Connecting web applications with interface agents

Analía Amandi and Marcelo Armentano

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

Abstract: Interface agents are one of the most relevant applications of agent technology to assist humans in using computer software. However, the development of agents assisting users working on the web, using multiple applications to browse information, represents a challenging task. Particularly, when these agents need to act on standard web applications whose source code is not available, developers have one additional requirement: the complete independence between the web application and the agent. This paper proposes a method for connecting these two independent components based on a software architecture that specifies parts and relations of this particular type of web agents.

Keywords: interface agents; mixed-initiative systems.

Reference to this paper should be made as follows: Amandi, A. and Armentano, M. (2004) ‘Connecting web applications with interface agents’, *Int. J. Web Engineering and Technology*, Vol. 1, No. 4, pp.454–470.

Biographical notes: Analía Amandi received a PhD Degree in Computer Science in the Universidade Federal do Rio Grande do Sul, Porto Alegre, Brazil in 1997 and the Computer Science degree at the UNLP University, La Plata, Argentina in 1990. Currently she is a Professor in the Computer Science Department and head of the agent group of the ISISTAN Research Institute of the UNICEN University at Tandil, Argentina. She has over 30 papers published in conferences and journals about agents. Her research interests includes intelligent agents and software architecture.

Marcelo Armentano is a PhD candidate. He obtained the Systems Engineer degree at the UNICEN University, Tandil, Argentina in 2003. Currently he is an Assistant Professor at Computer Science Department of the UNICEN University at Tandil, Argentina. His research interests includes intelligent agents and software architecture.

1 Introduction

Users working on the web have to deal with a variety of applications for reading information, buying from different shops, interacting with other users, among others. All these facilities involve users handling a large amount of data using different applications.

Copyright © 2004 Inderscience Enterprises Ltd.

Advances in user-interface design as well as in help systems have helped to reduce the overload problems. However, these overload problems are still present since the amount of data and applications are growing continuously. Users need to analyse texts, products and so on for detecting which of them are of their interest, and spend too much time on multiple applications with different user interfaces.

Agents, particularly interface agents [1], represent an answer for that problem. These agents interact with human users bringing a personalised assistance. For achieving this goal, agents work like a human secretary, learning about interests and habits of their boss and then acting according to such information.

Interface agents have the ability to receive orders from human users and answer in a personalised 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. These interface agents that execute orders and show initiative for helping human users define a kind of web agents that represent a solution for users working on the web.

To develop this kind of web agents is not a simple task, particularly if we want agents that can help users working on any standard application on the web. One particular requirement in relation to their life on the web has been detected in these agents. The range of web applications that a user could potentially use for a given goal (i.e. e-commerce) is unpredictable. Moreover, we do not have the possibility of changing these applications. Therefore, we need to develop these web agents completely independent of the web applications. This requirement represents a significant constraint on the design interface agents acting on the web when these agents are not directly built on specific applications.

To make this constraint clear, we could suppose that we want to build an agent for assisting users buying shoes using a group of different web applications. In order to achieve this goal, agents need to observe our users using those web applications, to detect each gesture performed on the user interface of each application and to process them for both improving the user profile and detecting user intentions and intervention situations. Once the agents have detected intentions and intervention situations, they can answer to the user's 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 one of those gestures and in which situation the agents could intervene. All of this information depends on the user interface of each application. Therefore we need to build interface agents completely independent of web applications.

This requirement affects the design of the functionality of the agent in relation to detecting when a user could be helped. Therefore, we analysed several agent designs and extracted two points in which a connection between the application and the agent are materialised. These are the observations of each gesture of the user on the application and the definition of situations in which the agent can act.

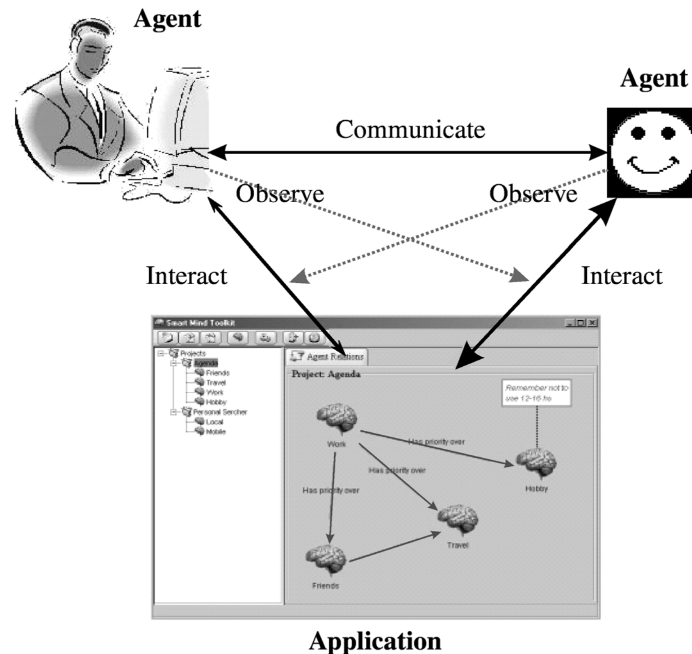
We developed a method named WAD (Web Agent Developer) supported by a particular software architecture named AGUSINA that defines explicit components for our goal. Following this method, a developer can connect any web application with an interface agent assisting any kind of user.

This paper introduces our method for developing interface agents on the web, detailing how to connect any web application with interface agents that assist users working on those applications. The presentation is structured as follows. Section 2 discusses the implications of building interface agents for assisting users working on standard web application. Section 3 presents the solution of the observation of the web user using web applications for detecting user intentions and for improving the user profile. Section 4 introduces the solution to trigger agent actions. Section 5 presents a discussion about related works. Finally, we present our conclusions.

2 Interface agents on the web

One approach to add a Collaborative Interface Agent [2] to a conventional direct manipulation graphical user interface is shown in Figure 1. This approach intends to imitate the relationships between humans who are working together on some specific task involving a shared artefact, as two painters painting a wall together, or two architects working on a plan together. In this approach, an interface agent replaces the second human. Both the user and the agent can interact with the application (the shared artefact), communicate each other, and observe each other's actions.

Figure 1 Collaborative interface agent



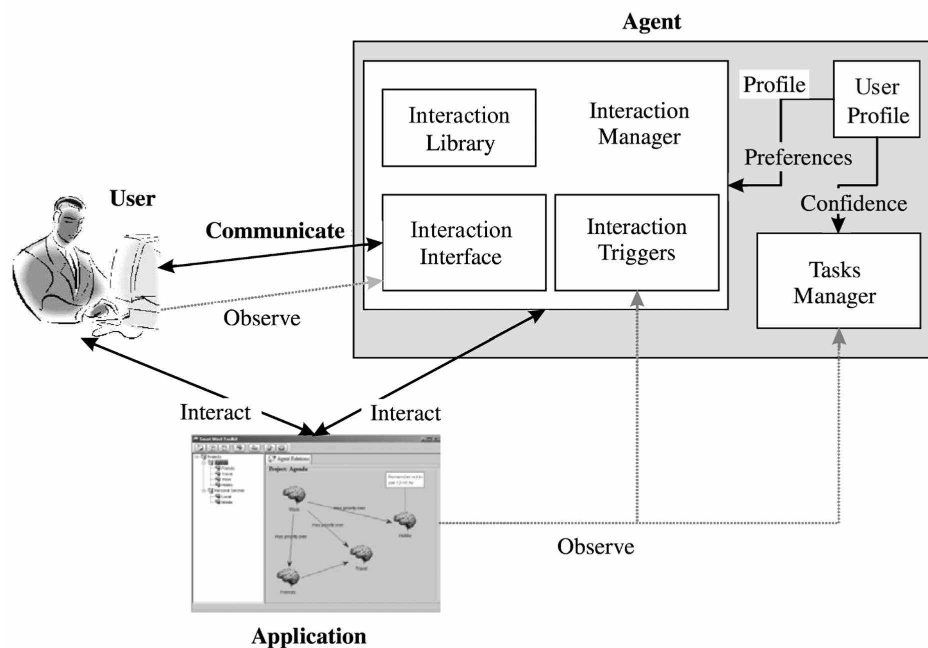
Our primary goal is to join existing web applications with the benefits of agency without much programming effort, and making minimum changes to the application code. Interface agents currently on the web like Letizia [3], NewsAgent [4], PersonalSearcher [5], QueryGuesser [6], Syskill and Webert [7] and WebWatcher [8], have been built together with the base application and, consequently, often agent applications were

contaminated with agent code. We do not want to judge the decision, but only want to point out that we do not have any opportunity of taking this design option when the application code is not available. Then, if we want to develop an agent that acts on standard web applications, we have the requirement of being completely independent of the base application.

Here we present a proposal to cope with these issues. Our method WAD specifies the steps for materialising the connection based on a software architecture named AGUSINA (AGent USer INteraction Architecture).

Figure 2 presents an overview of our software architecture AGUSINA, showing the main components of the agent and their relationships. As can be noted there is clear separation between the agent and the application while the relationships shown in Figure 1 still hold.

Figure 2 AGUSINA overview



The user interacts with the application as usual. The agent interacts with the application through its API. The agent reports its actions and communicates them to the user through the Interaction Interface component.

The architectural components that guide our method WAD are the following: Task Manager Component and Interaction Manager Component. The Task Manager Component specifies the alternative gestures of the user on the user interface of the application. The Interaction Manager Component specifies the triggers of the application for intervention of the agent.

Therefore, our WAD method has two main steps for making the connection between the web application and the agent. These two steps are the following:

WAD method:

Step 1: Detection of user gestures
Tool: task model
Goal: user intentions, user profile

Step 2: Detection of intervention situations
Tool: interaction schemes
Goal: answers, agent intervention

Firstly, the agent observes the user using the application to detect his intentions. It makes use of the application's task model to detect how each user gesture contributes to the current task. The user interaction with the application may also serve to enrich the user profile, detecting patterns in the tasks performed. The task model is an important source of knowledge of the agent, because it contains information about all possible tasks that the user can perform.

Finally, the agent should detect specific situations in which it can interact with the user. Communications with the user are given in two ways. Not only can the user request for the agent's help, or ask it a question, but also the agent can determine that it can initiate the communication act or perform a task on the user's behalf. To do so, it makes use of different kinds of interaction schemes representing intervention situations, along with schemes of the messages presented to the user. The interaction of the user with its personal agent is given in a separate window not to corrupt the application code with agent code.

Notice that there is no need of modifying the application itself to integrate it to the agent. The 'glue' between the application and the agent is given by the task model, whereas interaction schemes and its presentation in a user-agent communication window separate from the application prevents us from modifying it to carry out the interaction.

The following sections detail each step of the WAD method.

3 Detecting user gestures

The ability to reason about user activities is a key point in the development of any intelligent user interface [9]. If the agent is able to recognise what the user is doing, it can act to cooperate. Looking at the tasks that the user is performing, and reasoning about the sequence of actions, the agent can provide an interesting level of interaction. In addition, the understanding of the user's activities provides a context to plan future actions and to 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.

There are three ways that people usually use plans (in the non-technical sense of the word) [9].

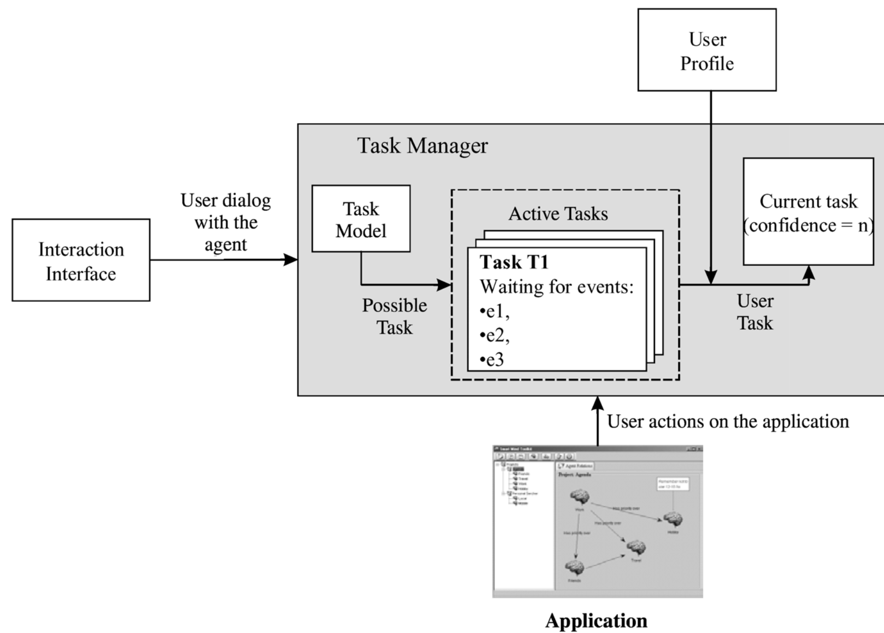
- Plan execution: to accomplish some goal, we may execute a plan that will achieve it.
- Plan recognition: to see what goal someone else might be pursuing, we may match his actions to the set of plans we know about.
- Projection: to guess what someone is likely to do next, so that we are able to cooperate, we can follow along with him in his plan and look at future actions. For example, if someone puts a product into the shopping cart, we can infer that he is going to buy the product.

These ways of reasoning about plans represent an essential part of how we interact with the world, and therefore any collaborative agent should support them.

Any interface agent implemented under AGUSINA is able to recognise the user's plan (i.e. the task he is trying to perform), perform that plan, and inspect it to collaborate with the user, if appropriate.

The task manager is the component of the architecture whose main function is to recognise the user intention while interacting with the application. To do so, this component keeps information about all the possible application tasks, recording them in the task model, and may be able to identify which of these tasks the user is performing in any given moment. In Figure 3 we expand the part of Figure 2 related to the Task Manager.

Figure 3 Task manager



Based on the use of the application through its user interface, agents must recognise which activity the user wants to perform for finding a way to collaborate. Often, a single action on the user interface is not enough to detect the task the user is carrying out.

For specifying the possible gestures of a user on a user interface of an application, WAD uses a task model. Each gesture is defined as a task, for instance, a click on a given icon. The next section presents details about this design tool.

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.

At first, all the tasks are active, because any given event was triggered (since the user did not execute any action). The task model 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 five 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 who may perform the current goal or a step in the current task.
- Identifies a parameter of current task.

If none of the last situations is given, the agent concludes that the current action initiates an interruption [10], 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 recipe 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 considers appropriate).

3.1 Specifying each gesture of the user

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 uses them is to formalise those intentions in an explicit task model. An explicit task model can be used to control the behaviour of a software agent that helps a user perform tasks using a GUI [11].

There are many different task model representations. In this work, we use the following variation of ConcurTaskTrees notation [12].

A task is defined by the following attributes:

- *Name:* used to represent the task.
- *Type:* defines if the task is concrete or abstract.
- *Subtask of:* name of the subtask that contains this one.
- *Steps:* a vector of tasks that, performed together, under given restrictions, achieve the task.
- *Restrictions:* a vector of restrictions in the execution order between the subtasks.
- *Iterative:* a boolean representing the iterative nature of the task.
- *First action:* a set of possible initial events of the task.
- *Last action:* a set of possible final events of the task.

To build a task model, we have to follow the three steps given below:

Step 1.1: Make a hierarchical logical decomposition of the tasks represented by a tree-like structure.

Step 1.2: Identify the temporal relations between tasks at the same level.

Step 1.3: Recognise the user (or agent) events that allow advancing from a task to the following. This is a level-to-level process.

The operators that we use to describe the temporal relationships are:

- T1 ||| T2, interleaving: the actions of the two tasks can be performed in any order.
- T1 |[]| T2, synchronisation: the two tasks have to synchronise on some actions in order to exchange information.
- T1 >> T2, enabling: when the first task is terminated then the second task is activated.
- T1 []>> T2, enabling with information passing, in this case we want to highlight that when T1 task terminates it provides some value for task T2 besides activating it.
- T1 [> T2, deactivation, when one action from the second task occurs the first task is deactivated.
- T1*, iteration, the task is iterative.
- T1(n) finite iteration, how many times the task will be performed is specified.
- [T1], optional task, its performance is not mandatory.
- T, recursion, the possibility to include in the task specification the task itself.

The first problem that may arise, if we simply build task models using these operators, is the possible ambiguity of some expressions. For example in Figure 4, we can interpret the specification in two ways: either (T1 [] T2) ||| T3 or T1 [] (T2 ||| T3). To solve this ambiguity we can introduce an abstract task, which removes the ambiguity of the expression, as shown in Figure 5.

Figure 4 An example of possible ambiguity

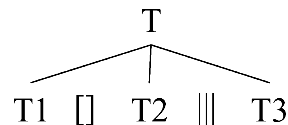
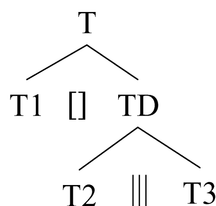
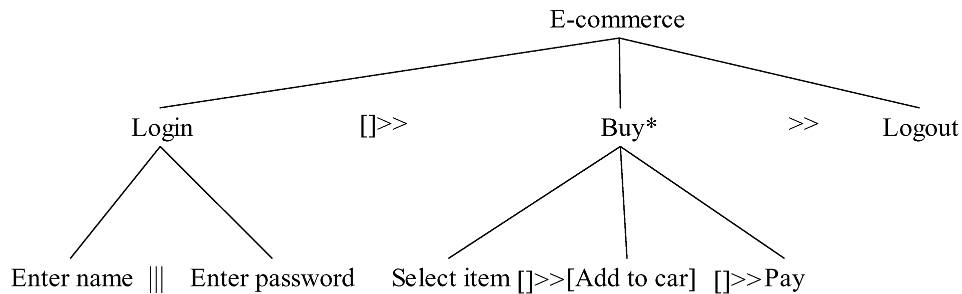


Figure 5 Possible solution to the ambiguity problem



In Figure 6, we expose an electronic commerce example, in which the involved tasks are: login, buy and logout. Temporal relationship between the first and second tasks is enabling with information passing (i.e., when login finishes, it provides a user name and password for buy task besides activating it). Then, the user will select one or more possible items to buy, and put them into the cart; if he does this, the next task will be to pay for the item. Finally, the user will do a logout.

Figure 6 An example of a task model of an electronic commerce application



4 Detecting intervention situations

The interaction between the user and the personal agent determines a mixed-initiative way of human-computer interaction. In this way, not only the user has the possibility of requesting for help from the assistant, but the assistant can also take the initiative for starting the interaction with the user.

The agent intervention in the activities that the user is performing on the application should be carefully designed to ensure that the user feels that he has the control over the application in every moment. It is also important for users to know when the agent can intervene, how the intervention is materialised and, more importantly, how the agent can act on their behalf.

A serious problem that comes up at this point is to specify when the agent believes it has the expediency of starting an interaction with the user and, what kind of interaction it takes up. If those major aspects about agent interventions are clear, it is more likely that the user will feel he has the control of the system every time.

Both specifying the advice the agent can give and the moment it will interrupt the user to show them are two open problems related to the interface with the user. These two problems involve significant difficulties that need to be solved to go forward in the state of the art of interface agents [13].

Therefore, the second contact point between an application and an interface agent is the situations that appear in the application that represent triggers for the agent to intervene. With the goal of defining steps to specify these triggers, we first classify the alternative types of interaction between users and agents and then we define the triggers that lead to this second connection point. Finally we expose these triggers inside the architecture and how they are used in our method.

4.1 Types of interaction

Agents can take the initiative during their interaction with users in several ways. We have analysed some interface agents and we have detected three basic types of interactions: Advice, Action, and Query. Each of these interaction types can be presented in three different ways of abstraction:

- Scheme: Interaction is represented in an abstract way, using variables to symbolise application elements that come into play. For example, buy (*Name*, *Type, Price*, *Store*), where *Name* is a product of type *Type* being sold at price *Price* in the store *Store.*
- Instance: It is a scheme with all of its variables linked to a concrete value. For example, buy ('Lord of the rings, the two towers', '*Book*', 'US$ 25', '*Amazon*').
- Visualisation: It is the way the agent shows the interaction to the user. For example, 'I suggest you buy the book *Lord of the rings, the two towers*. Amazon is selling it at US$ 25'.

We define both schemes and visualisation at design time, but instances are created during the agent runtime.

4.1.1 Advice

In essence, Advice type refers to suggestions of actions and reports of problems or special situations. Therefore we consider three subtypes of advice: Suggestion, Warning, and Remainder.

A suggestion is made up of four attributes:

- A set of actions expressing that the agent is suggesting.
- The trigger of the suggestion, that can be a problem, a special situation, explicit user requests.
- A set of facts expressing the context in with the suggestion is given.
- A set of possible side effects of executing the suggested actions.

For instance, a suggestion for the problem of an item being out of stock could be a change of the product by a similar one, and is defined by 'change(product(P1, T1, Pr1, S1), product(P2, T1, Pr2, S2))'. A possible visualisation for the suggestion is 'I suggest you buying the T1 P2, in S2 at Pr2, because S1 has run out of P1, at Pr1'. The context in which the suggestion is given may be similarProduct(P1, P2) and similarPrice(Pr1, Pr2). A possible side effect is notStock(product(P2, T1, Pr2, S2)), i.e., S2 run out of P2.

When the agent detects a problem, it can suggest an action or just report the problem. For instance, the agent may decide to advise the user of the 'out of stock' problem instead of suggesting anything. We can visualise it, for example, by 'S1 has run out of T1 P1 at Pr1'. Instantiating it we could get 'Amazon has run out of book Harry Potter and the Chamber of Secrets at US$ 10'.

Finally, sometimes the user may not take any action, when facing a problem, or may request the agent some time to respond. In this case, the agent can made use

of remainders, i.e. notifications of an already given problem or special situation. A remainder is made up of three attributes:

- the source of the remainder, which could be a special situation or a problem
- a set of facts expressing the context in which the remainder has no more sense
- a time interval in which the remainder will be given again.

For instance, notLoginYet() has sense until the user performs the login task.

4.1.2 Actions

This type of interaction handles a key concept of mixed-initiative systems. When the agent acts in favour of the user, it can also occur that the user has the feeling of losing control of the application. Therefore, users usually accept this type of control delegation under hard restrictions.

Taking this pattern of user behaviour into account, the action type groups two subtypes of agent interactions. On one hand, Offers to act defines an agent offering to act on behalf of the user; and on the other hand, To Act defines actions that the agent performs without asking the user (only reporting what it has done).

Both kinds of interactions have the following attributes in common:

- the action the agent will execute, or offer to execute
- the trigger of the initiative, which could be a problem, a special situation or an explicit user request.

Further, Offers to act have associated a kind of offer related to its continuity. These are now, by now, and forever.

4.1.3 Queries

The query type refers to interventions in which the agent asks the user about a situation where the agent is not sure of what to do or what to think about.

A query scheme is described by two attributes:

- a sequence of actions expressing the agent's doubt
- a set of possible instantiations of each non-instantiated variable inside the doubt definition. If it is the case of not containing any non-instantiated variable, the second component of a query will be one of the following values: *yes, no, sometimes, I don't know*.

For example, price(product('Hamlet', 'book', X, 'Amazon') expresses the agent does not know the price of the book Hamlet at Amazon. In this case, X could be instantiated with US$ 20. A doubt like usuallyBuy('jeans', 'Amazon'), consulting whether the user buys jeans at Amazon, may have as answer 'No'.

4.2 Interaction triggers

Interaction triggers are facts that stimulate the agent. There are basically two types of triggers: internal and user requests.

Internal triggers are facts in the face of which the agent needs to intercede, initiating an interaction with the user. This kind of trigger has three subtypes: problem, special situation, and doubt.

User-request interaction triggers are requests from him to the agent, and can be suggestion requests, and action requests.

Interaction triggers can be presented in the three levels of abstraction discussed earlier.

In the following section we give details of each of these subtypes of triggers.

4.2.1 Problems

We define a problem as a set of facts representing a conflictive context in presence of which the agent will act.

An example of problem in the e-commerce domain is, as mentioned earlier, the 'out of stock' fact. Its scheme could be count(product(P, T, Pr, S), 0), visualised, for instance, as 'There are 0 P, at Pr in S'. Once variables had been instantiated, we will get something like 'There are no more 'Hamlet' books at US$ 20 in Amazon'.

4.2.2 Special situations

A special situation is defined as a set of facts representing a context in which the agent needs to intervene. Unlike problems, a special situation is not conflictive. Rather, it expresses a situation or user behaviour unexpected by the agent.

As an example, consider the user putting in the shopping cart a very expensive product that he is not used to buying. The scheme of this special situation will be veryExpensive(P, T, Pr, S), visualised, for example, as 'The T P that your are going to buy at S is very expensive. Its cost is Pr, and you usually don't spend more than M'. Once instantiated, we may obtain 'The book Guinness 2003 that you are going to buy at The Book Store is very expensive. Its cost is US$ 80, and you usually don't spend more than US$ 30'.

4.2.3 Doubts

The doubt is the trigger associated to the Query interaction type, with agent initiative.

Understanding user intentions is the main task of a collaborative agent. There are only two options when the agent is uncertain about something: ask the user about his intentions, and infer them from the context.

When the information required by the agent can not be obtained from the context, a doubt will be activated. This trigger will stimulate a query from the agent to the user.

Prompting the user is good in some situations, but over-use of prompts may produce user tiredness. So, the agent may first try to solve the doubt with the available information.

4.2.4 Suggestion request

The user should have the possibility of asking an agent for a suggestion. The agent should consider the context in which the request is given to give an appropriate answer.

For example, the user may ask the agent to suggest the best place to buy a specific book: 'Suggest me a store to buy from'. The context the agent should take into account is the other elements of a buy scheme that the user may already have entered in the application, such as the type of product, name, or desired price. With context information, and user's profile, the agent will give a more effective suggestion.

A suggestion request is made up of two components:

- a set of facts expressing the request, for example bestPlaceToBuy('Hamlet', 'book', Pr, S)
- a set of facts expressing the context in which the user made the request.

4.2.5 Action request

The final user to agent interaction is explicitly soliciting the execution of a task. An action request has the following attributes:

- a task (from the application task model) to be executed
- a set of facts expressing the context in which the agent requests the execution of the task.

This is an important kind of interaction, because the agent can learn which tasks the user usually delegates to it, and then can offer to perform them autonomously, or execute them directly.

4.3 Interaction Manager

Figure 7 shows an expansion of Figure 2 related to the Interaction Manager. In this figure, we can observe the Interaction Manager Component of the AGUSINA architecture. In this component, elements that represent triggers for agent intervention are clearly defined, showing their relationships with other components explicitly.

The steps that we consider necessary for designing this connection with the application follows:

Step 2.1: Specify schemes of problems that represent triggers for agent intervention.

Step 2.2: Specify schemes of special situations that represent triggers for agent intervention.

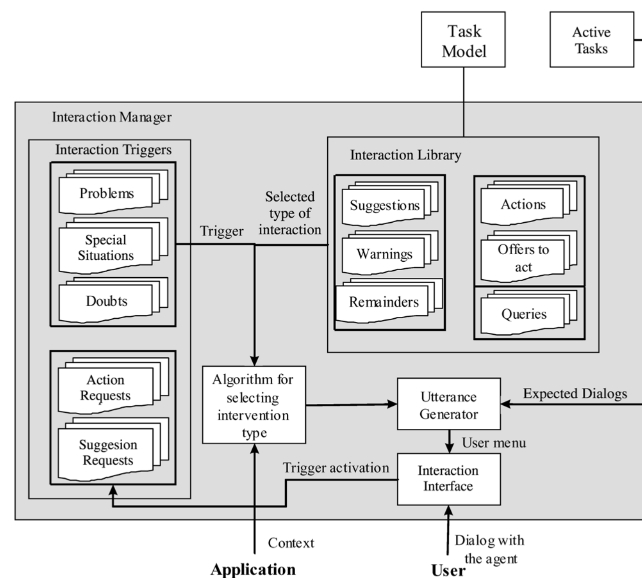
Step 2.3: Specify schemes of doubts that represent triggers for agent intervention.

Step 2.4: Specify schemes of action requests that represent triggers for agent intervention.

Step 2.5: Specify schemes of suggestion requests that represent triggers for agent intervention.

The detection of these triggers produces a situation that a communication from the agent to the user will be processed. The items in the communication user's menu have two additional sources. The first is the set of active tasks: each active task is associated to a set of expected events to go a step forward in its execution. Each of these expected events will have an associated entry in the user's menu, such as 'Perform X', indicating that the user would like the agent execute the task named X, which is a subtask (a step) in the current task.

Figure 7 Interaction manager



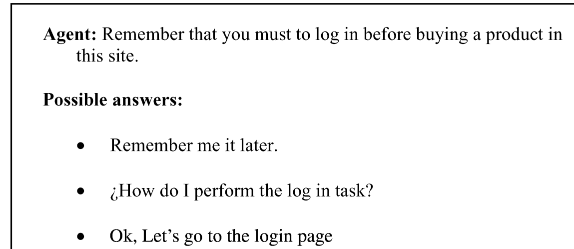
The second source of generated utterances is the interaction library. When the agent decides to take the communication initiative with the user, the algorithm for selecting interaction type chooses one from this library. This selected interaction has an associated visualisation that is the item added to the user's menu. We must also take into account that each interaction has associated a set of possible answers from the user. Therefore, when the user selects an interaction from the menu, all the possible answers associated to it are added to the utterance generator. Notice that now again, if the user selects an answer to the agent dialog, it may have a set of associated answer interventions, and so on.

All the facts placed above indicate the need for having a tool that allows us to design the talks between the user and the agent, i.e., that allows us to specify the responses associated to each intervention.

Finally, the utterance manager includes a set of special entries in the user's menu designed to help the user in the execution of the task he is pursuing, such as 'Where am I?', 'What should I do next', etc. [2].

Figure 8 shows an example of automatically generated answers from an agent remainder.

Figure 8 Automatically generated answers example



5 Related work

There is some work in the current artificial intelligent research area that describes interface agent architectures. Although there are methods to build agents in the wide sense of the word, none of them takes into account key concepts of interface agents and mixed-initiative systems on the web.

The most relevant work with respect to that the work presented here is COLLAGEN (for COLLaborative AGENt) [2]. COLLAGEN is a Java middleware developed at MERL to make it easier to implement collaborative interface agents. Among other things, COLLAGEN provides a generic implementation of discourse interpretation, plan recognition, and plan generation algorithms.

The code in the agent interface is, basically, a bridge between the primitive action types of the task model and the specific elements that achieve them in the given GUI implementation. This code can be quite onerous to write, particularly if the author of this code is not the same as the author of the GUI implementation.

This work intends to define a method based on an architecture that is applicable to any kind of collaborative interface agent. One of the first things that our work provides is a clear separation between the application and the agent, that is similar to that described in COLLAGEN, but in contrast, we allow the definition of agent-related features that can be reused in different interface agents. Further, our work provides a complete support to different interaction kinds and triggers defined. Instead, COLLAGEN provides only a generic framework to record decisions and communications that the agent made, but not to build them. Although it is possible to use COLLAGEN to obtain an agent automatically, the developer still has to hand code the agent interface, which allows the agent to observe the user's interactions with the application and to interact with the application itself. These issues can be an important barrier to using COLLAGEN.

Another approach is presented in [14], where the authors define a generic software architecture for modelling interface agents based on the BDI approach [15]. Defining the behaviour of an interface agent using a BDI model eases the design and maintenance of interface agents. This architecture is supported with a methodology that guides the designer to build a complex interface agent.

Other architectures focus on the benefits of adding interface agents to web applications. In [16], the authors present an architecture to build autonomous agents for the internet, using case-based reasoning (CBR) systems. The proposed methods facilitates the automation of their construction and provides them with the capacity of learning and therefore of autonomy.

6 Conclusions

In this paper we have introduced a method that enables web applications to be connected with interface agents without any changes in the source code of the base application. This method is based on a software architecture for interface agents and is the product of our experience in developing agents on the web.

The AGUSINA software architecture defines the main abstract component of an interface agent and, based on it, our method specifies the steps and the tools for materialising those components under the relevant restriction of independence between application and agent.

References

1 Maes, P. (1994) 'Agents that reduce work and information overload', *Communications of the ACM*, Vol. 37, No. 7, pp.30–40.

2 Rich, C., Sidner, C. and Lesh, N. (2001) 'COLLAGEN: applying collaborative discourse theory to human-computer interaction', *AI Magazine*, Vol. 22, No. 4, pp.15–25.

3 Lieberman, H. (1995) 'Letizia: an agent that assists web browsing', *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pp.924–929.

4 Cordero, D., Roldan, P., Schiafino, S. and Amandi, A. (1999) 'Intelligent agent generating personal newspapers', *Proceedings of the International Conference on Enterprise Information Systems, ICEIS'99*, Setubal, Portugal.

5 Godoy, D. and Amandi, A. (2000) 'PersonalSearcher: an intelligent agent for searching web pages', *Lecture Notes in Artificial Intelligence*, Springer, Vol. 1952, pp.43–52.

6 Schiaffino, S. and Amandi, A. (2000) 'The QueryGuesser agent', *Proceeding of the Argentine Symposium on Artificial Intelligence*, pp.29–42.

7 Pazzani, M., Muramatsu, J., Billsus, D., (1996) 'Syskill & Webert: identifying interesting web sites', *Proceedings of the Thirteen National Conference on Artificial Intelligence*.

8 Armstrong, R., Freitag, D., Joachims, T. and Mitchell, T. (1995) 'WebWatcher: a learning apprentice for the World Wide Web', *Workshop on Information Gathering for Heterogeneous Distributed Environments, AAAI Spring Symposium Series*, pp.6–12.

9 Franklin, D., Budzik, J. and Hammond, K. (2002) 'Plan-based interfaces: keeping track of user tasks and acting to cooperate', *Proceedings of the ACM 7th International Conference on Intelligent User Interfaces*, pp.79–86.

10 Rich, C. and Sidner, C. (1998) 'COLLAGEN: a collaboration manager for software interface agents', *User Modelling and User-Adapted Interaction*, Vol. 8, No. 3–4, pp.315–350.

11 Eisenstein, J. and Rich, C. (2002) 'Agents and GUIs from task models', *Proceedings of the ACM 7th International Conference on Intelligent User Interfaces*, pp.47–54.

12 Paterno, F., Mancini, C. and Menicori, S. (1997) 'ConcurTaskTrees: a diagrammatic notation for specifying task models'. *Proceedings of Human-Computer Interaction (INTERACT'97)*, Chapman and Hall, pp.362–369.

13 Lieberman, H. (2001) 'Interfaces that give and take advice', in Carroll, J. (Ed.): *Human-Computer Interaction for the New Millennium*, ACM Press/Addison-Wesley, pp.475–485.

14 Gómez-Sanz, J.J., Pavón, J. and Garito, F.J. (2000) 'Intelligent interface agents behaviour modelling', *Proceedings of MICAI 2000, Advances in Artificial Intelligence, Mexican International Conference on Artificial Intelligence*, Acapulco, Mexico, Lecture Notes in Computer Science 1973, Springer, pp.598–609.

15 Georgeff, M., Pell, B., Pollack, P.E., Tambe, M. and Wooldridge, M. (1998) 'The belief-desire-intention model of agency', *Intelligent Agents*, Springer Publishers.

16 Corchado, J.M., Laza, R., Borrajo, J.C., Yañez, J.C., Luis, A. and Glez-Bedia, M. (2003) 'Agent-based web engineering', *Proceedings of the Third International Conference on Web Engineering*, Oviedo, Asturias, Spain, pp.7–25.