Agents Detecting User's Intentions

Marcelo G. Armentano and Analía Amandi

ISISTAN Research Institute. Facultad de Ciencias Exactas, UNICEN.
Campus Universitario, (B7001BBO) Tandil, Buenos Aires, Argentina
{marmenta, amandi}@exa.unicen.edu.ar

Abstract. As computers are used for more tasks and become integrated with more services, users will need help dealing with the information and work overload. Interface agents radically change the style of human-computer interaction. The user delegates a range of tasks to personalized agents that can act on the user's behalf. In this way, the agent gradually learns how to better assist the user. This paper presents an approach that shows how an interface agent can detect intervention situation to cooperate or advise the user by observing user's activities on an application. The need of understanding what the user is doing lies in the fact that the agent has to act always in the context of the user intention. Acting in this way, the agent would not bother the user in an improper moment and the user can always feel that it has the control of the application. The approach presented is a portion of an own software architecture, named AGUSINA, which defines specific components to manage Agent-User interactions.

1 Introduction

Computers had become a useful tool for a variety of daily tasks. Information retrieval, e-mail, e-commerce, home banking, social interaction, and entertainment, are only examples of the wide diversity of computers applications. As the users get more used to the computers, they became more demanding on what they can obtain from computer applications.

The currently dominant interaction metaphor of direct manipulation [1] requires the user to initiate all tasks explicitly and to monitor all events. But the more users expect from applications, the more complex these became. So, a new kind of interaction is needed.

Techniques from the field of Artificial Intelligence, in particular so-called *autonomous agents*, can be used to implement a complementary style of interaction, which has been referred to as indirect management [2]. Instead of user-initiated interaction via commands and/or direct manipulation, the user is engaged in a cooperative process in which human and computer agents both initiate communication, monitor events and perform tasks.

Interface agents are computer programs that employ Artificial Intelligence techniques to provide active assistance to a user with computer-based tasks. Agents radically change the current user experience, through the metaphor that an agent can act as a personal assistant. The agent acquires its competence by learning from the user as well as from agents assisting other users [3].

This approach follows the metaphor of a personal assistant who is collaborating with the user in the work environment. The assistant becomes progressively more helpful, as it learns the user interests, habits and preferences and will act according such information. Interface agents have the ability of receiving orders from human users and answer in a personalized way. Moreover, these agents can show a mixed-initiative interaction with their users, detecting when they could help their users and either suggest an action or directly act for helping in the detected situation.

We can suppose that we want to build an agent for assisting users in the scheduling of an agenda. Thinking in to achieve this goal, agents need to observe the user using the application to detect each gesture performed on the calendar interface and to process them for both improving the user profile and detecting user intentions and intervention situations. Once the agent has detected intentions and intervention situations, it can answer to user request in the context of the user's intentions and moreover can intervene with suggestions, warnings or actions in the context of the detected situations. All these communications with the users are influenced by the user profile built until that moment.

As we can observe in the previous example, for that agent we need to specify how to know each gesture of the user, which is the interpretation of each of those gestures and in which situation the agents could intervene. All of this information depends on the user interface of each application.

These requirements affect the design of the functionality of the agent relative to detect when a user could be helped. We have developed AGUSINA, a specific interface agent software architecture to manage Agent-User interactions that defines explicit components for our goal.

This paper will focus on the part of AGUSINA related to the observation of the user interaction with application to detect his/her intentions in the sense of what he/she is trying to do. This will allow the agent to find a way to cooperate with the user. Furthermore, we will have in mind the idea of connecting *any* application with interface agents that assist users working on them.
The presentation is structured as follows. Section 2 discusses the implications of building interface agents for assisting user working on standard application. Section 3 presents the proposed solution for the problem of the observing a user using an application to detect his/her intentions and to improve the user profile. Finally, in Section 4 we present the work related to that presented here, and in Section 5 we sum up the presented solution.

2 Interface Agents

Our approach to add an *Interface Agent* [3] to a conventional direct manipulation graphical user interface is shown in Fig. 1. This approach intends to imitate the relationships held between a boss and his/her assistant that are working together in some specific task involving a shared artefact, as a manager and a secretary scheduling a meeting, or a student and a tutor learning a lesson. In this approach, an interface agent replaces the second human. Both the user and the agent can act on the application (the shared artefact), the boss can ask the assistant for help, the assistant can give advice to his/her boss, and both can observe each other's actions. The level of subordination between the boss and the assistant can vary from a purely ordering relation to a co-worker relation (as that shown at [4]).

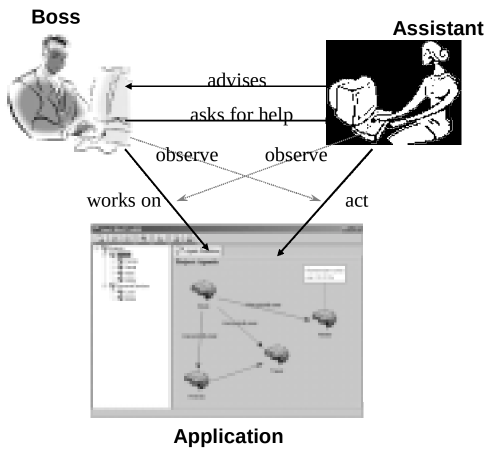

Fig. 1. Our approach to Interface Agents.

We developed a software architecture, named AGUSINA (AGent-User INteraction Architecture), that defines specific components to manage agent-user interactions. Fig. 2 presents an overview of AGUSINA, showing the main components of the agent and their relationships.
As can be noted in Fig. 2 there is the clear separation between the agent and the application while the relationships shown in Fig. 1 still hold. The architecture has four main components. The *Interaction Manager* component is used by the user to ask the agent for advice, by the agent to reports its actions and communicates them to the user, and to answer user's requests. This component is also in charged of observing user activities on the application, and communicating them to the Intention Manager and Intervention Manager components. The *Intention Manager* component keeps information of all the possible tasks that the user can perform on the application. This component uses this information to detect patterns of behavior that may allow the agent to detect the user intention. When the Intention Manager component has an appropriate level of confidence of the user's intention detected, it communicates this intention to the Intervention Manager component. The *Intervention Manager* component specifies the triggers of the application for intervention of the agent, and different types of interventions that may be appropriated in different contexts. The Intervention Manager component will be activated by user gestures on the application (intervention triggers will be thrown) or by the agent, when it detects the user intention. The *Profile Manager* component, will keep information about user's habits and preferences.

This information is used by both the Intervention Manager and Intention Manager components to collaborate with the user in a personalized way.

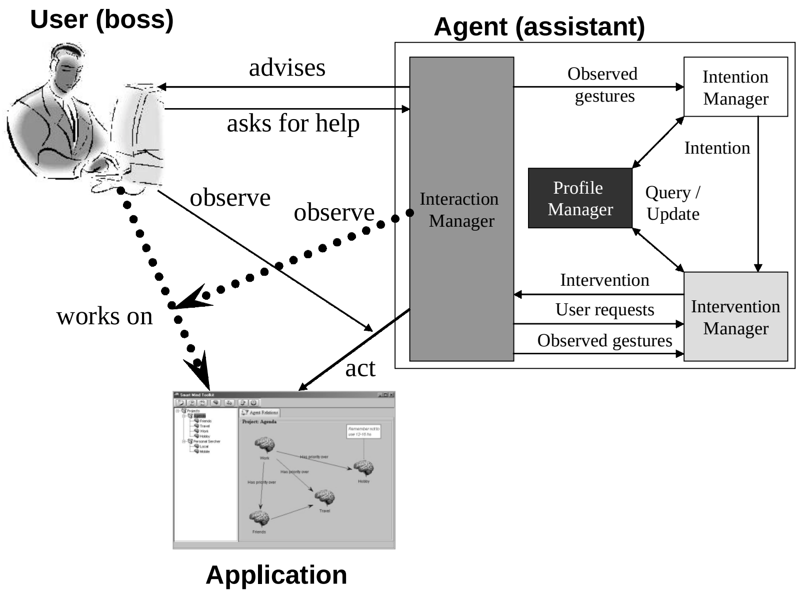

Fig. 2. AGUSINA overview.

For example, in an agenda application, the user will can, at least, schedule events. To do this task, he/she will need to complete some form indicating the time and place, and perhaps the participants of the event. All of these actions that have to be performed to accomplish the task will be kept by the Intention Manager, so that the agent can identify the task the user is performing in any given moment. The Interaction Manager component, in this example, will have a trigger associated to the problem that might arise if the user selects for the event a previously selected slot of time for other event. Furthermore, this component will have to choose a way of interaction with the user, and then, one of the possible interactions of the selected type. For example, the agent can propose a new slot of time (according to user preferences) for the new event, suggest canceling the *old* event, or only warn the user about the problem. Finally, the Profile Manager will be use to act according to the user preferences and habits. For example, if the new event of the last example is "playing tennis with a friend", and the already scheduled event is a "meeting with the boss", the agent shouldn't suggest canceling the meeting (unless the agent knows that the user prefer playing tennis than meeting the boss!). Also, if the user is scheduling a meeting at an unusual time of the day (such as 03:00 a.m. for a football match), the agent can intervene to inform him/her about this "abnormal situation".
In this paper we will focus on the description of user's tasks on the application, and on the detection of user's intention. These issues are highlighted in Fig. 2 with a different line style (•••➤).

3 Observation and Analysis of User Activities

The ability to reason about user tasks on an application is a key point in the developing of any intelligent user interface [5]. If the agent is able to recognize what task the user is performing, it can act to cooperate. The understanding of user's tasks provides a context to figure out future actions and try to be aware of what the user is most likely to do next. With this information, the agent can wait for the opportunity of collaborating with the user.
For example, if the agent observes that the user is scheduling work meetings, it will be *out of context* to remind him/her a friend's birthday. However, the agent might suggest, for example, canceling a football match if the slot of time assigned to it superpose with a work meeting, or offer sending a mail to each participant of a given work meeting.

The *Intention manager* is the component of AGUSINA software architecture whose main function is to keep the track of the user while interacting with the application. To achieve this goal, this component need knowing the set of tasks the user can perform, observe his/her gestures and determine which task he/she is carrying out in any given moment. With this information available, the agent would be able to recognize the user's intention (i.e. what task he/she is more probably to do next) and try to cooperate. Fig. 3 shows an explosion of Fig. 2, where different colors distinguish between the top level components shown in Fig. 2, and arcs represent flows of information.
We will limit the explanation of the architecture to those components related to the detection of the user intention.

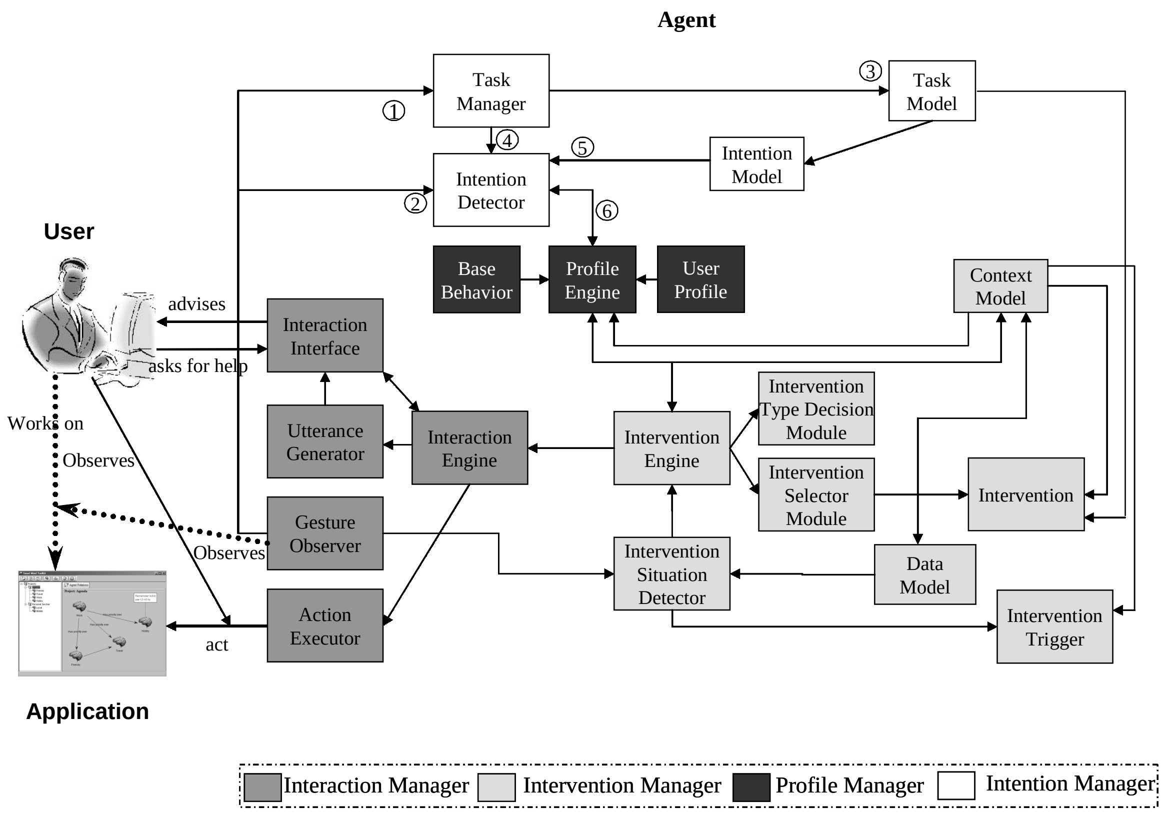

Fig. 3. Explosion of AGUSINA components.

The first step in detecting situations in which the agent can intervene to collaborate with the user or to inform him/her about problems or special situations that may arise is *observation*. The *Gesture Observer* component of the architecture is engaged to this job. Every gesture of the user that the designer of the agent specified as interesting is detected by the *Gesture Observer* and communicated to the *Task Manager* (marked as 1 in Fig. 3) and to the *Intention Detector* (marked as 2 in Fig. 3). The first component will use this information to know in every moment what task the user is performing, and what tasks are expected to be performed to complete the global task. The second component will use the detected gesture, along with the information provided by the Task Manager to detect the user intention.

3.1 Detection of User's Current Task: the Task Model

For specifying the possible gestures on the application user interface we use a *Task Model*. A task defines how the user can reach a goal in a specific user interface of an application. The goal is either a desired modification of the state of a system or a query to it.
Given that Graphical User Interfaces (GUI) typically have the objective of supporting a particular set of human tasks, the first job of a designer that use them is to formalize this intentions in an explicit task model. An explicit task model can be used to control the behavior of a software agent that helps a user perform tasks using a GUI [7].

The best known techniques for task modeling are based on hierarchical task analysis. These general models contain information about the structuring of the tasks into subtasks and about the order of their execution. This is a simplified view upon the tasks which reflects only part of their multiple dependencies and the real task situation. Due to this inappropriate reflection of reality several enhancements have been defined. The great variety with respect to techniques and notations used shows that currently there is no agreement on how a task should be described adequately and completely [9].
However, there is agreement about the central role of the following information:

- The structuring of tasks into subtasks,
- Pre and postconditions for tasks,
- Temporal relations between tasks, and
- Objects used within a task context.

There are many different task model representations. In this work, we use a variation of ConcurTaskTrees notation [8]. Each task of the model has a tree-like structure where leaves are simple tasks and internal nodes represent tasks composed by other (simple or composed) tasks. Both types of tasks might be associated to user gestures, such as a click on a button,
Fig. 4 shows a task model for the agenda's task example of adding a meeting. Different kinds of nodes indicate different kind of tasks: application tasks (task performed completely by the system), interaction tasks (user gestures over the application), and abstract tasks (tasks that require complex actions, and their performance does not completely fall into one of the previous cases.). The relation between nodes in the same level denotes different kinds of temporal restrictions in the execution order of the tasks.

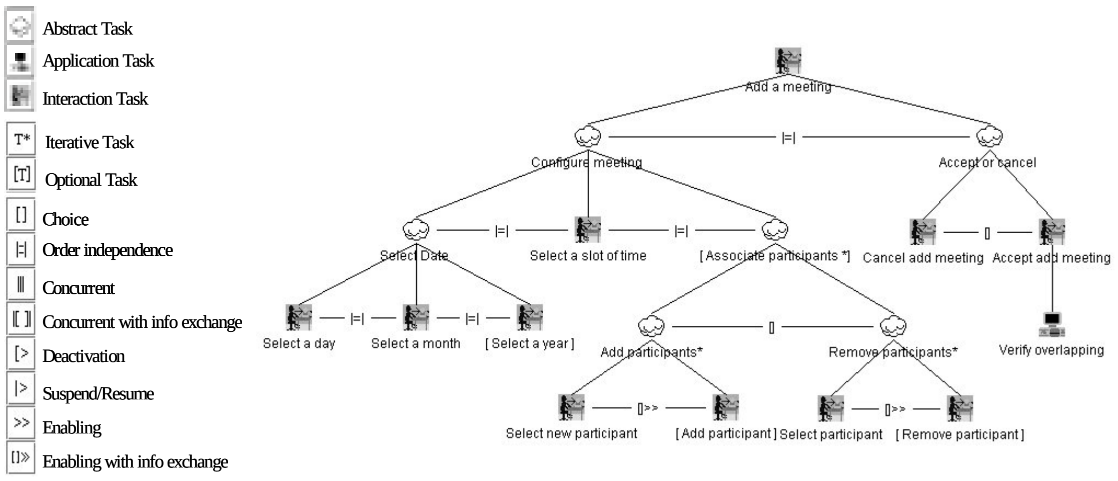

Fig. 4. Task Model for adding a meeting in an agenda application

Although the word *task* refers to those activities that can be performed within the application, for us it also includes smaller and more concrete actions, such as mouse click or a key pressed. That is to say that every concrete or abstract action will be modeled as a *task*. We define a *Basic Action* as a task of the task model that can be performed by the agent on the application or a task that the agent can suggest the user to perform. Then, the Task Model will contain a set of Basic Actions and other tasks, along with temporal constraints between all of them. For example, in an agenda application example shown in Fig. 4, we have a task named "Add a meeting", whose subtasks (or steps of the task) will be "Select a date", "Select a slot of time" and "Associate participants". Note that we need to introduce an abstract task named "Configure meeting" to isolate the main functionality of the task from the buttons for accepting or canceling it. This abstract task is necessary because if we start any of its subtasks, we have to finish it before accepting or canceling the root task. To select a date, the user has to select a day, a month and maybe a year. All of these subtasks will be probably associated directly with an interface widget, such as text boxes or lists. The "Select a slot of time" task could merely be related to the selection of an item of a list. Finally, the "Associate participants" task may be composed of two abstract iterative subtasks: "Add participants" and "Remove participants". Both tasks have two interaction subtasks: selecting a participant and adding or removing it to the meeting. The "Select participant" task enables the second task, passing it the selected participant as a parameter. "Add participant" and "Remove participant" are optional so that the user can freely select participants from the list without adding or removing them, just in case he/she selects a

wrong one. The parent task “Associate participants” is also iterative. This fact will allow the user to alternate adding and removing participants to the meeting.
The finalization of the root task “Add meeting” will be detected when the user press the “Accept” or “Cancel” button, both associated to respective tasks “Accept add meeting” and “Cancel add meeting”. In the first case, the application will verify that the meeting does not overlap any existing one. We model this using an application task named “Verify overlapping”.
We have to consider also restrictions in the order the subtasks are performed. The three top level subtasks may be executed in any order, but, for example, the user cannot perform “Add participant” (that is clicking on the “Add” button) before performing “Select new participant” first.
Having presented an example of task model, we will return to the sequence followed by the agent when it detects a user’s gesture. When the *Task Manager* receives a notification from the *Gesture Observer* of a gesture over a component, it inspects the *Task Model* (marked as 3 in Fig. 3) looking for the associated task and verifies that it is *expected*, that is, that it follows the logic execution of the task he/she was performing according to order restrictions of its subtasks. Then, we define an *active task* as a task from the task model that is consistent with user actions, in the sense that users follow a way in which their intention can be discovered. For example, the action of clicking on the “Add” button without selecting a participant first is not expected, because the task model specifies that the last task will be active, only when the first task is completed.
We define an *execution cycle* as a sequence of user gestures starting either when the application is loaded or when the user completes the previous top level task. At the beginning of an execution cycle, all the tasks are active, because any given event was triggered. The *Task Manager* may consider the way that a user action can be seen as a contribution to one candidate user activity that represents a user intention.
We can interpret any user action on the application by four different ways:

- It is a final action of an active task: current goal is achieved.
- It is the transition to a following task of an active task: the active task goes a step forward. If there is any active task that is not waiting for the last user action, it deactivates.
- Identifies the task the user is carrying out: there is only one active task remaining, therefore the agent is certain of the user’s goal.
- Identifies a parameter of current task. For example, the day of the meeting is a parameter of the “Select date” task.

If no one of last situations is given, the agent concludes that the current action initiates an interruption [6], that is, a non-expected action. The occurrence of interruptions may be due to actual changes on the task the user were carrying out or due to an incomplete task description that does not include the current act even though it ought to. We assume that, in general, the agent's knowledge about the task model will be complete and therefore, the agent should handle a non-expected action as a change in the user goal.
The agent may also consider that the interruption is *definitive* or *transitory*. In the first case, the agent will discard all current active tasks (since the current goal will not be resumed) and will use the task model to get a new set of active tasks based on the last user action. On the other hand, if the agent believes that the interruption is transitory, all active tasks will be kept active with a flag indicating that it could be resumed or reminded by the agent later on (when the new task is completed, or when the agent consider appropriate).

3.2 Detection of User’s Intention: the Intention Model

Presuming that the task performed by the user is valid, the agent will consider the set of possible intentions associated to it to find a way to collaborate with the user. So, the *Intention Detector* asks the *Task Manager* about the task being performed by the user (marked as 4 in Fig. 3), and uses this information along with the *Intention Model* (marked as 5 in Fig. 3) to select the candidate intentions. Then, it queries the *Profile Engine* about preferences or habits of the user about these intentions (marked as 6 in Fig. 3).
The Intention Model is a set of *intention graphs*. Each intention graph represents a context of execution, represented by a Bayesian Network [12]. The context, at this point, is view as the sequence of tasks that the user has performed recently (information kept by the Intention Detector), and will influence the confidence of any given intention.
Bayesian networks are a probabilistic knowledge representation used to represent uncertain information. The directed acyclic graph structure of the network contains representations of both the conditional dependencies and independencies between elements of the problem domain. The knowledge is represented by nodes called random variables and arcs representing the causal relationships between

variables. The strengths of the relationships are described using parameters encoded in conditional probability tables.
In an intention graph, the nodes of the network are tasks of the task model, and the directed relations between them represents the fact that one task is followed by the other task. Then if there are arcs form task *A* to task *B*, and from task *A* to task *C*, we will be able to compute the likelihoods that the user will perform task *C* or task *B given that* he previously performed task *A*, and all the previous tasks included in the context. We call this conditional value for each task, *confidence level*. Each task within an intention graph will have related a set of possible intentions, each with an associated confidence level computed using the belief update mechanism, given the evidence tasks. These confidence levels, along with the information kept by the Profile Engine, allow us to rank the possible "following tasks" and select the intention graph the user is pursuing. Fig. 5 shows an example of an intention graph and the variation of the confidence of possible intentions as the user goes on performing tasks.

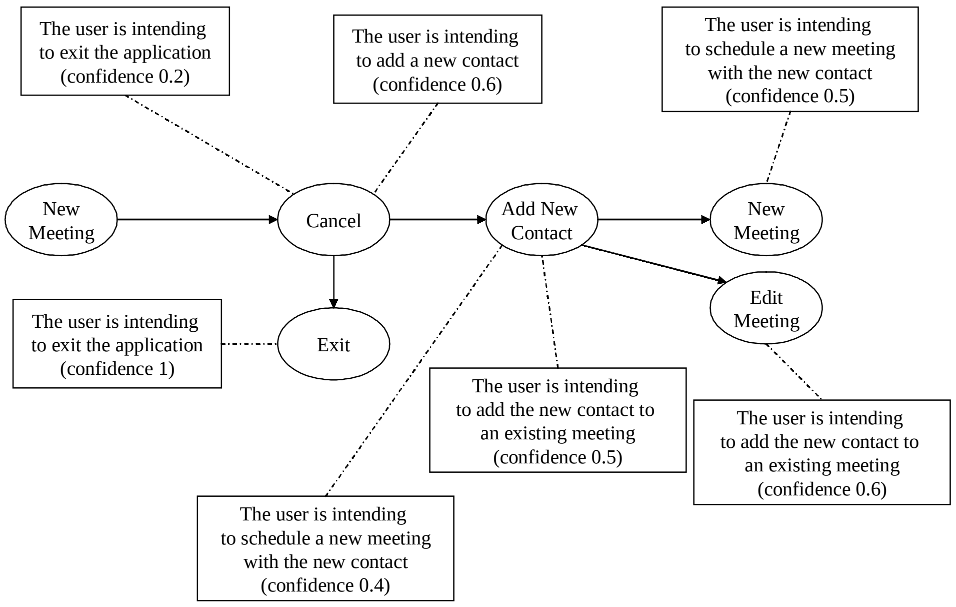
The user is intending to exit the application (confidence 0.2)
The user is intending to add a new contact (confidence 0.6)
The user is intending to schedule a new meeting with the new contact (confidence 0.5)
New Meeting
Cancel
Add New Contact
New Meeting
Edit Meeting
The user is intending to exit the application (confidence 1)
Exit
The user is intending to add the new contact to an existing meeting (confidence 0.5)
The user is intending to add the new contact to an existing meeting (confidence 0.6)
The user is intending to schedule a new meeting with the new contact (confidence 0.4)

Fig. 5. Intention Graph example

If the user presses the "Cancel" button after scheduling a new meeting, the agent may consider that the user will exit the application or that he/she will add new contacts. Note that the confidence of the first alternative is lower. This denotes the fact that the user has more probability of continue using the application after canceling a task than exiting, based on previous experiences. If then the user press the "Add new contact" button, the confidence of the second plan will be increased, and the agent has two new plans associated to considerate: the new contact can be used to schedule a new meeting, or to be added to a previous one. If none of these plans is followed, both confidences will be decreased. Otherwise, if the user selects an existing meeting for edition, for example, the agent will increase the probability of that plan.
As the user prefers one task in an intention graph instead of others, the confidence level of the selected task should be increased for future interactions. The learning capability of Bayesian networks allows us to represent this fact.
One of the criticisms of Bayesian networks is how they are constructed. The construction of Bayesian networks usually proceeds in one of two ways: "hand-coded" or "machine-coded". The first alternative means a Bayesian network is constructed by a knowledge engineer to capture the structure of the network, while the second, on the other hand, are generated by using previously captured data or data acquired as users interact with the system.
Currently, both the intention graphs and the initial probabilities associated to each node are statically defined at design time. Future versions of AGUSINA will include dynamically acquisition and updates of these values.

4 Related Work

There are an increasing number of projects in the interface agents community that are trying to integrate agents with conventional applications, though there is still no definitive methodology for this.
Collagen [6] is an application-independent collaboration manager based on the SharedPlan theory of task-oriented collaboration discourse [10]. In [6] the authors provide a discussion of user/agent collaboration issues, including initiative, dialogue and turn taking and in [11] they go further in the plan recognition system employed.
Bayesian networks [12] have become a popular representation for reasoning under uncertainty as they integrate a graphical representation of casual relationships with a sound Bayesian foundation. The Core Interface Agent (CIA) Architecture [13] is a multi-agent system composed of an interface agent and a collection of correction adaptation agents. The purpose of providing assistance to the user is accomplished my maintaining an accurate model of the user's interaction with the target system environment. The Bayesian user model is used to ascribe the user's intent. The task of ascribing user intent is delegated to the interface agent component of architecture, while continual adaptation of the interface agent's user model is a task shared by the interface agent and the collection of correction adaptation agents.
The system described in [14] uses a plan hierarchy to represent the actions in the domain, and it applies decision trees in combination with Dempster-Shafer theory of evidential reasoning to assess hypothesis regarding a user's plans in the context of a user's actions. In particular, the Dempster-Shafer theory takes into account the reliability of the data obtained so far in order to moderate the probability mass assigned to the hypotheses postulated by means of the decision trees. The Demster-Shafer theory is also applied in [15], where a threshold plausibility and different levels of belief are used to distinguish among competing hypotheses.
The plan recognition mechanism described in [16] works on a graph which represents the relations between the actions and possible goals of the domain. The system iteratively applies pruning rules which remove from the graph goals that are not in any consistent plan.

5 Summary

To sum up, an interface agent designed using the AGUSINA software architecture will be able to *observe* the user using the application the agent was designed for. By doing this, the agent will know in every moment what *task* the user is performing and will try to detect his *intention*, i.e. what task he is more likely to do next. The selection of the user intention will be done taking into account the user preferences and habits, so the agent will intervene and collaborate with the user in a personalized way.
The designer of the agent has to specify two different models: the Task Model, and the Intention Model.
To build the application's task model, we have to follow the next three steps:

1. Make a hierarchical logical decomposition of the tasks of the application represented by a tree-like structure.
2. Identify the temporal relations between tasks at the same level.
3. Recognize the user (or agent) events that allow advancing from a task to the following. This is a level-to-level process.

To build the agent's intention model, we have to:

1. Identify sequences of tasks in a graph-like structure, each of them representing a context of execution of the tasks involved.
2. Associate to each node possible intention tasks.
3. Associate prior and conditional probabilities to each node in the graph.

The extension of the task model concept to be used by an interface agent, and the presentation of the intention model are the main contribution of this work.

References

1. Schneiderman, B. Direct Manipulation: A Step Beyond Programming Languages. In IEEE Computer, Vol. 16, No. 8, (1983) 57-69
2. Kay, A. User Interface: A Personal View. In: The Art of Human-Computer Interface Design. B. Laurel (ed), Addison-Wesley (1990)

3. Maes, Pattie. Agents that reduce work and information overload. In Communications of the ACM, Vol. 37, No 7 (1994) 30-40
4. Rich, Charles; Sidner, Candace; and Lesh Neal. COLLAGEN: Applying Collaborative Discourse Theory to Human-Computer Interaction. In AI Magazine, Vol. 22, No 4 (2001) 15-25
5. Franklin, David; Budzik, Jay; and Hammond, Kristian. Plan-based Interfaces: Keeping Track of User Tasks and Acting to Cooperate. In Proceedings of the 7th International Conference on Intelligent User Interfaces (2002) 79-86
6. Rich, Charles; and Sidner, Candace. COLLAGEN: A Collaboration Manager for Software Interface Agents. In User Modelling and User-Adapted interaction, Vol. 8, Issue 3/4 (1998) 315-350
7. Eisenstein Jacob; and Rich Charles. Agents and GUIs from Task Models. In Proceedings of the Seventh International Conference on Intelligent User Interfaces (2002) 47-54
8. Paterno, F.; Mancini, C.; and Menicori S. ConcurTaskTrees: A diagrammatic notation for specifying task models. In Proceedings of Human-Computer Interaction (INTERACT'97), Chapman and Hall (1997) 362-369
9. Birgit Bomsdorf; and Gerd Szwillus .Coherent Modelling & Prototyping to Support Task-Based User Interface Design. In CHI'98 Workshop, Los Angeles, California, April 18-23 (1998)
10. Lochbaum K. E. Using collaborative plans to model the intentional structure of discourse. Technical Report TR-25-94, Harvard Univ., Ctr. For Res. In Computing Tech (1994). PhD thesis.
11. Lesh, N.; Rich, C.; Sidner C. Using Plan Recognition in Human-Computer Collaboration. In Proceedings of the Seventh International Conference on User Modeling, Banff Canada. Springer-Verlag (1999) 23-32
12. Pearl, J. Probabilistic Reasoning in Intelligent Systems. Morgan Kaufmann Publishers, San Mateo, California.
13. Brown, S. M., A Decision Theoretic Approach for Interface Agent Development, PhD Dissertation, Department of Electrical and Computer Engineering, Air Force Institute of Technology, 1998.
14. Bauer, M. Acquisition of user preferences for plan recognition. In Proceedings of the Fifth Internationa Conference on User Modelling (1996) 105-112
15. Carberry, S. Incorporating default inferences into plan recognition. In Proceedings of the eight National Conference on Artificial Intelligence, Boston, Massachusetts (1990) 471-478
16. Lesh, N.; Etzioni, O. A Sound and Fast Goal Recognizer. In Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, Montreal, Canada. (1995) 1704-1710