suggestion is low the agent will merely warn the user about the problem. This decision-making process improves the one described in Section 2 by taking into account a user's assistance requirements.

Algorithm 2 *WoS* learning algorithm

Input: A set Ex of user–agent interaction experiences $Ex_i = <Sit_i, A_i, UF_i, E_i>$ (where *Sit*: situation description, *A*: assistance action, *UF*: user feedback, *E*: evaluation)

Ouput: Facts F and hypotheses H about the user's assistance requirements

1: minsup ← Obtain value for minsup and mincof ← 0.8
2: AR ← Call rule-mining algorithm (e.g. Apriori) with Ex, minconf and minsup
3: AR_1 ← Filter out uninteresting rules from AR
4: AR_2 ← Eliminate redundant and insignificant rules from AR_1
5: H ← Transform rules in AR_2 into hypotheses (just notation transformation)
6: **for** $i = 1$ to size of H **do**
7: Find evidence for (E^+) and against $(E^-)H_i$
8: $Cer(H_i)$ ← compute certainty degree of H considering (E^+) and (E^-)
9: **if** $Cer(H_i) \geq \delta$ **then**
10: $F \leftarrow F \cup H$
11: end if
12: end for
13: **for** $i = 1$ to size of F **do**
14: Validate F_i (as in 8 to 12)
15: end for

4. Experimental results

To evaluate the performance of our interface agents' learning algorithm we used one of the metrics defined in [3]. The *precision metric* measures an interface agent's ability to accurately provide assistance to a user. We define our precision metric as shown in Eq. (2)

$$M_{\text{precision}} = \frac{\text{number of correct assistance actions}}{\text{number of assistance actions}} \quad (2)$$

The precision metric is used to evaluate the performance of an interface agent when it has to decide among a warning, a suggestion or no action. In this case, for each problem situation, we compare the number of correct assistance actions against the total number of assistance actions the agent has executed.

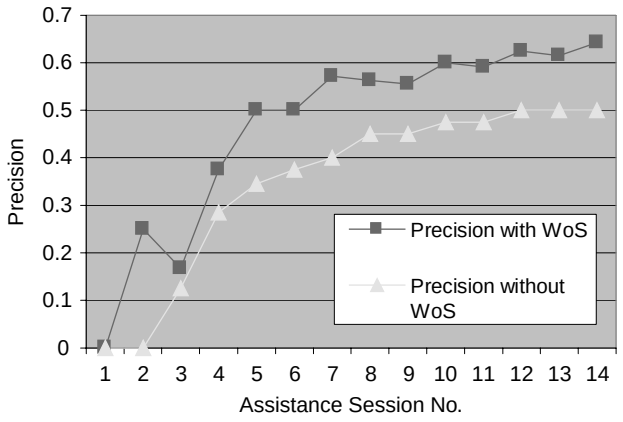

Fig. 2. Agent's assistance precision using *WoS*.

An assistance action is correct if it is the one the user expected in a given problem situation. The user's feedback tells us whether an assistance action is correct or not.

We tested our algorithm with a set of 30 users of an agenda system. For this purpose, we incorporated the *WoS* algorithm into an agent that provided assistance to these users. We compared the assistance capability of this agent with *WoS* and without *WoS*. Half of the users used the agent with *WoS* and the other half used the agent without our algorithm. We obtained the user feedback after each assistance action executed by each agent and we classified the assistance experience as a success or as a failure.

The graph in Fig. 2 plots the evolution of the precision metric both for an agent using *WoS* and for an agent not using WoS. For each assistance session the graph plots the average precision value. We used the following numeric parameters to perform the tests: $minconf = 0.8$, $minsup = 0.2$ (we fixed this value for simplicity purposes), $\delta = 0.5$, number of association rules generated = 2000, number of assistance actions = 30.

We can observe that the number of correct assistance actions is bigger for the agent using our learning algorithm. There is a pattern of improving performance in our agent's assistance capability, which indicates that the *WoS* algorithm enables agents to improve their interactions with the users they are assisting.

5. Related work

Some related works have considered different approaches to decide among several possible agent actions. In Section 1, we have discussed those that use confidence values associated with actions to decide what to do [3,9,11]. The main problem with this approach is that it does not take into account the user's requirements in different situations to compute confidence values and that threshold values are generally fixed or user-defined.

On the other hand, the works described in [7] and [5] try to infer the ideal action in light of costs and benefits of each possible action given uncertainties about a user's goals. Each action has a utility function established by a domain expert. In a given situation, the agent executes the action that has the maximum expected utility value. The work described in [7] presents the LookOut system, which helps users that use Microsoft Outlook messaging and scheduling system. Depending on the inferred probability about a user's goals and on an assessment of the expected costs and benefits of action (expected utility), the system decides to: do nothing but simply wait for manual invocation of LookOut, to engage the user in a dialog about his intentions with regards to providing a service, and to go ahead and attempt to provide its service. Similarly, the work presented in [5] describes a utility-based decision-making process to determine when to take the initiative to interact with a user to request further assistance from him in a mixed-initiative system. The decision-making is based on a calculation of the expected utility of several courses of action and the likelihood that the user would be an effective contributor of information, if an interaction was initiated. The main problem of this utility-based approach is how utility functions are computed. These functions have default values (as in LookOut), they are specified by users (LookOut also allows this) or they are established by a domain expert. None of these alternatives takes into account how the user wants to interact with the agent in different contexts.

6. Conclusions and future work

The work presented in this paper constitutes a contribution both to the interface agents and the human–computer interaction areas. The *WoS* algorithm enhances interface agents' capabilities by enabling them to determine how to best assist users in different problem situations. In this way, these agents personalize the interaction with each user. The results obtained so far are quite promising, since our agents have improved their assistance capabilities and their actions tend to the users' needs.

We used association rules to implement our proposed learning algorithm because they enable us to rapidly discover associations between different situations and the expected assistance actions. Other machine learning or data mining techniques could be also used, and they will be tested in the future and compared with our first proposal.

We are now working on a temporal version of our algorithm, which considers the 'age' of transactions to derive association rules that will serve as hypotheses and as evidence. In some cases, new positive or negative evidence is probably more relevant than older one.

References

[1] R. Agrawal, R. Srikant, Fast algorithms for mining association rules, in: Proceedings of the 20th VLDB, 1994, pp. 487–499.
[2] R. Bayardo, R. Agrawal, Mining the most interesting rules, in: Proceedings of the Fifth ACM SIGKDD, 1999, pp. 145–154.
[3] S. Brown, E. Santos, Using explicit requirements and metrics for interface agent user model correction, in: Proceedings of the Second International Conference on Autonomous Agents, 1998.
[4] S. Brown, E. Santos Jr., S. Banks, Utility theory-based user models for intelligent interface agents, in: Proceedings of the 12th Canadian Conference on Artificial Intelligence, 1998.

[5] M. Fleming, R. Cohen, A utility-based theory of initiative in mixed-initiative systems, in: IJCAI 01 Workshop on Delegation, Autonomy and Control: Interacting with Autonomous Agents, 2001, pp. 58–65.
[6] M. Hearst, Mixed-initiative interactions: trends and controversies, in: IEEE Intelligent Systems, September/October, 1999, p. 14.
[7] E. Horvitz, Principles of mixed-initiative user interfaces, in: ACM SIGCHI Conference on Human Factors in Computing Systems—CHI99, 1999.
[8] M. Klementinen, H. Mannila, P. Ronkainen, H. Toivonen, A.I. Verkamo, Finding interesting rules from large sets of discovered association rules, in: Third International Conference on Information and Knowledge Management, 1994, pp. 401–407.
[9] R. Kozierok, P. Maes, A learning interface agent for scheduling meetings, in: ACM-SIGCHI International Workshop on Intelligent User Interfaces, 1993, pp. 81–93.
[10] B. Liu, W. Hsu, Y. Ma, Pruning and summarizing the discovered associations, in: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 1999.
[11] P. Maes, Agents that reduce work and information overload, Communications of the ACM 37 (7) (1994) 31–40.
[12] R. Srikant, Q. Vu, R. Agrawal, Mining association rules with item constraints, in: Proceedings of the Third International Conference on KDD, 1997.