Proactive Assistance to Users of Electronic Brokerage Systems: The Design of the GAIA Interaction Agent

Authors: Koutsabasis P., Darzentas J. S., Spyrou T., Darzentas J.,

Research Laboratory of Samos, Department of Mathematics,

University of the Aegean, GR 83200 Karlovassi, Samos, Greece,

email: {kgp, jennyd, tsp, idarz}@aegean.gr

Abstract

In this paper, interaction agents are proposed as a metaphor of the agent/personal assistant for computer-based environments that are innovative and rich in functionality, thus have a tendency to produce information overload and are possibly difficult to use. Brokerage systems are typically such complex environments. The manner by which interaction agents can provide useful assistance to users is presented both generically and in the context of the GAIA brokerage environment. The requirements for the interaction agent architectural components are explained and the agent's conceptual architecture is described. The modelling of the agent environment and interactions, and the selection of inference mechanisms is discussed, as well as the technical approach taken. While the requirements, design and development approach for interaction agents is generic, it is examined in the context of the ACTS (AC 221) project GAIA (Generic Architecture for Information Availability), where this work has been curried out.

1. Introduction

Currently, information brokerage systems are capable of searching enormous quantities of data, often from heterogeneous information sources. In this they can be extremely efficient, producing wide-ranging result sets, drawn from repositories sited globally. In addition, they provide links into supplier sites, and a whole host of value-added information. However, electronic brokers need to provide some means of personalising customer services to be effective at channelling information from suppliers to customers. By analogy, human brokers, such as travel agents, or librarians, before commencing a search, spend time in discussion with their clients to better ascertain their needs. After a broker collects information for a client, he spends time discussing the significance of the information with the client, weighing it up and assessing whether or not it is the information required by the client.

This mediation phase is non-existent or very rudimentary in most information brokerage systems. They rely on the input to the system being a fairly precise expression of the customer's requirements. This gap between information availability and the relevance of information to the customer often results in "information overload" in various guises. This can be seen, for instance, in the sheer overwhelming volume of returned results and in information being returned that is not relevant. The result is frustration and exasperation, and loss of precious time. At the same time, at the other end of the supply chain, information providers know that the quantity of information they provide is increasing and that unless their customers are provided with useful and usable ways of accessing that information, customers will go elsewhere.

Further complexity is introduced by the innovative nature of the brokerage systems user interfaces, as they integrate a rich variety of services, from searching and retrieving information to on-line ordering and delivering of digital goods, incorporating various other facilities (e.g. tariffing, payment). Nor is complexity restricted to these types of services. As bandwidth expands, and new networking technologies

progress, the information that is transmitted is increasingly multimedia based. In addition, the possibility to link up devices such as cellular phones further increases the functionality and complexity of the interaction space.

Information system developers attempt to tackle the problems of both user and information providers, and combat the negative effects of information overload and user interface comprehension and usage with a number of software solutions. Many of these solutions (among others search engines, softbots, recommendation systems) are sited under the umbrella of 'software agents'. Despite the fact that these are sometimes piecemeal solutions which offer limited help, the design and architectural issues that are related to those systems are often vague, even to other system designers.

In this paper interaction agents [25] are proposed as a metaphor of the agent/personal assistant, to support and enhance the interaction between the user and the brokerage system. The interaction agent functions as both an assistant of the computer-based system and (following the metaphor of the human assistant) of the application domain. In order to provide such assistance the interaction agent architecture encompasses a model of the (human and software) environment, and a model of the interaction. The organisation of those models into software capable of exhibiting behaviour is determined by the conceptual architecture of the interaction agent The specific implementation of those models, and the inference mechanisms used, make up the functional architecture of the interaction agent. This paper presents the view of those models at a conceptual and functional level. With regard to existing classifications of agent systems, interaction agents employ characteristics that are mostly met in personal assistants [31], believable agents [7] and pedagogical agents [23,41,28]

This work is being carried out in the context of a European Union ACTS (AC221) project GAIA. **[1, 2]** The GAIA project is developing a sector and supplier independent Generic Architecture for Information Availability to support multilateral information trading. GAIA offers a robust platform for electronic commerce based on the concept of brokers. A broker integrates a variety of electronic commerce functions and components, namely: authentication, directory services, search, order, item and stream delivery, payment, tariffing, alerting **[1, 2]**. The GAIA system is distributed, based on CORBA **[19]**, and is being demonstrated in three application domains: music, technical components data and publishing.

The paper is structured as follows. Section 2 briefly presents the related work from the fields of Human – Computer Interaction, Artificial Intelligence and Software Agents and highlights the open research issues and challenges of this research. Section 3 presents the interaction agent conceptual architecture highlighting the important aspects of this design in an abstract manner. Section 4 discusses in a detailed manner the approaches for identifying the basic internal models, which enable the interaction agent to represent and utilise knowledge about the domain of application and the (human and software) environment, towards the aim of interacting with the user and offering useful assistance. Section 5 presents the issues related to the integration of an interaction agent to an existing application which need to be taken into account towards the design of an interaction agents development environment. Section 6 presents the conclusions.

2. Related Work

Maes [30] defines an agent as "a system that tries to fulfil a set of goals in a complex dynamic environment". In the case of the interaction agent the 'set of goals' that need to be achieved can be summarised as the provision of useful assistance to the user. Thus it is very important to specify the context of agent assistance and the rationale by which these contexts have been identified. The 'environment' of the interaction agent consists of the application user interface and the underlying system (in this case, the GAIA user interface, and underlying GAIA brokerage system). The user actions upon the application user interface provide evidence about the tasks that users try to perform. Knowledge of the underlying system functionality is central, in order to know how to invoke specific modules, for example, in the case of autonomous operation.

The interaction agent must be capable of observing its environmental changes and adapting to those changes by either learning the characteristics of its environmental entities, or choosing to act in order to influence its environment when it 'judges' appropriate. The capability of being adaptive is very important in this sense. Benyon and Murray [3] claim that all adaptive systems must possess a user model, a domain

model (a model of the system), and an interaction model (a model of the user-system interaction). The definitions of those models as well as of the manner in which those models are utilised i.e. the interaction agent architecture. are considered as fundamental design tasks in the modelling of the interaction agent.

As is to be expected, methods and techniques from the field of Human Computer Interaction (HCI) are central to this research. Task analysis and user modelling are of particular interest. Task analysis comprises of a set of techniques that can reveal the knowledge that is required for an agent (human or artificial) to perform a task; task analysis is performed, methodologically or informally, in any type of user interface design. In this research it is used to identify the user–system interaction knowledge. User modelling comprises of a set of techniques that identify and correlate user characteristics. Characteristics can include abilities, beliefs, mental states, human factors, domain preferences, etc. User models can be 'hand-coded': the designer constructs a static user model, or 'machine-coded': the user model is built by the system following a specific rationale and thus dynamically updated. The decision of selecting a static or dynamic user model is domain dependent, as both methods have advantages and disadvantages [4].

For interaction agents to be adaptive, capable of learning, and proactive, Artificial Intelligence (AI) methods need to be applied. Theories and techniques of particular interest are cognitive architectures, plan recognition, reasoning under uncertainty, and machine learning. A variety of intelligent systems cognitive architectures [30] have been applied in various application domains. It is claimed [30] that "an agent does not have general or task-independent functional modules". As code reusability is a key goal in all cases of software implementation, this claim raises the question of whether it is possible to build an interaction agent development environment in which the designer will be enabled to insert the domain dependent knowledge easily. Plan recognition and machine learning techniques have been widely deployed rather interchangeably to model the internal knowledge of an intelligent software agent. The emphasis in interaction agent research is on the adaptability features of those techniques. Thus reasoning mechanisms that can model uncertainty [38] are very important to this research.

With regard to existing classifications of agent systems, interaction agents employ characteristics that are mostly met in personal assistants [31], believable agents [7] and pedagogical agents [23,41,28]. Personal assistants "watch over the shoulder" [31] user actions, deducting about user states and goals, and most commonly employ information filtering algorithms [37] to deduct about user preferences. Pedagogical agents [29] have been applied successfully in virtual learning environments with the principal task of assisting their users to learn to manipulate a virtually represented real-life environment (for example a surgery room or a ship engine room).

In 'traditional AI', a key feature of architectural models is that "the agent has a complete internal model of the environment", which is not the case for software agent design [30]. The interaction agent environment is both, ill defined - as there are no general models for representing the characteristics of humans (at least) -, and dynamic - as various aspects of user behaviour may change over time (cognitive factors, goals, preferences, etc.). The emphasis in this research is towards the representation of the dynamics of the interaction agent models rather than the complete enumeration of attributes. Another reason for this approach is related to software engineering aspects: highlighting only the 'important' (as these are provided by user requirements) attributes, increases significantly software reusability. DeWitt [9] also notes that modelling every possible naturalistic property in the user's world does not lead to the most accurate model.

3. Interaction Agent Architecture

This section overviews the interaction agent conceptual architecture. The architectural components are presented in more detail in the following section. A more abstract view of the interaction agent architecture was presented in [25].

The interaction agent architecture represents the relations among the three basic models as well as the representation of the agent user interface components. The agent environment in the GAIA case consists of the application user interface, as well as the underlying brokerage system. The interaction agent architecture needs to enable the agent to observe, represent knowledge about and affect its environment.

The basic models into which the architecture is based upon (reasoner) are the related to the representation of knowledge about the interaction, the user and the selection of useful assistance. Those three components are totally internal to the agent. Additionally there are components (agent interface) that need to be integrated to the software environment in order to enable the agent to both perceive and affect it. Emphasis is given on investigating the mechanisms by which the application user interface can be utilised and integrated to the agent functionality, for two reasons. Firstly, the agent continuously attempts to adapt to user characteristics, which are observed basically on the application user interface. Secondly,. the underlying brokerage system is determined by a static application program interface (API), which is used only in cases of autonomous agent operations, which are only a few in the GAIA case.

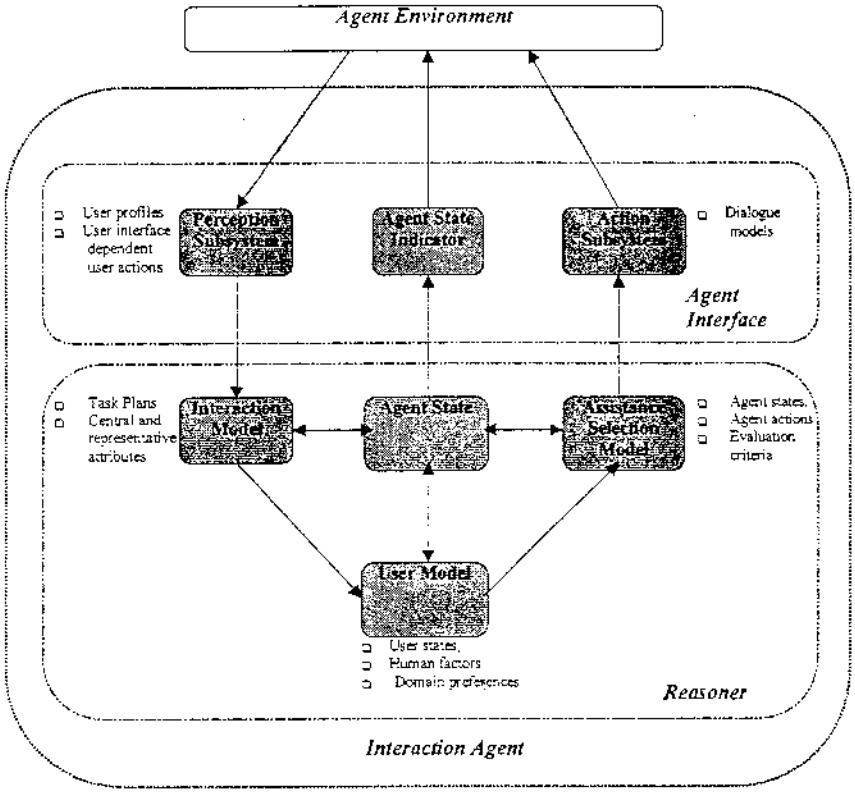

Figure 1: Interaction Agent Conceptual Architecture

Action selection and learning from experience are two important aspects of agent architectures [30]. In the interaction agent architecture, these issues are handled by the three basic models internally. The issue of action selection arises because there is a need to ensure that the agent understands the user tasks (interaction model) and user preferences (user model) while observing user actions. Also that the agent is capable of initiating the dialogue and offering useful assistance. The issue of learning from experience arises because there is a need for the agent to adapt to alternative user plans towards performing a task. Also to be capable of learning user characteristics that affect his behaviour, as these are defined in the user model. Finally to be able to evaluate and possibly correct assistance selection. The fact that each software component that implements the internal interaction agent models handles these issues internally, means that the interaction agent architecture can be implemented as both a single agent or a multi-agent architecture.

With regard to action selection, emphasis is given on the collaborative behaviour of the agent. Firstly, autonomous agent operations may have a financial cost for the user, as the GAIA brokerage system supports tariffing of user actions (tariffing is not enabled in all applications of electronic commerce, however). Secondly a collaborative agent behaviour can provide safest (explicit - provided by the user) means for the evaluation and self-correction of the offered assistance, which is an important aspect of action selection.

Learning from experience is an essential characteristic of any adaptive and intelligent system. The interaction agent architectural components are designed in a manner that supports learning and adapting. However there are also domain and task dependent characteristics of those models that are static. This is the curse of every intelligent system: the representative model needs to incorporate some form of task and / or domain knowledge. These task and domain characteristics are identified and discussed in this paper. The issue of providing the designer with means of easily specifying these task and domain dependent characteristics as well as applying them to the implementation of the interaction agent is discussed as future work at the end of the paper. The interaction agent has been partially developed in Java, while the communication with the underlying brokerage system was implemented in CORBA.

4. Modelling Interaction Agents

4.1. Interaction Model

As noted above the issue of identifying, modelling and utilising the interaction knowledge is central to this research. Interaction knowledge cannot be seen isolated from the tasks that need to be performed from the interacting entities. Task analysis techniques [21, 22] can identify the knowledge an entity (human or artificial) will require in order to achieve its goals, the stages of interaction at which a user would require interaction agent assistance, and alternative plans that this entity would have to execute in order to perform a (sub) task. Among a variety of known and applied techniques [6,11], the Task Knowledge Structures (TKS) technique [21,22] was selected to identify the knowledge for the performance of user and agent tasks. TKS identifies the knowledge about various components for a given task: mental states, roles, goals, sub-goals, sub-tasks, procedures, strategies, actions and objects. It can assist the analyst to capture interaction requirements by identifying the stages of interaction and the knowledge required in each stage of it, in order to perform a task.

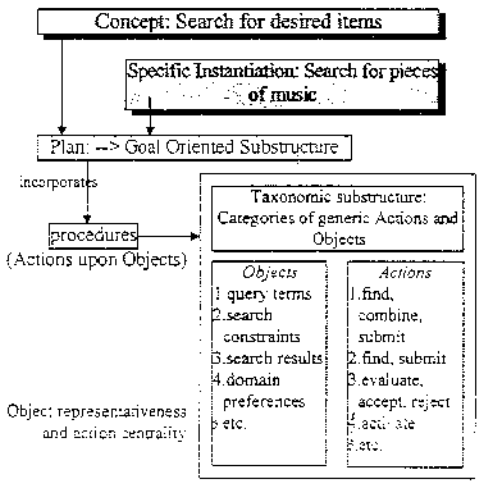

Figure 2: "Search for Desired Items" Task Knowledge Structure.

Figure 1 provides an approximation to describe a user's TKS when the task of 'searching for desired items' is carried out. The actions the user needs to perform in this TKS can be separated into *actions that may put heavy cognitive load on users* (find, evaluate, activate), and *actions that are user interface related:* actual instructions to the system that may also cause difficulties to users unfamiliar with the interface. The interaction agent can assist the user in both contexts. In the case of actions that require domain knowledge, the interaction agent, as an entity knowledgeable about the domain, can suggest to the user information items that may be of interest or query terms to issue a search request. In the case of user actions that are performed on the user interface, the agent can guide the user in manipulating the application user interface. TKS results in sets of plan-goal oriented substructures (plans) that the users would need to follow in order to perform the task. An example is shown in figure 2.

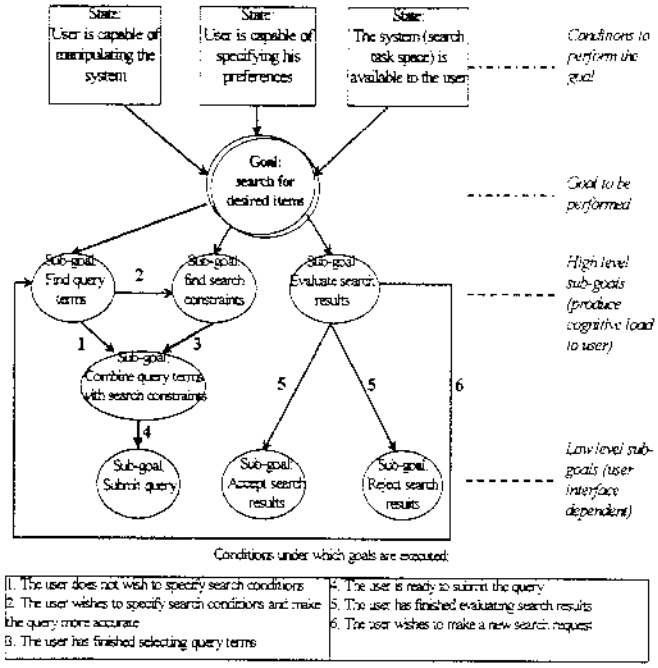

Figure 3: A "Search for Desired Items" Plan.

The production of such plans is very important, as this is a basic feature of the interaction agent architecture. Each (sub) goal in the plan can be evaluated as *central* (obligatory for all users) or *representative* (experienced users might not perform it) for the performance of the task. The interaction agent can evaluate user behaviour (as this is interpreted by user actions upon the application user interface) against the set of plans provided by TKS, in a manner that will enable the agent to reason about user behaviour and adjust its behaviour accordingly. With the use of a plan recognition model the interaction agent is capable to "watch over the shoulder" user actions (typically user interface generated events), in a way that higher level abstractions can be drawn. Also utilisation of plans introduces uncertainty to the interaction agent model of operation. This is true when a user action may correspond to a set of alternative user plans, and possibly sub-goals.

It may be often the case that the user plan is ambiguous to the agent (e.g. a set of alternative plans are candidate to be true). An obvious agent approach to resolve ambiguity would be to explicitly ask the user what is the higher level goal. This is an agent action that can be taken when the user action initiates an 'very important' system operation (a safety critical operation, in the case of GAIA the payment operation, etc.). Such operations can be determined by the user requirements stage. Furthermore specific issues about controlling the dialogue must be taken into account in this case, i.e. does the agent ask the user about the high level goal at once, or does it follow a bottom-up approach and re-constructs the plan (or possibly generates a new plan). In the case of other types of operations, an uncertainty model that will enable the agent to deduct about the user plan without asking the user may be used, taking also into account cases of past user operations. Belief networks [38] fit well in this type of problem. Additionally the issue of selecting past cases of user actions is very important, as there are cases, which may have to be deleted as obsolete.

Modelling the complete set of task plans is a painful requirements and design process. Additionally it may cause severe problems in the case that the application user interface is re-designed: the designer has to provide (once again) the complete task knowledge (plan) structures to the agent. In this design the approach towards modelling plan knowledge was intentionally targeted towards the *incomplete representation of intermediate plan knowledge*. That means that only the two edges of plan knowledge were completely modelled. These were the (high level) tasks, user and system states, the higher level goals and the lowest

level (user interface dependent) goals. For the selection of intermediate knowledge components, only the ones that are characterised as central or representative are selected.

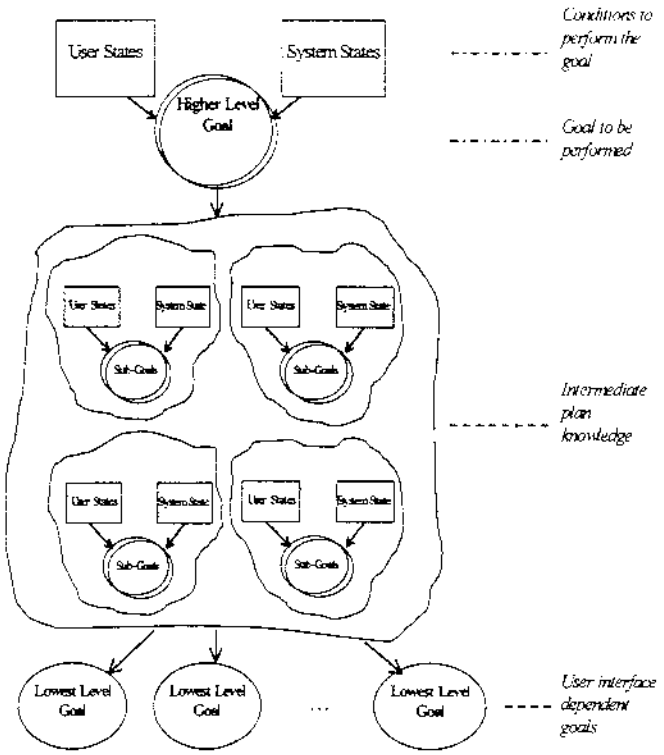

Figure 4: The intermediate plan knowledge is represented incompletely in the interaction model.

This approach presents some very important advantages. Firstly it does not require exhaustive enumeration of plans and attributes, which is a complex and painful task for the designer. This approach also reduces the agent software computational complexity. Only the higher-level user states need to be evaluated (by the user model, as described in the next chapter), and the most important user actions need to be taken into account in this case. In addition this approach contributes significantly towards the minimum changes and the reusability of the agent software in future enhancements that may happen on the application user interface side of the system (central and representative attributes are unlikely to change). The incompleteness of the task model is also valid for the lowest level user actions (application user interface dependent). Actions that are not considered important for the "watching over the shoulder" process are not taken into account. For example, such an action could be a 'Clear' button that is pressed by the user when he wishes to clear the contents of a form that he has filled in the past. The rest of the actions, which can be observed by the agent, are referred as observable user actions in the rest of the paper.

The application of TKS results in a definition of *user and system states* that need to be met in order for the user to execute the required *tasks*, *alternative plans* for the performance of those tasks, and the *identification of user-system interaction stages*. Furthermore it reveals stages of interaction where the actions and objects used, typically place cognitive load on the user in the contexts of *employment of domain knowledge* and actual *manipulation of the system*. The context of the interaction agent assistance is therefore to be useful to the user in both those contexts. In order to model the task knowledge an approach towards the "incomplete representation of intermediate plan knowledge" was taken based on the centrality and representativeness of interaction components. This approach contributes towards reduction of design and operational complexity, as well as reusability and minimum re-design of the agent software. In the next chapter the user model and the manner in which it is utilised in the case of the GAIA interaction agent is discussed.

4.2. User Model

Users have different characteristics that can affect their capability of using the computer-based system. The term characteristics can include various and composite attributes as human factors (tiredness, short-term memory, long-term memory, disabilities, skill, experiences), domain preferences, emotional states, beliefs, expectations, biases, etc. As one might expect firstly there are no commonly acceptable user models for every computer-based system and secondly not all observable human characteristics are essential in order to construct a user model [4].

A big portion of the research in user modelling has been around models that represent characteristics of certain user communities, for example students. Such user models are typically static – once constructed they never change during interaction. However, static user models cannot be used in this case. Brokerage systems are Internet-based and appeal to users worldwide, users that do not belong to a certain community with traceable characteristics. The emphasis in this research is therefore given on the adaptability of the selected user model, than the completeness of the attributes used. For the construction of the user model at least two questions need to be answered by the designer: "what are the human characteristics that are incorporated into the user model?", and "how is the user model knowledge elicited?". The rest of this chapter attempts to answer those questions.

What are the human characteristics that are incorporated into the user model? The types of characteristics that are used in the case of the interaction agent user model can be separated to three categories: user states, human factors and user preferences. User states are initially provided by the task analysis performed. According to TKS, in order for a user to perform a task, both the user and the system need to be in specific states. User states are therefore provided by the designer; they are always provided in a 'positive' manner, for example "the user is capable of performing a search request", and the interaction agent attempts to evaluate to which degree a user state is met by observing user behaviour. In order to evaluate the degree, to which user states are met, human factors and user preferences are used.

There is a variety of human factors that can be taken into account towards the user model [4]. In this design skill, workload and short-term memory have been selected. The first two characteristics are considered dynamic, while the latter is considered static. Short-term memory is determined by the well-known rule proven by practical experiments, that 'the average human can store in the short-term memory 4-7 chunks of information'. Workload is calculated per task space (TS) the number of windows (NW), which need to be opened in order to perform the task and the number of information chunks (IC) on the last (working) window. The information chunks are assigned to each application window by the designer. Thus:

$$Workload = 1(TS) + NW + IC.$$

Workload is evaluated according to the short-term memory rule: it is low when it is below 4, average when it is between 4 and 7 and high otherwise.

The skill of a user is calculated per observable user action. Thus an array of agent characterisations about user actions (with binary values: successful (sa) / unsuccessful (ua)) is kept per each observable action which enables the agent to remember the history of the user while using the system. From this array the following types of skills can be calculated:

$$User\ Skill\ per\ Task = (sa) / (sa+ua),$$

$$User\ Skill\ per\ Workload = Wi*sa/sa+ua,\ for\ all\ tasks\ with\ Workload\ W=i,$$

$$Absolute\ Skill = (W1+W2+...+Wi)*(sa) / (sa+ua).$$

Domain preferences involve user tastes with regard to the domain of application. They can be utilised in a variety of models, which can be separated in two broad categories. The first category is that of using knowledge engineering techniques in order to create higher level abstractions of information items. Any new information item can be grouped under those abstractions. Afterwards information items can be evaluated as relevant to the user in comparison with the types of information that the user prefers (after elicitation of user preferences). The second category is that of information filtering techniques. Information filtering techniques can be separated into two broad categories, content-based and social (or collaborative) information filtering. Content-based filtering is an effective technique even when the number of users is small, but tends to propose information items that have been proposed in the past. Social information

filtering can propose unique information items to users, but is not effective until the users of the system are many [37]. Those user preferences models are dynamic, as they can provide different suggestions, as the user (or other users) provide domain preferences information to the system.

How is the user model knowledge elicited? To elicit the required knowledge for the user model (user states, human factors, domain preferences) two approaches are used: the 'watching over the shoulder' approach and user profiles. As noted above, the designer according to TKS, determines user states and the agent continuously evaluates the degree to which they are satisfied.

'Watching over the shoulder' is an implicit way to acquire knowledge about the user at specific stages of the interaction. In the GAIA case and with regard to user modelling knowledge elicitation, this approach is used to acquire knowledge about human factors and user preferences. The agent watches observable user actions, for updating the human-factors knowledge (skills, workload). The agent watches some of the observable user actions, for updating the user preferences. The actions when this approach is taken for updating user preferences are: when a user puts an item in his shopping basket, when he orders an information item, and when he evaluates an information item highly without ordering it.

User profiles provide a simple way for eliciting user knowledge in an explicit manner. In this design user profiles are used in order to acquire information about user preferences explicitly. User profiles are important in GAIA for other operations of the brokerage system as well (alerting). The information that users provide the system, besides the alerting operation, is used for future information items recommendations.

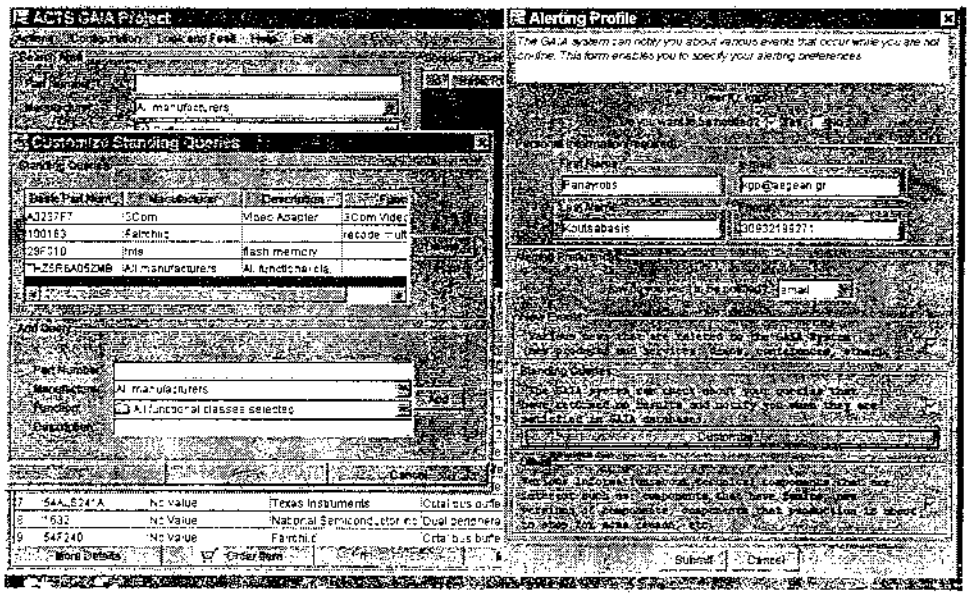

Figure 5: The GAIA alerting profile can acquire knowledge about user preferences explicitly.

The user modelling component evaluates the degree to which the user is in the required set of states for the performance of the action. The user skill as this is evaluated by the user model (utilising other human factors), is used to evaluate the user states, while the user preferences are employed in order to offer domain dependent assistance if needed. These attributes are directly related to the contexts of assistance, which the interaction agent cares to provide. The user state knowledge is essential for the agent in order to select an action (or a set of actions) that correspond(s) to the evaluated this user state, as we will see in the following chapter.

4.3. Assistance Selection Model

The outcome of the previous two models is a set of user states (which are the prerequisites for the performance of the selected task) and the degree to which they are met, as evaluated by the agent. The assistance selection model is based on the assumption that every user state corresponds to an agent state. This mapping is straightforward. However the user state is determined by attributes that are either dynamic (user skill, user preferences), or are determined dynamically during interaction (interaction knowledge, workload). Each agent state corresponds to a set of possible agent actions, which can be evaluated according to their usefulness to the user dynamically. These actions are related to the provision of assistance to the user in the two contexts identified by the TKS. In order to provide useful assistance a number of issues need to be taken into account.

A first issue is related with the degree of agent autonomy. Does the agent autonomously undertake tasks on behalf of the user, or does it follow a collaborative approach? The most common ways to decide the autonomy level of agent assistance are based on threshold values that define the limits of action selection (autonomous versus collaborative) and the types of assistance [30]. The answer depends on the task that needs to be performed [4]. A design discipline that has been followed is that the user needs to approve agent actions before the agent starts any autonomous operation. Additionally the option of selecting agent assistance explicitly is always a user possibility.

A case of an autonomous agent operation is the provision of assistance at the domain of application level. When the agent 'thinks' that the user could make use of agent recommendations with regard to a search request, the agent can provide items recommendations according to the utilised recommendation model (e.g. collaborative information filtering), after the user has given approval to such an agent operation. A case of a collaborative agent operation could be the guidance of the user when he 'seems' (by the evaluation of user states) not aware of (or does not understand) the context of some user interface facilities (which are considered representative tasks). In GAIA, such a case is the task of evaluation of search results (in the GAIA application user interface the complete view of search results is provided in consequent windows). The detailed evaluation of search results is considered a representative task for users. Unless the agent evaluates that the user has a high skill (user model), it will prompt him to examine search results in detail. It is essential that users have the option to avoid or even forbid such operations, controlling the autonomy of the agent. Generally, the most probable to be beneficial alternative assistance proposals are presented to the user, who can select among them the one that meets his needs (if there is such).

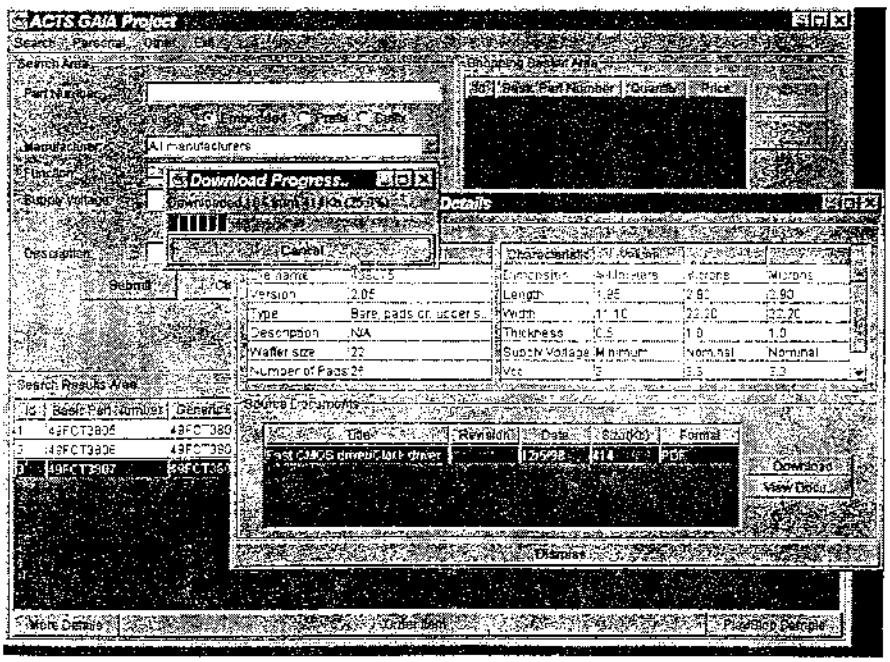

Figure 6: In the GAIA user interface, the complete view of search results is provided in consequent windows.

Another important issue relatively to assistance selection is that the agent need to be capable of learning which of the alternative assistance suggestions are most beneficial to the user and adjust its proposals accordingly. In order to measure the effectiveness of agent suggestions a number of evaluation criteria have been selected.

The precision of assistance is an important criterion in the model. Precision is defined as follows:

Precision = (number of correct suggestions) / (number of suggestions)

The actual evaluator of assistance selections is the user. When the user selects a specific means of assistance, it is considered that this is useful to the user. Otherwise the user can select that he doesn't need assistance or that the assistance proposed is not is not appropriate. The first case of the user respond affects the reactivity criterion which is defined as follows:

Reactivity = 1 – ("user doesn't need assistance" occurrences) / (number of suggestions)

In the case, which the user states that the selected assistance was not appropriate, the disruption criterion is affected, which is defined as follows:

Disruption = 1 – ("not appropriate assistance" occurrences) / (number of suggestions)

Those criteria can be also evaluated per agent state. In this way a more detailed view of the agent's assistance selection behaviour can be modelled. The existence of these criteria also allows the designer to map a variety of alternative assistance selections to the agent, because they can be dynamically evaluated and interchanged.

5. Agent Functionality and Application Integration – Software Engineering Issues

The issue of integrating agent functionality to an application has some very important software engineering aspects [Σφάλμα! Δεν έχει οριστεί σελιδοδείκτης.]. The manner by which the agent can perceive the changes in the software environment depends on the technology by which this is developed. The same is the case when the agent needs to affect its environment. In most cases of personal assistants, the application and the agent are the same software package (Microsoft Word, 31). Although the agent may appear in a separate window in some cases or express its states to the user in a form of feedback, it is not clear whether these agents are modelled in a generic manner towards their employment in other applications as well. Moreover there are other applications which claim to have agent functionality which is unfortunately hidden to a conventional user interface. This chapter attempts to highlight the aspects of the interaction agent design that are domain or application dependent (with emphasis on the application user interface, than the underlying brokerage system) pointing out to future research relatively to the development of an interaction agent development environment. The examples given are from the GAIA development experience in which the Java language was used.

5.1. Perception Subsystem

At the lowest level of abstraction, the perception subsystem needs to be able to capture low level user (underlying system) actions, which are user interface (system) dependent. Such low level user actions can be the pressing of a button, the filling of a form, the selection of an item in a scrolling list, a menu, etc. For each action (event in event-driven languages terminology, as Java) which has been identified as observable by the agent, the application user interface has to timely notify the perception subsystem that it occurs. This implies that in order for the application user interface to plug-in the interaction agent functionality, the application user interface software will have to import (include) the perception subsystem software package (library) in order to access its functionality. Therefore the application user interface development needs to be frozen in order to start plugging in agent functionality.

There are many other aspects of application user interface design which need to be finalised in order to provide input to the design of the agent functionality. The design of the interaction model is an example

case. According to the interaction agent implementation of the interaction model (based on TKS), the agent needs to possess some knowledge about task plans in advance. This knowledge includes user states, user tasks (related to tasks), central and representative knowledge components (user interface components that create observable events). The designer must model this knowledge in a manner that plan knowledge is searchable and easy to modify. In GAIA the implementation of plan knowledge was modelled in XML, but there was no time, in the time constraints of the project, to model this knowledge in a user-friendly manner for the designer, in case there is a need to modify this knowledge. The development of a plan editor that would enable the designer to enter plan knowledge in a graphical user interface is being developed.

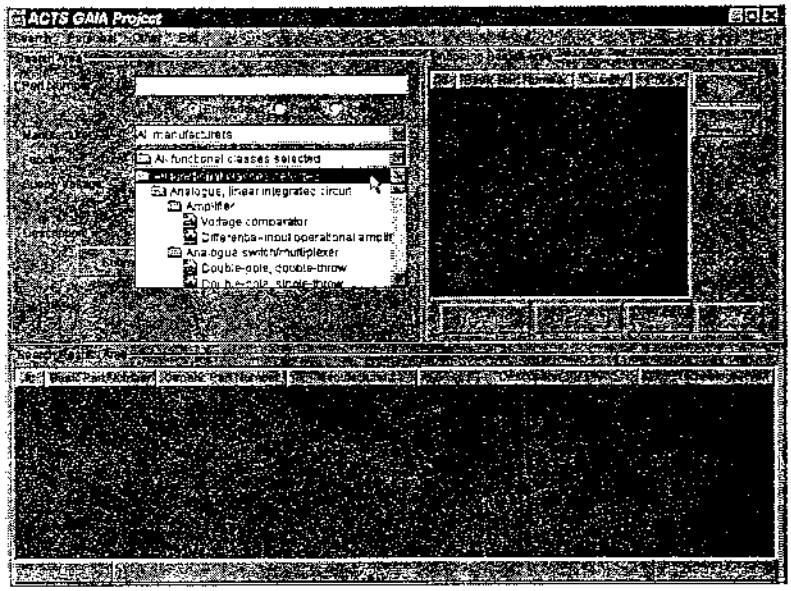

Figure 7: A snapshot of the GAIA user interface: a) it is divided in task spaces; b) for tasks that it was necessary that they should be performed sequentially, the user has no access to the following task spaces (incremental user interface).

In a higher level of abstraction, the actual form of the user plans may be user interface dependent as well (the latter actually affects the former). In a robust user interface design, high level goals should be designed in a manner which enables the user to perform them from different access points, thus the complexity of the plan recognition process is reduced. The design of the application user interface based on task analysis and usability principles is an important step in contributing towards reducing the agent's plan recognition process complexity [25]. An example of a usability feature that has been used in the design of the GAIA user interface was that it was incremental: for tasks that it was necessary that they should be performed sequentially, the user had no access to the following task spaces [25]. This technique contributes other user interface design aims, as error avoidance.

5.2. Action subsystem

The availability of the application interface (underlying system) remains an interaction agent design curse in the case of adding agent actions to the overall system functionality. If agent actions need to be performed by the underlying system, then the problem might be much easier, especially if the underlying system API is determined from the first steps of the design (as must happen with CORBA). If agent actions are targeted to the user then a number of user interface design issues need to be solved.

A first issue is whether the agent has a (partially) separate user interface than the application user interface. Relevant approaches include exhibition of emotional agent states [31], part of the user interface in a separate window (Microsoft Word) and personas / avatars in the cases of virtual environments. Personas employ features that add expressiveness and believability to the look of the user interface, features that have proven to enhance user motivation and satisfaction [29].

Another related issue is that of the dialogue models that are used by the agent in order to provide user interface assistance to users. These models need to be really configurable and reusable, adjusting to the information which is constructed dynamically by the agent. The existence of a variety of dialogue model representations would be a desirable feature of an interaction agent development environment, as a basic feature of any user interface is the ease of configuration and alternative means of representation of information.

6. Conclusions

The paper presented the conceptual and functional architectural design of the GAIA interaction agent. Interaction agents can support and enhance the interaction between the user and the brokerage system by acting as both assistants of the computer-based system and (following the metaphor of the human assistant) of the application domain. The interaction agent conceptual architecture composes of the internal agent components and the agent interface components. The latter are dependent to the design of both the application user interface and the underlying brokerage system. Both types of components were presented with relevance to a variety of agent design issues: action selection, learning from experience, 'watching over the shoulder' process, user profiles, as well as technological and software engineering issues.

In the context of brokerage environments, interaction agents are a promising solution to well-known problems of information load, usability difficulties, and recommendation mechanisms. Brokerage systems are characterised by complex user interfaces offering innovative and rich functionality. The need for personalisation of information services and development of mechanisms that push the information to the customer is important for content providers. On the other hand, customers often find themselves trapped in complex and difficult to use systems. Interaction agents can be a tool for system developers towards satisfying those requirements. An important issue towards that purpose is the identification of software engineering issues and the implementation of a development environment for interactions agents.

This work is being carried out in the context of a European Union ACTS (AC221) project GAIA [1, 2].

6. References

[1] ACTS (AC221) GAIA, Generic Architecture for Information Availability: http://syspace.co.uk/GAIA.

[2] Bessonov M., Hands J., Patel A., Smith R., An Inclusive and Extensible Architecture for Electronic Brokerage", to be presented to Hawaiian International Conference on system Sciences HICSS-32, 1999.

[3] Benyon D., Murray D., (1993) Adaptive Systems: From Intelligent Tutoring to Autonomous Agents, Knowledge Based Systems 6(4), 197-219

[4] Brown S. M., (1998) A Decision Theoretic Approach to Interface Agent Development. PhD Dissertation. Air Force Institute of Technology, Air University.

[5] Byerley, P.F. Barnard, P.J. and May, J., editors (1993) Computers, Communication and Usability: Design issues, research and methods for integrated services. (North Holland Series in Telecommunication) Elsevier: Amsterdam.

[6] Buckingham Shum, S Jorgensen, A. Hammond, N. and Aboulafia, A. (eds.) (1993) Amodeus HCI Modelling and Design Approaches: Executive Summaries and Worked Examples, http://www.mrc-cbu.cam.ac.uk/amodeus/.

[7] Cesta A., Collia M., D'Aloisi D., (1998) An interactive agent-based meeting scheduler. First International Workshop on Interaction Agents, IA'98, L'Aquila, Italy.

[8] Communications of the ACM (1997). Special Issue : Recommender Systems, Vol. 40, Number 3, March 97.

[9] DeWitt95 DeWitt R., (1995), Vagueness, semantics, and the language of thought.

[10] Diaper D., (1997) HCI and Requirements Engineering - Integrating HCI and Software Engineering Requirements Analysis, SIGHCI Bulletin Vol. 29 No 1, January 1997.

[11] Dix A., Finlay J., Abowd G., Beale R., (1993) Human-Computer Interaction, Prentice Hall 1993.

[12] Firefly: http://www.firefly.com/.

[13] Foner L., What's an agent anyway?, (1994) online paper:

http://foner.www.media.mit.edu/people/foner/Julia/Julia.html .

[14] Franklin S., Graesser A., (1996) Is it an agent or just a program? A taxonomy for autonomous agents, Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages. Springer-Verlag. 1996.

[15] Goldberg, D.: Oki. B.: Nichols, D.: Terry, D. B. (1992) ''Using Collaborative Filtering to Weave an Information Tapestry," Communications of the ACM, December 1992, Vol 35, No 12, pp. 61-70.

[16] Hayes-Roth B. (1991) An Integrated Architecture for Intelligent Agents SIGART Bulletin 2, 1991, 82-84.

[17] Hayes-Roth B (1993) Opportunistic Control of Action in Intelligent Agents. Knowledge Systems Laboratory, Stanford University, IEEE Transactions on Systems, Man and Cybernetics v. 23, 1993 pp 1575-1587.

[18] Haynes T., Sen S., Arora N., Nadela R., (1996) An automated meeting scheduling system that utilises user preferences.

[19] IONA Technologies (1997) . Orbix 2 Reference Guide, IONA Technologies PCL, March 1997.

[20] Jennings N. R., J. Corera, I. Laresgoiti, E. H. Mamdani, F. Perriolat, P. Skarek and L. Z. Varga, (1996) Using ARCHON to develop real-word DAI applications for electricity transportation management and particle accelerator control, IEEE Expert, December 1996.

[21] Johnson P., Johnson H., Waddington R., Shouls A., (1988) Task-Related Knowledge Structures: Analysis, Modelling and Application. People & Computers: from research to implementation. Cambridge Press pp. 35-62, 1988.

[22] Johnson P., Johnson H., (1988) Task Knowledge Structures: Psychological basis and integration into systems design, Acta Psychologica, 78, pp. 3-26.

[23] Johnson W. L., Learning from agents (1998) First International Workshop on Interaction Agents. IA'98, L'Aquila, Italy.

[24] Koutsabasis P. Darzentas J.S. Spyrou T. Darzentas J. (1998) An interaction agent in a brokerage environment, First International Workshop on Interaction Agents, (IA'98), L'Aquila, Italy.

[25] Koutsabasis P. Darzentas J.S. Spyrou T. Darzentas J. (1999) Facilitating User–System Interaction: the GAIA Interaction Agent, to be presented to Hawaiian International Conference on system Sciences HICSS-32, 1999.

[26] Laird J., Hucka M., Huffman S., (1991) An analysis of Soar as an integrated architecture. SIGART Bulletin 2, 85-90, 1991.

[27] Lashkari Y., Metral M., Maes P., (1994) Collaborative Interface Agents, In proceedings of the National Conference on Artificial Intelligence, 1994.

[28] Lester J. C., Stone B. A., (1997) Increasing believability in animated pedagogical characters, First International Conference on Autonomous Agents (Agents '97), California, USA.

[29] Lester J. C., Converse S. A, Kahler S. E., Barlow S. T., Stone B. A., Bhoga R. S. (1997) The personal effect: Affective impact of animated pedagogical agents, First International Conference on Autonomous Agents (Agents '97), California, USA.

[30] Maes P., (1994) Modelling Adaptive Autonomous Agents, Artificial Life Journal 1(1, 2), MIT Press (C. Langton, Ed.)

[31] Maes P. (1994) Agents that Reduce Work and Information Overload, Communications of the ACM July 1994, Vol. 37, No. 7.

[32] Maes.P., (1996) "Intelligent Software: Easing the Burdens that Computers Put on People." IEEE Expert. Special Issue on Intelligent Agents. edited by Jim Hendler, December 1996.

[33] Moukas A., (1996) Amalthea, Information Discovery and Filtering using a Multiagent Evolving Ecosystem, Proceedings of the Conference on Practical Application of Intelligent Agents & Multi-Agent Technology. London. 1996.

[34] Negroponte, N. (1970) The Architecture Machine; Towards a More Human Environment, MIT Press, 1970.

[35] Nielsen J., (1992) The Usability Engineering Life Circle, IEEE Computer, March 1992

Nielsen J., (1992) Usability Engineering, Academic Press, 1992.

[37] Oard D. W., (1996) A conceptual framework for text filtering, Technical Report. University of Maryland. College Park.

[38] Pearl J., (1988), Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference, San Mateo CA, Morgan Kaufmann

[39] Petrie, J. C. (1995) Agent-Based Engineering, the Web and Intelligence, IEEE Expert - Intelligent Systems and their Applications, Vol. 11, Issue 6, December 1996, pp. 24-29.

[40] Resnick, P., Iacovou, N., Sushak, M., Bergstrom, P., and Riedl, J.GroupLens: An open architecture for collaborative filtering of netnews. Proceedings of the 1994 Computer Supported Collaborative Work Conference.(1994).

[41] Rickel J., Johnson W. L., (1997) Integrating pedagogical capabilities in a virtual environment agent, First International Conference on Autonomous Agents (Agents '97), California, USA.

[42] Smith R., Maroulis D., (1996) GAIA Domain and User Requirements, Deliverable 0301,

http://www.syspace.co.uk/GAIA.

[43] Sullivan J. W., Tyler S., W., (Editors) Intelligent User Interfaces, ACM Press Frontier Series, 1991.

[44] Tate A., (1995) O-Plan Knowledge Source Framework, Artificial Intelligence Applications Institute, University of Edinburgh, Technical Report, March 1995, Internet Page: http://www.aiai.ed.ac.uk/~oplan/oplan/oplan-doc.html.

[45] Wang H., Wang C.(1997), A Taxonomy of Intelligent Agents, in Intelligent Agents in Nuclear Industry, IEEE Computer November 1997, pp. 31.

[46] Whatley J. E., Scown P. J. A., (1998) A method to specify interactions between human operators and multi – agent systems, First International Workshop on Interaction Agents, IA'98, L'Aquila, Italy, 1998.