Available online at www.sciencedirect.com
ScienceDirect
Int. J. Human-Computer Studies 68 (2010) 209–222
International Journal of Human-Computer Studies
www.elsevier.com/locate/ijhcs

Building respectful interface agents

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

[a]ISISTAN Research Institute, Fac. Cs. Exactas, UNCPBA Campus Universitario, Paraje Arroyo Seco, Tandil 7000, Argentina
[b]CONICET, Consejo Nacional de Investigaciones Científicas y Técnicas, Argentina

Received 15 May 2008; received in revised form 11 November 2009; accepted 11 December 2009
Communicated by C. Sierra
Available online 23 December 2009

Abstract

To provide personalized assistance to users, interface agents have to learn not only a user's preferences and interests with respect to a software application, but also when and how the user prefers to be assisted. Interface agents have to detect the user's intention to determine when to assist the user, and the user's interaction and interruption preferences to provide the right type of assistance without hindering the user's work. In this work we describe a user profiling approach that considers these issues within a user profile and a decision making approach that enables the agent to choose the best type of assistance for a given user in a given situation. We also describe the results obtained when evaluating our proposal in the tourism domain, and we compare these results with some previous ones in the calendar management domain.

Keywords: User-agent interaction; User profiling; Interface agents; Plan recognition

1. Introduction

Research on Interface Agents has mainly concentrated on learning users' habits, preferences and interests to provide them personalized assistance with the tasks they perform with software applications. However, few efforts have been directed towards learning how to best assist the user, that is providing the right help, at the right time, and in the right way.

A user can perform different tasks with a given software application. Thus, it is very important for an interface agent to know, at every moment, the task the user is carrying out because it gives the context in which the user is working. By taking this context into account, the agent may infer the user's intention and try to collaborate with him/her in a respectful way. In addition, if the agent knows the user's intention, it will avoid interrupting him/her at an improper moment. Users generally do not want to be interrupted while working on an specific task, unless this interruption is strongly related to the task they are performing, or it has a high priority for them (Rudman and Zajizek, 2006). Also, users differ in their preferences about how and when they want to be assisted, and even a single user may differ in the type of assistance he/she prefers for different contexts (Schiaffino and Amandi, 2004, 2006; Serenko et al., 2007).

Consider for example the following situation. An employee of a tourist agency has the intention of arranging a tour for a client named John Smith. To achieve this, he/she has to perform a set of tasks in the application, such as selecting John Smith from the list of clients, creating a new tour, entering the location, the type of accommodation, the estimated dates, the tour price, and finally sending an email to John Smith with the proposal. The sooner the agent detects the user's intention, the better it will assist him in accomplishing his intention. In our approach, we use Plan Recognition to detect the user's intention. Plan recognition aims at identifying the goal of a subject based on the actions he/she performs (Kautz, 1991). The goal usually

*Corresponding author at: ISISTAN Research Institute, Fac. Cs. Exactas, UNCPBA, Campus Universitario, Paraje Arroyo Seco, Tandil 7000, Argentina.
E-mail addresses: sschia@exa.unicen.edu.ar (S. Schiaffino). marmenta@exa.unicen.edu.ar (M. Armentano), amandi@exa.unicen.edu.ar (A. Amandi).

1071-5819/$ - see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.ijhcs.2009.12.002

has one or more associated plans that can predict the subsequent user behavior. Furthermore, the user's interaction history can be used to refine the context that he/she is implicitly setting, by considering not only the possible tasks that he/she will probably perform next, but also the attributes of those tasks. Thus, an interface agent can observe the user while he/she is working with an application to detect each action performed on the graphical user interface and determine what the user's intention is.

Following our example, once the agent has detected that the user wants to create a tour for John Smith, it can use the information contained in the user profile to assist him. For example, the agent can suggest the employee the preferred type of accommodation or the means of transport for John Smith, it can complete these fields automatically, and offer to send an email to John Smith. However, different users may have different preferences about the type of assistance they welcome from an interface agent. For example, some users may prefer the agent to automatically complete all the tasks it can, while others just prefer to receive suggestions (Schiaffino and Amandi, 2004). Moreover, this information is strongly dependent on the situation in which the agent is about to assist the user. In our approach, the information needed to determine what type of assistance a user wants to receive in a given situation is contained in the user interaction profile. This profile also comprises the expected modality of the assistance. In a certain context the user might want just a notification containing a complaint from a client while in a different context the user might prefer an interruption.

In summary, to assist a user successfully, interface agents have to detect: a user's intentions, learn a user's preferences and habits with respect to a software application, and a user's interaction and interruption preferences. In this work, we propose a profiling approach to acquire the different components of the user profile mentioned before, and a decision making algorithm that uses this profile to decide how to best assist a user.

Our profiling approach uses, first, plan recognition to detect a user's intentions. Then, we use two user profiling methods we developed, namely *WATSON* and *IONWI*, to learn a user's interaction and interruption preferences (Schiaffino and Amandi, 2006). Finally, we combine the different components of the user profile in a decision making algorithm that enables an interface agent to decide how to best assist a user in a given situation.

The rest of the work is organized as follows. Section 2 presents an overview of our proposed approach. Section 3 describes how to detect a user's intention using plan recognition. Section 4 describes how to learn a user's interaction and interruption preferences. Section 5 presents the results obtained when evaluating our approach in the tourism domain. Then, Section 6 analyzes some related works. Finally, Section 7 presents our conclusions and future work.

2. Proposed approach

A user profile typically contains information about a user's interests, preferences, behavioral patterns, knowledge, and priorities, regarding a particular domain. However, such information is not enough to personalize the interaction with a user. The user's intentions with a software application and his/her interaction preferences play a relevant role in user-agent interactions. To obtain the components of our user interaction profile, we developed two profiling methods: *WATSON* and *IONWI*. *WATSON* learns a user's assistance preferences, that is, when a user wants a suggestion, a warning, an automated action or no assistance. *IONWI* learns a user's interruption preferences, that is, when a user prefers an interruption and when a notification. To achieve their goals, these user profiling methods analyze the user's interaction with the agent recorded when observing the user's behavior, and consider the feedback provided by the user after the agent assisted him/her. An overview of our proposal is shown in Fig. 1. In the following sections we describe our approach in detail.

2.1. Our user profile

In our approach, a user profile should contain information about the user's intentions with a software application, and about the user's interruption and assistance preferences (see Fig. 1). We therefore add to the classic user profile[1] information about the situations or contexts in which the user: requires a suggestion to deal with a problem, needs only a warning about a problem, expects an action on his or her behalf, does not want the agent's help. In turn, we include in the user profile the situations or contexts in which the user: accepts an interruption from the agent, or wants a notification rather than an interruption.

Thus, our user profile comprises:

$$\begin{aligned} \textit{User Profile} &= \textit{Classic UserProfile} + \textit{User Intentions} \\ &\quad + \textit{UserInteraction Profile} \\ \textit{User Interaction Profile} &= \textit{UserAssistance Preferences} \\ &\quad + \textit{UserInterruption Preferences} \end{aligned}$$

We define the user intentions as the set of all the possible intentions the user can be trying to achieve, each of them with an associated certainty:

$$\textit{User Intentions} = \{\langle \textit{User intention}, \textit{Certainty} \rangle\}$$

We define the assistance preferences as a set of problem situations or contexts with the required assistance action and a parameter (certainty) indicating how sure the agent is

[1]We call classic user profile to the profile considered thus far for most interface agents, that is, without interruption and interaction preferences. User intentions have been considered as part of user profiles in the literature (Horvitz et al., 1998), but we separate them explicitly since they are an important part of our proposal.

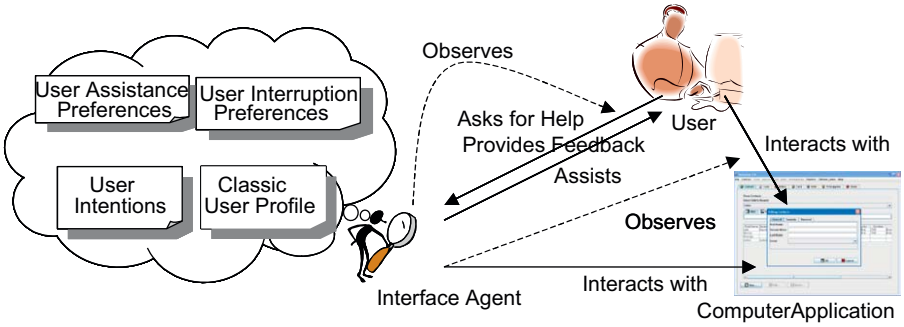

Fig. 1. Overview of our proposed approach.

about the user wanting that assistance action in that particular situation:

$$User\ Assistance\ Preferences = \{\langle Situation, AssistanceAction, Certainty\rangle\}$$

The situation describes the problem situation, situation of interest or assistance opportunity underlying the interaction between the user and the agent. In a tourism application, some situations might be: the user is creating a tour for a given client, or the client himself is asking for information about a tour; or the user is processing a complaint submitted by a certain client. Each situation is described by a set of attributes or features, and each attribute can take a predefined set of values. Formally, a situation is defined as $situation = (feature_i, value_{ji})+$, where $value_{ji}$ is the j-th value feature i can take. For example, in the case of booking a tour, the situation will be described by the tour features such as the type of accommodation, the location, the duration, the cost, among other characteristics.

The agent action associated with a given situation may be a warning, a suggestion, an action on the user's behalf, or no assistance action at all.The certainty degree can take a value between 0 and 1.

$$Assistance\ Action = warning|suggestion|action|noaction$$

$$Certainty\ \varepsilon\ [0..1]$$

We define the interruption preferences as a set of situations with the preferred assistance modality (interruption or no interruption). They might also contain the type of assistance action to execute. A parameter indicates how certain the agent is about this user preference:

$$User\ InterruptionPreferences = \{\langle Situation, [AssistanceAction], Assistance\ Modality, Certainty\rangle\}$$
$$Assistance\ Modality = interruption|notification$$
$$Certainty\ \varepsilon\ [0..1]$$

In the components mentioned above, how the certainty value is obtained, depends on the underlying machine learning algorithm. As we will explain later, in the case of association rules it is a combination of support and confidence, which are parameters of association rule mining algorithms; in the case of Bayesian networks, it is defined by probability values.

Following, there are two examples of user interaction profile components in a tourism application. The first indicates that, when the user is creating a tour and the client is John Smith, the user requires a suggestion about some tour attributes with a certainty degree of 0.7:

$$assist\text{-}pref\ (\{(type, createtour), (client, JohnSmith)\}, suggestion, 0.7)$$

The second one expresses that, when an email arrives with a complaint of a "vip" customer, the user wants to be interrupted no matter the task he/she is carrying out, with a certainty degree of 0.8:

$$int\text{-}pref\ (\{(type, complaint), (customertype, vip)\}, warning, interruption, 0.8)$$

In the first example, the possible user's intentions could be:

$$\{\langle Create\ tour, 0.9\rangle, \langle Write\ email, 0.4\rangle, \langle Book\ tour, 0.3\rangle\}$$

2.2. Proposed decision making algorithm

The most widely used approach to select an agent action in the interface agent area is the one proposed in Maes (1994). In this confidence-based approach, interface agents have generally three possibilities when they want to assist a user: executing a task autonomously, suggesting to the user what to do, and doing nothing. Agents use two threshold values to take decisions, which are established by the user to control the agents 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 tells the user what it thought he/she would do and it waits for confirmation to automate the action; and if the confidence value is greater than the *do-it* threshold the

agent executes the task autonomously on the user's behalf, sending him/her a report. The *do-it* threshold is higher than the *tell-me* threshold. If the agent does not execute an action, then the user has to deal with the situation at hand, and the agent observes his/her behavior to learn from it.

The confidence-based approach has several problems. The main one is that confidence values do not consider the way in which the user wants to interact and work with his/her agent. These agents do not take into account when the user wants each type of assistance action. Moreover, threshold values are generally set by the user 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 (context) or situation of interest.

To solve these problems we propose the algorithm shown in Algorithm 1. The inputs for this algorithm are the actions the user executed in the application observed by the user, and the user profile. The output of the algorithm is a decision that consists of the type of the assistance action (warning, suggestion, action on the user's behalf), the content of the assistance action (what to do or what to suggest), and the modality of the assistance action (interruption or notification). First, the algorithm tries to determine the user's intentions with the software application in order to detect a situation in which the user might need assistance. We use a plan recognition algorithm to achieve this goal. Then, given such a situation, the agent looks in the user profile for a profile item containing that situation. To make a decision, the agent first searches the user profile looking for a profile item involving the situation at hand. If no profile item is found for this situation, then the agent cannot provide assistance to the user because it does not have information to do it. Thus, no assistance action is executed. Another alternative could be warning the user about the problem.

If a profile item is found, then the agent has information to assist the user. Thus, it has to decide the type of the assistance to be provided. To achieve this goal, the agent looks into the user interaction profile for an assistance preference containing the situation at hand or a similar situation. If such an assistance preference is not found, then the confidence-based decision making algorithm is used. This algorithm has to be used because the agent does not know the user's assistance or interruption preferences for the situation it is dealing with. Thus, the algorithm used currently by interface agent has to be used.

On the other hand, if such an assistance preference is found the agent retrieves from this item the type of the assistance action, the assistance to be provided, and the assistance modality associated with the situation and the current user task. Then, the confidence on the action is computed. The confidence is compared against the corresponding threshold value, one for each action type. If the confidence value is greater than the threshold value, then the assistance action is correct and it will be executed. However, if the confidence value is smaller than the threshold value, then we have to analyze what the user wants. If the user is risky, that is if he wants the action despite it has a low confidence value, then the action contained in the user assistance requirement is performed. Otherwise, a different action is executed. If the action is an action on the user's behalf but the confidence on the action is smaller than a threshold value τ_2, then the agent will make a suggestion. If the action contained in the assistance requirement is a suggestion and its confidence value is smaller than τ_3, the agent will only make a warning.

Algorithm 1. Decision making algorithm

Input: The user profile *UP* composed by the standard user profile *SUP*, the user interruption preferences *UIP* and the user assistance requirements *UAR*; a set of observed actions *Actions*

Output: The agent has chosen a type of action to deal with *Sit*: warning *W*, suggestion *S*, action *A* or no action *N*; a modality: interruption *I* or no interruption *NI*; and a content. These three items constitute the decision *Dec*

1: *intentions* ← plan-recognition (*Actions*)
2: *Sit* ← get-most-probable-intention (*intentions*)
3: compute *Conf*(*Sit*)
4: **if** *Conf*(*Sit*) ≥ τ_1 **then**
5: *Dec.content* ← select from *SUP* a profile item containing *Sit*
6: **if** *Dec.content* ≠ ∅ **then**
7: *AR* ← select from *UAR* a profile item containing *Sit*
8: **if** *AR* ≠ ∅ **then**
9: *type* ← *AR.agentaction*
10: **if** risky user **then**
11: *Dec.action* ← *type*
12: **else**
13: **if** *type* = *action* and *Conf*(*Dec.content*) ≥ τ_2 **then**
14: *Dec.action* ← *A*
15: **else if** (*Conf*(*Dec.content*) < τ_2) or (*type* = *suggestion* and *Conf*(*Dec.content*) ≥ τ_3) **then**
16: *Dec.action* ← *S*
17: **else if** (*type* = *warning*) or (*Conf*(*Dec.content*) < τ_3) **then**
18: *Dec.action* ← *W*
19: **else**
20: *Dec.action* ← *N* {*type* = *noaction*}
21: **end if**
22: **end if**
23: **else**
24: *Dec.action* ← Call confidence-based decision making algorithm
25: **end if**
26: *IP* ← select from *UIP* a profile item containing *Sit*
27: **if** *IP* ≠ ∅ **then**

```
28:        Dec.modality ← IP.modality
29:       else
30:        Dec.modality ← notification
31:       end if
32:      else
33:       Dec.action ← N or W
34:      end if
35:     else
36:      Dec.action ← N {the agent considers the situation is
         not worth handling}
37:     end if
38:     return Dec
```

An important issue interface agents developed thus far do not consider in their action selection algorithms is *how* to provide assistance to the user. Once the agent has selected which action to perform it has to decide whether to interrupt the user or not, in order to provide him assistance. Thus, it looks for an interruption preference containing the situation and, preferably, also containing the current user task. If such a preference exists, the agent retrieves the modality associated with the situation-task pair or with the situation only. If no interruption preference exists, the agent provides him/her assistance without interrupting his work, letting him decide when to pay attention to that assistance.

3. Detecting a user's intentions

Plan recognition can be used to infer the user's intentions based on the observation of the tasks the user performs in the application. Plan recognizers take as inputs a set of goals the agent expects the user to carry out in the domain, a plan library describing the way in which the user can reach each goal, and an action observed by the agent. The plan recognition process itself, consists in foretelling the user's goal, and determining how the observed action contributes to reach it.

There are two main aspects that make classical approaches to plan recognition unsuitable for being used by interface agents. First, the agent should deal with transitions and changes in the user's intentions. The agent will usually not be certain about the moment in which the user starts a new plan to achieve a new goal. Classical approaches do not consider this problem and they restrict the plan recognition process to only one session that starts with the first observed action and ends when the algorithm recognizes the user's intention. In an interface agent environment the user not only may start pursuing new goals in the application with no preplanned behavior, but also he/she may change his/her intention without completing the previous one. Those approaches that consider this problem restrict the memory of the plan recognizer so that it only considers the most recent tasks performed by the user, or it considers each task for only a fixed interval of time and then they are completely disregarded.

Second, in the interface agents approach a user's preferences have a fundamental role and should be considered in the plan recognition process. The reason for this is that the behavior of the user in a specific situation is usually determined by his/her preferences related to that situation. In this context, a situation is described by the information the user is handling to achieve his/her intention. In the calendar scheduling domain, for example, the situation related to the addition of a new event may be described by its date, time, participants, place, and organizer. These domain variables strongly influence the user's behavior. For instance, if the user adds a new event that takes place during his/her working hours, there is a high probability that the event will take place at the office.

We propose Bayesian Networks (Jensen, 2001) as a knowledge representation capable of capturing and modeling dynamically the uncertainty of user-agent interactions. We represent the set of intentions the user can pursue in the application domain as an Intention Graph. An Intention Graph is materialized as a Bayesian Network and represents a context of execution of tasks. The context is viewed as the set of tasks that the user has performed recently, and will influence the certainty that the agent has in any given intention that the user may be pursuing. The number of tasks taken into account is variable and depends on a function we will explain later in this section. Bayesian networks are directed acyclic graphs representing both the conditional dependencies and independencies between elements in the problem domain. The knowledge is represented by nodes called random variables and arcs representing the causal relationships between variables. Each variable has a finite set of mutually exclusive states. Nodes without a parent node have an associated prior probability table. On the other hand, the strengths of the relationships are described using parameters encoded in conditional probability tables.

Bayesian networks are used for calculating new probabilities when some particular information becomes available. This information is called evidence. In our case, we will have new evidence every time the user performs a task in the software application. Therefore, evidence will be in the form $AddContact = true$ meaning that the user performed the *AddContact* task. By making use of Bayesian networks probabilistic inference, we will be able to know, having as evidence the set of tasks performed by the user, the probability that the user is pursuing any given intention modeled by the Intention Graph. Moreover, if the user explicitly declares his/her intentions, we will be able to probabilistically assess the tasks he/she has to perform to achieve his/her goal. In the following section we describe the concept of Intention Graph.

3.1. Intention graph

In our Intention Graph variables correspond to goals that the user can pursue in the application domain and to

tasks that the user can perform in the application to achieve those goals. The two possible states of these variables are true and false. A true value in a variable representing a goal indicates that the user is pursuing that goal. On the other hand, a true value in a variable representing a task indicates that the user performed that task in the application. We call certainty of an intention the probability of the corresponding variable being in a true state. Notice that we will not have direct evidence about the goal the user is pursuing unless the user makes an explicit declaration of his/her intention. Evidence on a task node will be set when the user interacts with a widget in the application GUI that is associated to the execution of that task. Our Intention Graph includes a third kind of variable: context variables. This kind of variables will be used to personalize the intention detection process by learning new relations that may arise between the attributes of the tasks performed by the user and the intention nodes in the intention graph.

For example, in a tourism application, the user can select a tour with the objective of creating a package including that tour, inviting a friend to make that tour or with the objective of making a claim, as shown in the Intention Graph presented in Fig. 2. The Intention Graph constructed manually by a domain expert will allow the agent to rank which of the three goals is more probable, given that the user selected a tour in the application. However, the information of the selected tour can be relevant in discerning which goal the user is pursuing.

To consider this information, we introduce to the definition of our Intention Graph, the concept of traceable nodes. A traceable node is a node in which we want to register the values taken by some attributes of the corresponding task performed by the user with the aim of adding new variables that represent the context in which the user performs that task, and to find new relations between these variables and the nodes in the Intention Graph. In the example above, the task corresponding to the selection of a tour is a traceable node. The designer of the Intention Graph should decide which attributes of this task are of interest (for example, the destination, the season and the cost of the tour) for which set of intentions (in the example, creating a package including that tour, inviting a friend to make that tour or making a claim about that tour).

According to the values taken by the variables represented by these traceable nodes, probabilities in the Intention Graph will be updated, as described in the following section.

3.2. Probability adaptation

Every time the user performs a task corresponding to a traceable node, the agent will observe the values taken by the attributes of the task (for example, the selected tour has destination Rome and its cost is expensive). Then, the agent will continue observing the user until it can infer what his/her intention(s) are and will save the experience in an interaction history. Each experience will be in the form: $\langle attribute_1, attribute_2, \ldots, attribute_n, intention_1, intention_2, \ldots, intention_k \rangle$, where $attribute_i$ is the value taken by the $attribute_i$ and $intention_j$ is true if the agent infers that the user was pursuing $intention_j$, or false otherwise. This database of experiences is then used by the agent to run K2 batch learning algorithm (Cooper and Herskovits, 1992) to find relations between the attributes of the traceable node themselves, and between the attributes and the intentions in the Intention Graph. K2 algorithm starts with a network without arcs, and assumes that nodes are sorted to prevent cycles. In each iteration, for each variable X_i the algorithm adds to its set of parents Π_i the node which order is previous to it and that it increments the quality of the network according to the quality metric selected. The process is repeated until the quality is not incremented (i.e. the increment in quality is not greater than a certain

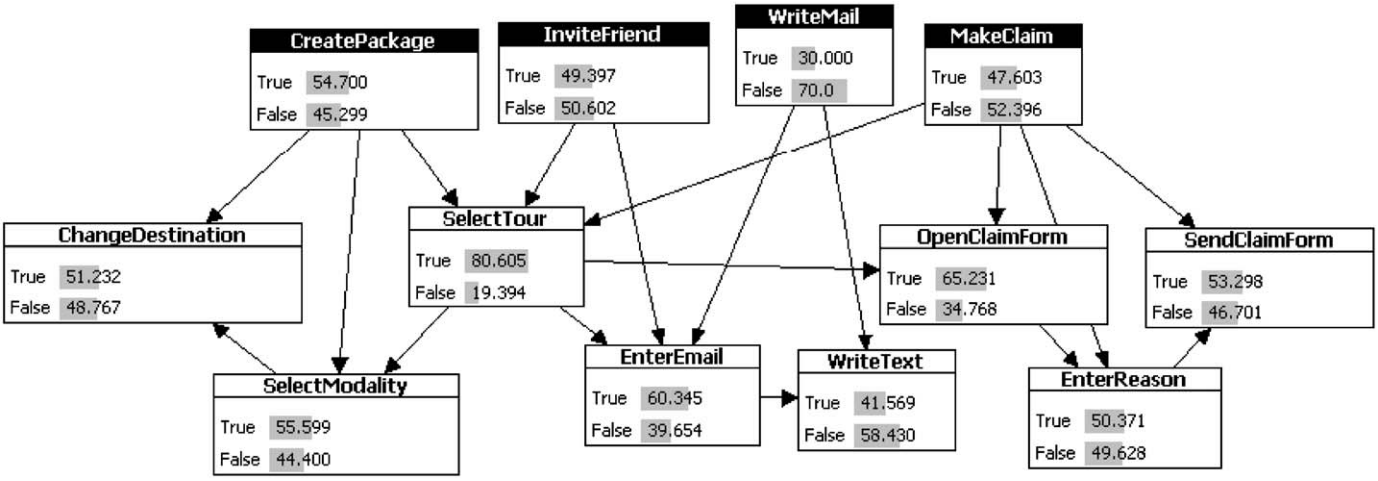

Fig. 2. Example of an intention graph.

minimum value) or it ends up in a complete network. The maximum number of parents for each variable can be also limited in the searching process.

Next, EM (Lauritzen, 1995) parametric learning algorithm is run to learn the associated probabilities of the Bayesian Network learnt by K2. In each step EM estimates the missing values to complete the dataset. This estimation is done using the values of the variables that were observed. EM algorithm, is an iterative method which alternates between performing an expectation (E) step, which computes an expectation of the log likelihood with respect to the current estimate of the distribution for the latent variables, and a maximization (M) step, which computes the parameters which maximize the expected log likelihood found on the E step. These parameters are then used to determine the distribution of the latent variables in the next E step.

Fig. 3 shows an example of a learned network for the example we are describing. In this machine-learned Bayesian network, variables representing attributes are incorporated to the network as a new kind of variables, context nodes (with a gray label in the figure), and variables representing intentions as intention nodes (with a black label in the figure). The network resulting from the running of learning algorithm is then merged with the Intention Graph to incorporate this knowledge to the detection of the user's intention.

When the intention graphs are constructed, the conditional probabilities chosen by the designer might not be accurate to every future user of the application. Furthermore, when the user uses the application, new situations are observed and it is desirable that the agent could be able to learn about them. To adapt the conditional probabilities of the nodes in the Intention Graph, we follow the statistical on line learning approach presented in Jensen (2001) called *Fractional Updating*. As the user interacts with the application, the agent remembers the tasks performed by the user. When the agent infers that the user is pursuing a certain goal, it adapts the probabilities of the intention graph to reflect this new experience. The agent will believe that the user has already completed the current goal(s) when a certain threshold is exceeded. This threshold should be empirically determined for each application domain.

3.3. Fading evidence

Finally, as stated in Section 2, most of previous plan recognition approaches do not consider the uncertainty related to the moment in which the user starts a new plan to achieve a new goal. Those which consider this issue limit the memory of the plan recognizer by making evidence to be present in a fixed interval of time and then completely disregarding it. We take a different approach in which evidence is gradually forgotten.

In Bayesian networks terms, evidence is a collection of findings on variables. A finding may be hard or soft. A hard finding specifies the value a variable takes. Findings on the values taken by the variables introduced before are of this kind. A soft finding, on the other hand, specifies the probability distribution of a variable. Hard evidence is a collection of hard findings and Soft evidence is a collection of soft findings. We adopt the concept of soft evidence to fade the evidence we entered to the Bayesian network as the user performs further tasks. To do so, we use a fading function to gradually forget the tasks performed by the user.

When the agent observes the execution of a task t, it sets a *hard evidence* in the intention graph, that is to say that we know that the variable corresponding to task t took a "true" value with 100% of certainty: $P(t = true) = 1$.

However, as the user performs subsequent tasks, we gradually "forget" of past observations, reducing the likelihood of the evidence we have so far, according to a

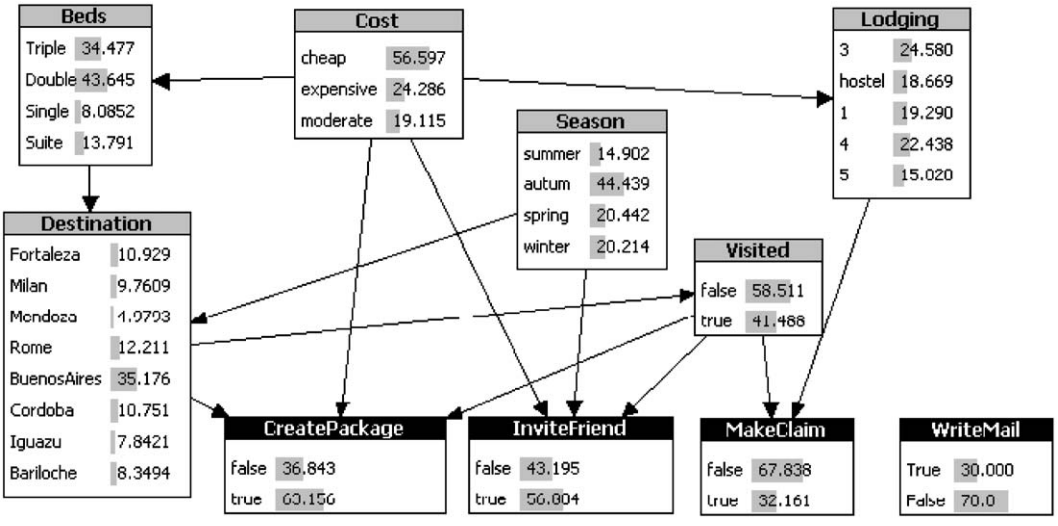

Fig. 3. Example of a Bayesian network built from data.

fading function $\mathcal{F}(\cdot)$, as shown in Eq. (1).

$$P(t = true) = P_{prev}(t = true) - \mathcal{F}(\cdot) \quad (1)$$

That is, the current probability that task t was performed $P(t = true)$ is computed by subtracting from the previous probability assigned to task t $P_{prev}(t = true)$ a value assigned by the fading function $\mathcal{F}(\cdot)$. The use of a probability distribution to set evidence is known as *soft evidence*. Evidence is faded according to this fading function until it reaches its original value, that is until the probability of a given node becomes less than the value that it would have if we would not have observed the execution of the corresponding task in the application.

Fading functions can be any function that, given the Intention Graph and the evidence on tasks performed so far, decrements the certainty of the evidence gradually according to some heuristic (Liu et al., 2007). For example, we can decrement current evidence by a fixed factor $0 \leq \Delta \leq 1$ every time the user performs a task in the application. This way, for all evidenced nodes t_i we will update the probability distribution of its evidence according to Eq. (2)

$$P(t_i = true) = P_{prev}(t_i = true) - \Delta \quad (2)$$

This simple fading function will allow the agent to disregard, after some actions performed by the user, a previously performed noisy action or a previously pursued goal. This function, however, might have the problem of rapidly forgetting evidence in tasks that actually contribute to the current goal of the user. This problem can be overcome using a more sophisticated function that, for example, keeps almost intact the evidence of some number of tasks and then quickly decrement up to the original value of the node (without evidence).

Allowing to gradually forget past observation, the agent not only would be able to manage changes in the goal of the user, but also will allow it to forget the execution of noisy tasks, that are tasks that do not belong to the main goal the user is pursuing. For example, in the tourism application, the user can check the currency exchange rate for its own sake, and not because it is part of the plan the user is pursuing in that moment.

4. Learning a user's interaction and interruption preferences

To learn a user's interaction and interruption preferences, the information obtained by observing a user's behavior is recorded as a set of user-agent interaction experiences. An interaction experience $Ex = \langle Sit, Act, Mod, UF, E, date \rangle$ is described by six arguments: a situation *Sit* that originates an interaction; the assistance action Act the agent executed to deal with the situation (warning, suggestion, action on the user's behalf); the modality *Mod* that indicates whether the agent interrupted the user or not to provide him/her assistance; the user feedback *UF* obtained after assisting the user; an evaluation E of the assistance experience (success, failure or undefined); and the *date* when the interaction took place. The situation, assistance action and modality were already defined in Section 2.1.

The user feedback can be explicit (direct) or implicit (indirect), as well as positive or negative. It is explicit when the user explicitly evaluates the agent's actions using some mechanisms provided by a user interface. It is implicit when the agent has to observe the user's actions after it has assisted him in order to obtain the feedback. In turn, the user can also explicitly state some interaction preferences. However, asking the user when it is convenient to interrupt him or when he needs a suggestion instead of a warning may be not a viable solution, since he probably does not know what to answer and he may not want to spend time providing this information. In this work we will not consider that the agent can obtain information about the user from other agents. Although two users may have similar interests, they would certainly not interact with their agents in the same way. The implicit feedback can adopt different forms such as the agent executing an action different from the one suggested by the agent, the user executing the task the agent suggested him, asking the agent for another solution, or not dealing with the problem at hand.

Once the agent has gathered feedback for its assistance, it has to evaluate if the assistance experience was a success, a failure or neither a success nor a failure, i.e. it is undefined. An interaction experience is evaluated as a success if the type of assistance the agent provided was the one the user expected. An interaction experience is evaluated as a failure if the assistance provided was of the wrong type. Finally, if the agent cannot tell whether the assistance experience is a success or a failure, because the agent could not gather implicit feedback and the user did not provide explicit feedback, the interaction is rated as undefined.

Our profiling approach takes as input the set of user-agent interaction experiences and learns when the user requires a suggestion to deal with a situation, when he/she needs a warning and when the user wants the agent to perform a task on his/her behalf. It also determines when the agent can interrupt the user's work to provide him/her assistance and when it can only send him/her a notification without interrupting the user's work. The outputs of the algorithms constitute part of the user interaction profile, which is used by the agent to decide how to assist a user in a given situation. The content of the assistance action (e.g. what was suggested) is obtained from the user's preferences and interests, as interface agents have done thus far.

As we have said before, we developed two profiling methods, *WATSON* and *IONWI*, to obtain a user's interruption and assistance preferences. These algorithms use association rules to discover the existing relationships between situations or contexts and the assistance actions a user requires to deal with them, as well as the relationships

between a situation, a user task, and the assistance modality required. Fig. 4 shows the main steps of the algorithms. These steps are similar for both algorithms, but they differ in how rules are processed in each of them and the structure of the outputs.

Association rules imply a relationship among a set of items in a given domain. As defined by Madani et al. (2009), association rule mining is commonly stated as: Let $I = i_1, \ldots, i_n$ be a set of items and D be a set of transactions, each consisting of a subset X of items in I. An association rule is an implication of the form $X \rightarrow Y$, where $X \subset I$, $Y \subset I$, and $X \cap Y = \emptyset$. X is the rule's antecedent and Y is the consequent. The rule has support s in D if s percent of D's transactions contains $X \cup Y$ ($\text{support}(X \rightarrow Y) = \text{prob}(X \cup Y)$). The rule $X \rightarrow Y$ holds in D with confidence c if c percent of D's transactions that contain X, also contain Y ($\text{confidence}(X \rightarrow Y) = \text{prob}(Y/X)$). Given a transaction database D, the problem of mining association rules is to find all association rules that satisfy minimum support (minsup) and minimum confidence (minconf). Minsup and minconf are parameters of the mining algorithm. In our work, the "items" are the different components of an interaction experience, that is the different values for the features describing the situation, the assistance action, the modality, and so on.

We use the Apriori algorithm presented in Agrawal and Srikant (1994) to generate association rules from a set of user-agent interaction experiences (step 1 in Fig. 4). An interaction experience describes a unique interaction between the user and the agent, which can be initiated by any of them. The interaction records the situation or context originating it, the assistance the agent provided, the modality of the assistance, the user feedback to the assistance type and the modality (if available), and an evaluation of the interaction (success, failure, or undefined). For example, if the interaction experiences record the assistance provided by an agent when the user wants to create a tour for his/her holiday, the input would look like in Fig. 5. The first eight attributes in each line represent the situation originating the interaction, that is the different features of a tour. For simplicity we represent situations as "situation1", and so on, in Fig. 4.

Once association rules are generated, we automatically post-process the rules the Apriori algorithm generates so that we can derive useful information about the user's preferences from them. Post-processing includes: detecting the most interesting rules according to our goals (step 2), eliminating redundant rules from the set of interesting rules (step 3), eliminating contradictory rules of the remaining set (step 4), and summarizing the rules obtained building hypothesis about the user's preferences (step 5).

To filter rules (step 2 in Fig. 4), we use templates or constraints as proposed in Klementinen et al. (1994) that select those rules that are relevant to our goals. For example, we are interested in those association rules of the forms: *situation*, *assistanceaction* → *userfeedback*, *evaluation*, in the *WATSON* algorithm, and *situation, modality*, [*assistanceaction*] → *userfeedback*, *evaluation*, in the *IONWI* algorithm, where brackets mean that the attributes are optional. Rules containing other combinations of attributes are not considered. To eliminate redundant rules (step 3 in the figure), we use a subset of the pruning rules proposed in Shah et al. (1999). Basically, these pruning rules state that given the rules A, B → C and A → C, the first

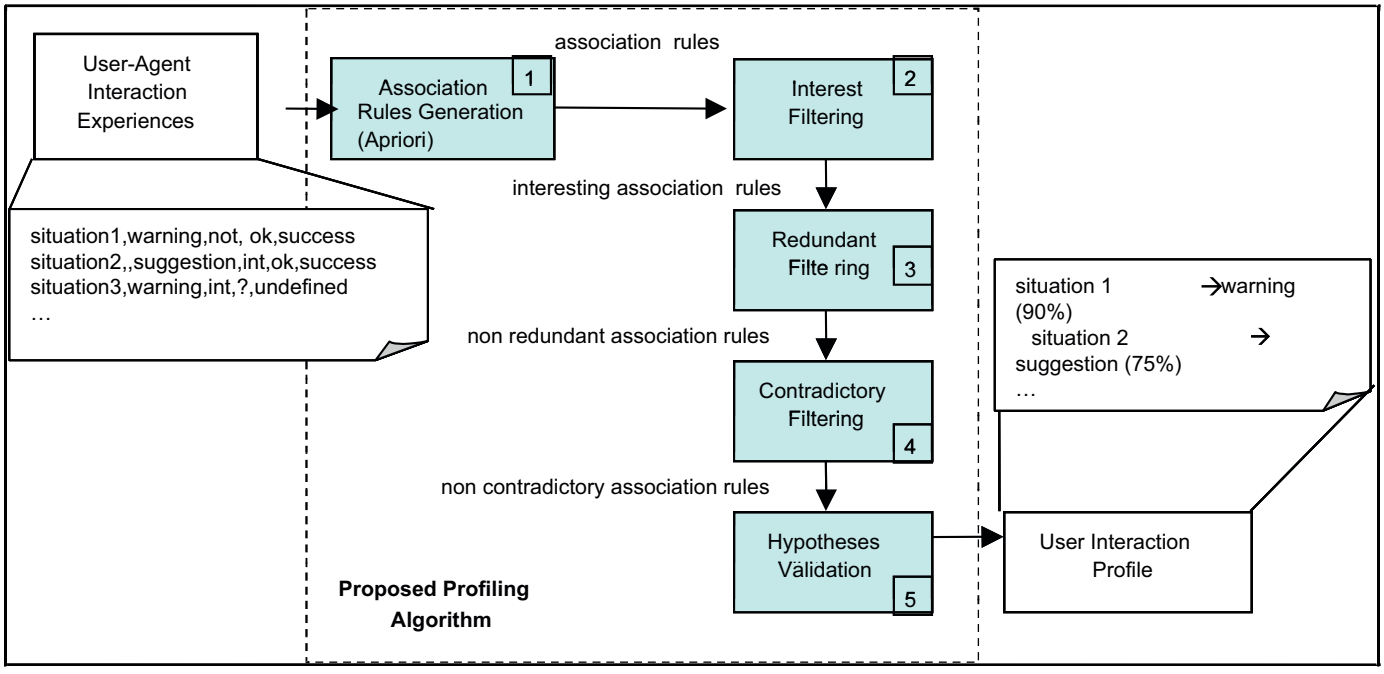

Fig. 4. Overview of the proposed profiling algorithm.

```
@relation icreatepackage

 @attribute tmodality {family,friends,couple,teenagers,honeymoon,retired,alone }
 @attribute season {summer,winter,spring,autumm }
 @attribute cost {?,cheap,moderate,expensive,veryexpensive }
 @attribute duration {?,longweekend,holiday,oneweek,twoweeks,threeweeks,onemonth,twomonths }
 @attribute accomodation {onestar,twostars,threestars,fourstars,fivestars,inn }
 @attribute transport {?,planebusiness,planetourist,planefirst,bus,train }
 @attribute place {beach,mountains,hills,country,entertainment,relax,cultural,historical }
 @attribute typeac {?,breakfast,allinclusive,onemeal,twomeals }
 @attribute agentaction {?,warning,suggestion,action,noaction }
 @attribute modality {interruption,notification }
 @attribute userfeedback {ok,close,donotinterrupt,interruptnextime,other }
 @attribute evaluation {success,failure,undefined }

 @data

 couple,summer,veryexpensive,threeweeks,threestars,planefirst,beach,breakfast,warning,notification,ok,success
 couple,summer,expensive,?,fourstars,?,beach,?,warning,notification,ok,success
 couple,summer,expensive,?,fourstars,?,beach,onemeal,suggestion,notification,ok,success
 couple,summer,expensive,longweekend,fourstars,planebusiness,beach,?,warning,notification,ok,success
 couple,summer,moderate,longweekend,fourstars,planebusiness,beach,?,suggestion,notification,ok,success
 couple,summer,moderate,twomonths,threestars,train,beach,twomeals,suggestion,notification,ok,success
 couple,summer,veryexpensive,holiday,inn,planetourist,beach,breakfast,warning,interruption,other,failure
 couple,summer,moderate,oneweek,fourstars,train,beach,onemeal,warning,notification,interruptnextime,success
 friends,winter,cheap,longweekend,onestar,?,country,allinclusive,warning,interruption,ok,success
....
```

Fig. 5. Part of an input file.

rule is redundant because it gives little extra information. Thus, it can be deleted if the two rules have similar confidence values. Similarly, given the rules $A \rightarrow B$ and $A \rightarrow B, C$, the first rule is redundant since the second consequent is more specific. Thus, the redundant rule can be deleted provided that both rules have similar confidence values.

Then, we eliminate contradictory rules (step 4 in the figure). We define a contradictory rule in *WATSON* as one indicating a different assistance action for the same situation and having a small confidence value with respect to the rule being compared. Similarly, in *IONWI*, a contradictory rule is one that indicates a different assistance modality for the same context. After pruning, we group rules by similarity and generate a hypothesis (step 5 in Fig. 4) that considers a main rule, positive evidence (redundant rules that could not be eliminated), and negative evidence (contradictory rules that could not be eliminated). The main rule is the rule in the group with the greatest support value. The summarized rules consists of a set of situation attributes in the antecedent and an assistance action (*WATSON*) or assistance modality (*IONWI*) in the consequent. Once we have a hypothesis H, the algorithm computes its certainty degree by taking into account the main rules support values and the positive and negative evidence. To compute certainty degrees, we use Eq. (3):

$$Certainty(H) = \alpha Sup(AR) + \beta \frac{\sum_{k=1}^{r} Sup(E+)_k}{\sum_{k=1}^{r+t} Sup(E)_k} - \gamma \frac{\sum_{k=1}^{t} Sup(E-)_k}{\sum_{k=1}^{r+t} Sup(E)_k} \quad (3)$$

where α, β, and γ are the weights of the termsSupAR is the main rule support, Sup(E +) is the positive evidence support, Sup(E−) is the negative evidence support, Sup(E) is the support of a rule taken as evidence (positive or negative), r is the amount of positive evidence, and t is the amount of negative evidence. We use $\alpha = 0.7$, $\beta = 0.15$ and $\gamma = 0.15$, since we considered the support of the main rule to be more important, and the negative and positive evidence as equally important. If the certainty degree of the hypothesis is greater than a given threshold value δ, it becomes part of the user profile. Otherwise, it is discarded. The value of δ was experimentally set to 0.1 for the tourism domain. More information about how this parameter can be determined can be found in Schiaffino (2004).

5. Experimental evaluation

To evaluate our proposed approach, we carried out two types of experiments. First, we studied the influence of user's preferences on the detection of user's intentions.

Then, we analyzed the precision of our profiling approach at assisting users. In a previous work (Schiaffino and Amandi, 2006), we evaluated the performance of part of our approach in the calendar management domain. In this article, we analyze the performance of agents using our proposed approach to assist users in the tourism domain. The following sections describe the experiments and the results obtained.

5.1. Influence of user's preferences on user intention detection: an example scenario

With the objective of showing the influence that the user's preferences have on the detection of the user's intention, we selected a scenario observed from the interaction of a regular user of a tourism application in which he used the application to select a tour he/she made to Iguazu during the summer to make a complaint about it and then created a travel package to visit Buenos Aires the following autumn with a moderate cost. To achieve these two goals, the user performed the following sequence of tasks: ⟨SelectTour, OpenClaimForm, EnterReason, SendClaimForm, SelectTour, SelectModality, ChangeDestination⟩. We recorded the certainty values for all intentions both in the Intention Graph containing the user's preferences information and in the same Intention Graph without context nodes (the one owned originally by the agent).

Fig. 5 a shows the evolution of the certainty values of each possible intention in the original Intention Graph without context nodes. In the first time slice, we show the a priori probabilities of each intention when the user did not perform any tasks in the application. CreatePackage is the most probable intention, while WriteMail is the least probable one. When the user performed the first task, SelectTour, the ranking remained unchanged, although there was a small increment in those intentions that contained this task (CreatePackage, InviteFriend and MakeClaim). Then the user performed OpenClaimForm, and MakeClaim became the most probable intention. With the following set of tasks performed by the user, EnterReason and SendClaimForm, the certainty of MakeClaim increases, while the other intentions certainty decreases. WriteMail certainty remained unchanged along this session because it is disconnected of the tasks performed by the user. The agent considered a threshold level with a value of 0.7 to believe in the intention pursued by the user; it predicted the first intention with the third performed task and the second intention in the third task. If we consider the number of tasks in each intention, it needed three tasks out of four to be performed to detect MakeClaim intention, and all three tasks to detect CreatePackage intention.

Fig. 5 b shows the same scenario but performed using the Intention Graph with context nodes merged. The first SelectTour task was performed when the user selected a tour he/she previously taken during the summer with destination Iguazu and with a high cost. We can see that the certainty for MakeClaim is higher only with the first task performed (which corresponds to the second time slice in the plot). We can also see that the other intentions noticeably decreased their certainty values. The second tour selected by the user had destination Buenos Aires, was planned for the following autumn with a moderate cost. We can observe that with the mere selection of the tour, CreatePackage intention could be predicted.

A remarkable observation in the scenario shown is that the incorporation of the user's preferences allows an earlier distinction of the actual intention of the user. Another interesting fact that can be appreciated in Fig. 5 is that the certainty of finished intentions gradually decrements to its original value, as happens with MakeClaim intention. This is due to the fading function used by the intention graph that gradually decrements by a fixed constant to the strength of the evidence on the performed tasks. Similar results were observed with other scenarios.

5.2. Evaluation of agents' decision making

To evaluate the precision of our interaction profiling approach at assisting users, we studied the number of correct assistance actions executed by agents assisting users with a calendar application, where the correctness is determined by the user through explicit and implicit feedback. To do this, we used a precision metric that measures a personal agent's ability to accurately assist a user, which is shown in Eq. (4). This equation is an adaptation of the precision metric proposed by Brown et al. (1998).

$$Precision = \frac{\text{number of correct assistance actions}}{\text{number of assistance actions}} \quad (4)$$

We used this metric to evaluate the agent's performance in deciding between a warning, a suggestion, or an action on the user's behalf; and between an interruption or a notification. For each problem situation, we compared the number of correct assistance actions against the total number of assistance actions the agent executed.[2] To carry out the experiments, we used 30 data sets containing user-agent interaction experiences in the tourism domain. This datasets correspond to the interaction between 15 users and an agent assisting them in different contexts. The users had different roles, some acted as clients and others as employees of the travel agency. Each database record contains attributes that describe the problem situation (or the situation originating the interaction), the assistance action the agent executed, the user feedback, and the user's evaluation of the interaction experience.

[2]We considered an assistance action correct if it was the action the user expected in a given situation.

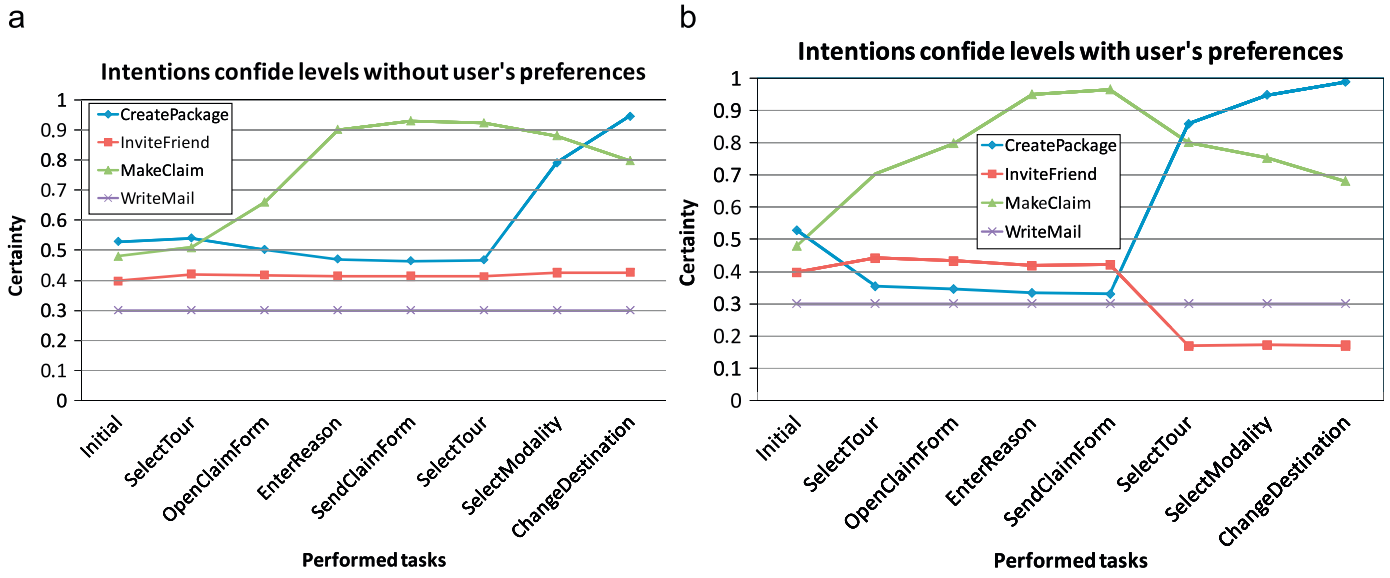

Fig. 6. Experimental results.

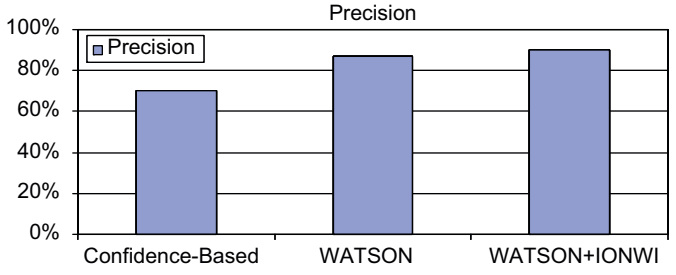

Fig. 7. Precision of the proposed profiling methods.

The data sets contained anywhere from 20 to 45 user-agent interaction experiences.[3] We analyzed two assistance situations in the tourism domain: the user creates a tour (in order to ask travel agencies for information about it), where the situation information consists of the modality of the tour, the season, the cost, the duration, type of accommodation, means of transport, place and meals included; and the user, in this case an employee of a travel agency, receives a complaint from a client, where the situation description comprises the customer type, the requested compensation or refund if there is any, and the information of the tour originating the complaint (Fig. 6).

Fig. 7 compares the precision values for the three approaches we evaluated. These three approaches are: the confidence-based algorithm used traditionally by interface agents; deciding the assistance action with *WATSON* (without considering interruption preferences); deciding the assistance action with *WATSON+IONWI*. The two last approaches considered user intention recognition. The values shown were obtained by averaging the precision for the different datasets belonging to the different users. We used percentage values because the number of user-agent interactions varied from one user to another. As the figure shows, our approach had a higher overall percentage of correct assistance actions or interactions than the confidence-based algorithm for the two datasets.

In previous experiments in the calendar management domain, we have obtained precision values of 86% and 91% for correct actions and interactions, which were higher than the precision of the confidence-based algorithm, 67%. In the tourism domain the values we obtained were rather similar: 70% for confidence-based, 87% for correct actions and 90% for correct assistance actions and modalities. However, we observed a phenomenon that has to be pointed out. In the calendar domain, the number of user-agent interactions we worked with datasets bigger (80 instances in average) than in a tourism application. This may be because, in our experiments, users schedule events more frequently than they book or request information about tours, in the same period of time. If we have a small number of interaction instances in a dataset (e.g. 20–25), there might be few repeated interactions and thus the profiling algorithm will learn only those that are most frequent. To deal with this problem, some parameters of *WATSON* and *IONWI* were adjusted for the tourism domain. The values of the parameters used in these experiments were: $minsup = 0,1$; $minconf = 0,6$ (0,8 for the calendar domain); $\delta = 0,1$ (0,2 for the calendar domain); $\tau_1 = 0,1$; $\tau_2 = 0,8$; $\tau_3 = 0,3$; $do{-}it = 0,8$; $tell{-}me = 0,3$. The first three parameters were set experimentally (Schiaffino, 2004) and the last five were set according to Maes (1994).

6. Related works

Some algorithms have been proposed to decide which action an agent should execute next. These algorithms

[3]The datasets are available at: http://users.exa.unicen.edu.ar/~sschia/personalization.html

adopt mainly one of two approaches: some use decision and utility theory (Fleming and Cohen, 2001; Horvitz, 1999), and others use confidence values attached to different actions (Kozierok and Maes, 1993; Maes, 1994). However, these works do not consider a user's interaction preferences, the possibility of providing different types of assistance, or the particularities of the situation at hand.

Our work is related to those works that study the etiquette of human–computer relationships (Miller, 2004), since learning when to interrupt a user, detecting a user's intentions, and deciding how to best assist him can be considered as part of this etiquette. Considering these issues within interface agent development, and particularly within user profiles, is novel.

Regarding interruptions, there have been numerous studies exploring them in a general way, mainly in the Human Computer Interaction (HCI) area. These studies revealed that the disruptiveness of an interruption is related to several factors, including complexity of the primary task and/or interrupting task, similarity of the two tasks (Gillie and Broadbent, 1989), whether the interruption is relevant to the primary task (Czerwinski et al., 2000b), stage of the primary task when the interruption occurs (Czerwinski et al., 2000a), management strategies for handling interruptions (Mc Farlane, 1999), effects of interruptions (Bailey and Iqbal, 2008; Bailey and Konstan, 2006), and modalities of the primary task and the interruption (Arroyo et al., 2002). Most of these studies are related to instant messaging systems (Garrett and Danziger, 2008), they have been rarely considered for interface agent development.

With respect to agents using plan recognition to detect a user's plans and intentions, some works have been done in this direction (Horvitz et al., 1998; Rich et al., 2001; Duong et al., 2006; Geib and Goldman, 2005; Philipose et al., 2004; Madani et al., 2009). However, most of them do not consider the user's preferences within the process, and those who consider them have limitations. For example, Bauer (1996) looked into the acquisition of user preferences for plan recognition for an email system. He consider repeated patterns in the user behavior as preferences and used ID3 to learn classes of situations based on the user's actions. With Bauer's technique it is possible to learn, for example, that a particular user will delete his email with 80% certainty, unless the email is from his manager, in which case he saves it 90% of the times. The problem with this approach compared to ours is that only one intention can be ascribed for a given set of attributes, while we assign a degree of certainty to each possible intention.

Finally, the work reported in this article can be considered an extension of a previous work (Schiaffino and Amandi, 2006). In that work we studied how the agent could choose the best type of assistance for a given user in a given situation. We analyzed the precision of the algorithms proposed in the calendar management domain. Now, we not only consider how an agent can choose the best assistance action for a certain user, but also how it can determine the user's intention first in order to provide proper and timely assistance. We describe a plan recognition approach to detect the user's intentions and we combine this approach with the algorithms presented in our previous work to build the user profile. We also study the performance of our proposal in the tourism domain and compare it with the one obtained for the calendar domain.

7. Conclusions

In this article, we presented an approach to personalize the interaction with users that considers the users' intentions and the users' interaction preferences. To detect a user's intentions we propose a plan recognition approach, which considers the user's preferences to allow an earlier detection. To learn a user's interaction preferences, we proposed two profiling methods.

We consider that our work contributes both to the interface agents and human–computer interaction areas. Our user profiling and decision making approaches enhance an interface agent's capabilities. Agent developers can use our results to build interface agents that can adapt to users' expectations and preferences regarding user-agent interaction. Our approach enables agents to personalize their user interactions by learning the type of assistance users' need in different contexts and by learning how to provide this assistance without annoying users.

We evaluated our proposal with promising results in a calendar management application (Schiaffino and Amandi, 2006) and in a tourism application. As a future work, we will carry out further experiments in these and other domains. Particularly, we will try to evaluate our proposal with bigger datasets, that is, containing more user-agent interactions. In addition, we are enriching now the information contained in the assistance situations by using ontologies to model users' context. A related work of our research group in this direction is Eyharabide et al. (2009), in which personalized assistance is provided to students depending on their context in an e-learning system.

Acknowledgment

This work was partially supported by ANPCyT, Argentina, through PICT 2004 Project no. 20178.

References

Agrawal, R., Srikant, R., 1994. Fast algorithms for mining association rules. In: Proceedings of the 20th International Conference on Very Large Data Bases (VLDB 94), pp. 487–499.

Arroyo, E., Selker, T., Stouffs, A., 2002. Interruptions as multimodal outputs: which are the less disruptive? In: IEEE International Conference on Multimodal Interfaces, ICMI 02, pp. 479–483.

Bailey, B.P., Konstan, J.A., 2006. On the need for attention-aware systems: measuring effects of interruption on task performance, error rate, and affective state. Computers in Human Behavior 22 (4), 685–708.

Bailey, B.P., Iqbal, S.T., 2008. Understanding changes in mental workload during execution of goal-directed tasks and its application for

interruption management. ACM Transactions on Computer–Human Interaction 14 (4), 1–28.

Bauer, M., 1996. Acquisition of user preferences for plan recognition. In: Proceedings of the Fifth International Conference on User Modeling, pp. 105–112.

Brown, S.M., Santos Jr., E., Banks, S.B., Oxley, M., 1998. Using explicit requirements and metrics for interface agent user model correction. In: Proceedings of the Second International Conference on Autonomous Agents (Agents '98). ACM Press, New York, pp. 1–7.

Cooper, G.F., Herskovits, E., 1992. A bayesian method for the induction of probabilistic networks from data. Machine Learning 9 (4), 309–347.

Czerwinski, M., Cutrell, E., Horvitz, E., 2000a. Instant messaging: effects of relevance and timing. In: People and Computers XIV: Proceedings of HCI 2000, pp. 71–76.

Czerwinski, M., Cutrell, E., Horvitz, E., 2000b. Instant messaging and interruption: influence of task type on performance. In: Proceedings OZCHI.

Duong, T.V., Phung, D.Q., Bui, H.H., Venkatesh, S., 2006. Human behavior recognition with generic exponential family duration modeling in the hidden semi-Markov model. In: International Conference on Pattern Recognition, vol. 3, pp. 202–207.

Eyharabide, M.V., Gasparini, I., Schiaffino, S., Amandi, A., 2009. Personalized e-learning environments: considering students' contexts. In: Proceedings WCCE 2009—IFIP World Conference on Computers in Education—Bento Goncalves, Brazil, July, in press.

Fleming, M., Cohen, R., 2001. A user modeling approach to determining system initiative in mixed-initiative AI systems. In: Proceedings of the 18th International Conference On User Modeling, pp. 54–63.

Garrett, R.K., Danziger, J.N., 2008. IM = interruption management? Instant messaging and disruption in the workplace. Journal of Computer-Mediated Communication 13 (1), 23–42.

Geib, C., Goldman, R., 2005. Partial observability and probabilistic plan/goal recognition. In: IJCAI-05 Workshop on Modeling Others from Observations, Edinburgh, Scotland.

Gillie, T., Broadbent, D., 1989. What makes interruptions disruptive? A study of length, similarity and complexity. Psychological Research 50, 243–250.

Horvitz, E., Heckerman, D., Hovel, D., Rommelse, K., 1998. The Lumière Project: Bayesian user modeling for inferring the goals and needs of software users. In: Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence, pp. 256–265.

Horvitz, E., 1999. Principles of mixed-initiative user interfaces. In:Proceedings ACM Conference Human Factors in Computing Systems (CHI 99), pp. 159–166.

Jensen, F., 2001. Bayesian Networks and Decision Graphs. Springer, New York.

Kautz, H., 1991. A Formal Theory of Plan Recognition and its Implementation, Reasoning About Plans. Morgan Kaufmann Publishers, Los Altos, CA, pp. 69–125.

Klementinen, M., Mannila, H., Ronkainen, P., Toivonen, H., Verkamo, A.I., 1994. Finding interesting rules from large sets of discovered association rules. In: Third International Conference on Information and Knowledge Management, pp. 401–407.

Kozierok, R., Maes, P. A learning interface agent for scheduling meetings. In ACM-SIGCHI International Workshop on Intelligent User Interfaces, pp. 81–93.

Lauritzen, S.L., 1995. The em algorithm for graphical association models with missing data. Computational Statistics and Data Analysis 19 (2), 191–201.

Liu, Y., Cai, J., Yin, J., Wai-Chee Fu, A., 2007. Clustering massive text data streams by semantic smoothing model. In: Advanced Data Mining and Applications, Lecture Notes in Computer Science, vol. 4632, pp. 389–400.

Maes, P., 1994. Agents that reduce work and information overload. Communications of the ACM 37 (7), 30–40.

Mc Farlane, D., 1999. Coordinating the interruption of people in human–computer interaction. In: Human–Computer Interaction, INTERACT 1999, pp. 295–303.

Madani, O., Bui, H. Yeh, E., 2009. Efficient online learning and prediction of users' desktop actions. In: Proceedings of IJCAI-09.

Miller, C., 2004. Human–computer etiquette: managing expectations with intentional agents. Communications of the ACM 47 (4), 31–34.

Philipose, M., Fishkin, K.P., Perkowitz, M., Patterson, D.J., Fox, D., Kautz, H., Hahnel, D., 2004. Inferring activities from interactions with objects. Pervasive Computing Magazine 3 (4), 10–17.

Rich, C., Sidner, C., Leash, N., 2001. Collagen: applying collaborative discourse theory to human–computer interaction. Artificial Intelligence 22 (4), 15–25.

Rudman, P., Zajizek, M., 2006. Autonomous agent as helper—helpful or annoying? In: Proceedings 2006 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT 2006 Main Conference Proceedings), IAT'06 2007, pp. 170–176.

Schiaffino, S., Amandi, A., 2006. Polite personal agents. IEEE Intelligent Systems 21 (1), 12–19.

Schiaffino, S., Amandi, A., 2004. User—interface agent interaction: personalization issues. International Journal of Human Computer Studies 60 (1), 129–148.

Schiaffino, S., 2004. Personalization of user—interface agent interaction. Ph.D. Thesis, Fac. Cs. Exactas, UNCPBA, Tandil, Argentina (Chapter 7).

Serenko, A., Bontis, N., Detlor, B., 2007. End-user adoption of animated interface agents in everyday work applications. Behaviour and Information Technology 26 (2), 119–132.

Shah, D., Lakshmanan, L., Ramamrithnanm, K., Sudarshan, S., 1999. Interestingness and pruning of mined patterns. In: Proceedings Workshop Research Issues in Data Mining and Knowledge Discovery, ACM Press, New York.