AN ARCHITECTURE FOR DISTRIBUTED EXPERT SYSTEMS BASED ON COLLABORATIVE AGENTS

N. Galal[1], Y. Abdelhamid[2], A. Rafea[3]{nevien, yasser, rafea} @mail.claes.sci.eg

ABSTRACT

This paper addresses the possibility of building an agent-based architecture that enables users to interact with multiple knowledge-based system applications through an intelligent interface agent. This intelligent interface works as a personal assistant that works on behalf of the knowledge-based system user, and makes the necessary actions to interact and communicate with multiple knowledge-based system agents transparently from the user. This approach highly increases the usability of knowledge-based system applications, and introduces a new methodology for knowledge-based system building, based on collaborative agents. In order to prove the applicability of the presented architecture we have applied the system on two applications in the domain of agriculture namely diagnosis and irrigation expert systems.

KEYWORDS : Intelligent agents, Knowledge-based systems, Human-computer interaction, Distributed AI.

1 INTRODUCTION

In recent years there has been considerable interest in building complex problem solving systems as groups of cooperating experts. A cooperating expert system is composed of a group of agents; agents are grouped together to form communities that cooperate to achieve the goals of individuals and of the system as a whole. Most of the problems require or involve multiple agents, the agents will need to interact with one another, either to achieve their individual objectives or else to manage the dependencies that ensue from being situated in common environment [1].

Intelligent interfaces are one of the applications of intelligent agents, through which the software agent works intimately with the user, functioning as a personal assistant. An intelligent interface can be defined as an intelligent entity mediating between two or more interacting agents who possess an incomplete understanding of each others' knowledge and form of communication [2]. A good interface will lead to better user/expert system interaction and task performance [3].

In this paper we introduce a distributed architecture for knowledge-based systems using a community of co-operative intelligent agents. An intelligent interface agent has been designed for managing the interaction between the KBS user and the KBS agents. This interface agent acts as a personal assistant to the KBS user, relieving him from the burden of managing the interaction with these agents. Eventually, we can add more knowledge-based system applications without affecting the manner of interaction between the user and the system.

2 Agents and Distributed AI

The concept of an *agent* has recently become important in Artificial Intelligence (AI) and its relatively youthful sub-field of Distributed AI (DAI) [4]. Agents are being used in an increasingly wide variety of applications, ranging from comparatively small system such as e-mail filters to large, open, complex, mission critical systems such as air traffic control [5]. Research in DAI has proposed two approaches for cooperative problem solving. The first of these is called *task sharing*, where a problem is divided into a set of smaller problems each of which could be assigned an agent and where an agent works independently in order to solve the problem assigned to it. Such an approach is suitable in task-oriented domains, which are domains in which agent activities could be defined in terms of a set of tasks. The second approach is called *result sharing* where agents assist each other by sharing partial solutions that allow them to complete their tasks [6]. Unlike, task-sharing agents, result-sharing agents are highly interdependent, are not autonomous, and rely on heavy communication among themselves to accomplish a given task.

2.1 Interface agents

The objective of interface agents' research is to provide indirect management for human-computer interfaces. Current computer user interfaces only respond to direct manipulation, i.e. the computer is passive and always waits to execute highly specified instructions from the user. It provides little or no *proactive* help for complex tasks or for carrying out actions such as searching for information that may take an indefinite time.

[1] Central Laboratory for Agricultural Expert systems (CLAES).

[2] Institute of Statistical Studies & Research (ISSR) – Cairo University.

[3] Department Of Computer Science – American University In Cairo.

The *goal* is to migrate from the direct manipulation metaphor to one that delegates some of the tasks to software interface agents in order to accommodate novice users. The *hypothesis* is that these agents can be trusted to perform competently some tasks delegated to them by their users.

The main functions of an interface agent include:

1- Collecting relevant information from the user to initiate a task.
2- Presenting relevant information including results and explanations.
3- Asking the user for additional information during problem solving.
4- Asking for user confirmation, when necessary.

The knowledge held by an interface agent:

1- A model of the user's goals and preferences pertaining to a task.
2- Knowledge of the relevant task assistants that can perform the task.
3- Knowledge of what must be displayed to the user and in what way.
4- Protocols for interacting with relevant task assistants.

2.2 Agent Communications

For interoperability, agents should be able to communicate with other agents supplied by different implementers or vendors. The obvious solution is a lingua franca, whereby all the agents who implement the same lingua franca can communicate. To approach this idea, an agent communication language (ACL) must be standardized so that different parties can build their agents to interoperate. The Knowledge Query and Manipulation Language (KQML) is a language and an associated protocol to support the high level communication among intelligent agents. KQML expressions consist of a content expression encapsulated in a message wrapper, which is in turn encapsulated in a communication wrapper. Thus the language is thought of as being divided into three layers: content, message and communication. The content layer contains an expression in some language, which encodes the knowledge to be conveyed. The message layer adds a set of features that describe the content, e.g. the language it is expressed in, the ontology it assumes and the kind of speech act it represents (query assertion, etc.). The communication layer adds a set of low level communication features that include the identity of the sender, and receiver and whatever communication is meant to be synchronous or asynchronous [7]. Arcol is another ACL based on speech acts [8], [9]. Arcol was the basis for the first version of the proposed standard of the Foundation for Intelligent Physical Agents (FIPA), and many of its components survive in the second version as well. Agents conforming to the FIPA specification can deal explicitly with actions. They make requests, and they can nest the speech acts.

2.3 Agents and Expert Systems

In recent years there has been considerable interest in the possibility of building complex problem solving systems as a groups of cooperating experts. A cooperating expert system is composed of a group of agents, each of which contains an autonomous knowledge based system. Typically, agents will have expertise in distinct but related domains. Agents co-operate together to solve a given problem and achieve the goals of individuals and of the system as a whole. An important research project in this direction was transforming expert systems into a community of cooperating agents [10] where the aim of this work was to construct a community of cooperating agents from two standalone and pre-existing expert systems. This was achieved by using the GRATE system, which is a general framework for constructing communities of cooperating agents for industrial applications. The cooperating community worked together to diagnose faults that occurred in the real particle accelerator process.

3 A Distributed Expert System Based on Collaborative Agents

The basic idea of our work is to convert the pattern of interaction between the user and a number of knowledge-based systems from that where the user interacts with each individual knowledge-based system separately, to another pattern where the user has only one interface, through which he can interact with multiple knowledge-based systems without having to worry about the requirements of each one.

Our approach is to model each KBS in the current environment as an agent with its own knowledge, and services. Two other special agents are introduced: *Interface agent*, and *Coordinator agent*. The interface agent is responsible for managing the interaction between the user and proposed environment, through handling request messages asking the user for a required input, and response messages displaying the reply. The coordinator is an agent that is intelligent enough to determine the user requirements by analyzing the users inputs, and to specify which KBS agent to contact for achieving these requirements, including calling intermediate agents for other services.

3.1 Overall Architecture Of The Proposed Environment

As shown in Figure [1] the proposed architecture consists of a number of agents. The following is a brief description of the components that comprise the proposed architecture.

3.1.1 User Interface Agent:

The internal structure of the proposed environment

is completely transparent to the user. The only component that is visible is the user interface agent. The user interface agent has two main services: the first is to collect user input in the form of observations, events, data, or requirements, and sending these inputs to the coordinator agent. The second service is to receive and display the replies that could come back from the coordinator agent.

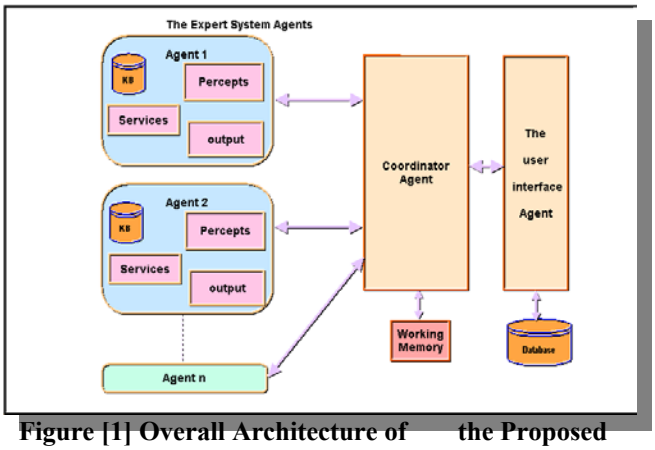

Figure [1] Overall Architecture of the Proposed Environment

3.1.2 The Coordinator Agent:

The coordinator has its own knowledge about the KBS agents in its environment, and the services offered by each of them. It is stimulated by any message received from the user interface agent, or any of the KBS agents. So, any input introduced by the user interface agent, or a reply from a KBS agent will activate its inference process. The action of the coordinator is a message sent to one or more KBS agents, asking for their services, or a message to the user interface agent as a result of processing. This cycle will stop when no more services are required, or there are no more requests from the user.

3.1.3 KBS Agents

The KBS agents are the components that are doing the real work in the proposed environment. In other words, they are achieving the requirements of the user in the form of services.

3.1.4 The Database

The database component is used as a common store of static data that is often required by KBS agents. For example, in the domain of agriculture, the database is used to store data about farms like plantation data, climate data, soil data, and water data.

3.2 Agent Infrastructure

The proposed agent environment presumes an infrastructure within which the agents operate and interact. Two major facilities comprise this infrastructure: *Common Ontology*, and *Communication Protocol*.

3.2.1 Common Ontology:

Making agents understand each other is a major challenge. Agents may have different meanings for the same concept, or they may have different concepts with the same meaning. To overcome this difficulty, there must be some sort of shared knowledge about the domain of discourse that made available to all components in the proposed environment.

In our proposed environment, this shared knowledge is provided through a *common Ontology* that is a set of concepts and attributes, as well as relations between these concepts and attributes. We used this ontology as the underlying background knowledge that is common for all participating agents.

3.2.2 Communication Protocol:

As mentioned before KQML is one of the pioneer research projects in the field of agent communication languages. KQML provides a large set of primitives through which agents may *tell* facts to other agents, evaluate expressions for other agents or *subscribe* to services provided by other agents. KQML suffers from poorly defined semantics. As a result, many implementations have been introduced, but each seems unique. This makes communication difficult, and KQML agent may not be understood.

In our proposed environment, we use HTTP as the underlying transport protocol, and we use XML language to formulate messages passing between different agents. According to this approach, messages are not tied to a particular implementation, and basically we are not in need to implement a standard communication language like KQML.

4. Design of the Proposed Environment:

After we have had a general idea bout the proposed approach, and the overall architecture of the proposed agent environment. The following is a detailed discussion of the components constituting the environment.

4.1 User Interface Agent:

The user interface agent has two sub-components; request model, and response model. The request model formulates the list of inputs that the user can provide during the session. The expected inputs are derived from the common ontology used by the knowledge-based system agents in the form of properties related to some concepts in the domain of discourse, and the acceptable values for each of these properties.

At any time during the session, the user can use this unified interface to convey some inputs to the system. These inputs can be events like rain or wind,

observations on the plant like leaf color or root shape, current date, etc. The user doesn't know which knowledge-based system will reply to these inputs, or which will provide its services. Finally, the request model submits the user inputs to the coordinator agent in a request message.

The response model collects and displays the results or replies coming from the coordinator agent as a reply from one or more knowledge-based system agents. These results and replies can be classified into one of the following categories:

- Predicted disorder name(s).
- Confirmed disorder name(s).
- Treatment materials, quantities, dates and methods of using these materials.
- Irrigation schedule or part of it.
- Fertilization schedule or part of it.
- A plant care operation to be done.

4.2 The Coordinator

The coordinator manages the interaction between the user through the user interface agent, and the different knowledge-based system agents in the environment. This is done through messages sent to the various agents requesting for some service, or supplying some information back to the user interface agent. The coordinator has the knowledge that makes it able to decide which service to be requested for the current session. This knowledge is represented as rules that work on the current property values stored in the working memory, and maps them to one or more services provided by other agents. Figure [2] displays a sample of this knowledge represented in XML format.

The coordinator interacts with the knowledge-based system agents in the following manner:

1. First, the coordinator formulates the input data existing in the working memory into an appropriate request message and sends it to the related agent(s).
2. Second, it receives the reply messages from participating agent(s), updates the working memory and starts a new cycle of sending new messages to the same or other agents if needed, by redirecting the results to them.
3. Third, it sends the final results to the user through the user interface agent in the form of response message.

```
<Agent Name="diagnosis">
        <Tuple Cpt="leaves_observations" Prop="color/">
        <Tuple Cpt="leaves_observations" Prop="shape/">
        <Tuple Cpt="stem_spot" Prop="exist/">
        <Tuple Cpt="stem_observations" Prop="shape/">
        <Tuple Cpt="fruit_observations" Prop="color/">
        <Tuple Cpt="fruit_observations" Prop="shape/">
```

Figure [2] Sample of coordinator knowledge.

Figure [3] displays a sample message sent to diagnosis agent. As we can see, the message has three parts, Header, Body, and Fault. The Header contains information about the sender, receiver agent, and the requested service. The Body contains working memory contents related to the designated agent. The Fault element is used to provide information about errors that occurred while processing the message. By nature this element can only appear in answers (response messages).

4.3 KBS Agents:

In our proposed environment, we imposed a certain structure on KBSs to achieve our goal for supporting intelligent communication between them. The main architecture of the KBS agents is:

The knowledge base: Which in its turn consists of Domain concepts, and Domain relations:

Domain concepts are structured collections of domain terms (e.g. concept and its properties and values). Concepts can have properties that are defined through their names and the values that they can take.

Domain relations represent the relation between concepts/properties defined in the domain of discourse, these relations are represented in the form of: Rules, tables, or functions.

Services: Which are the functions that the knowledge-based system agent provides. Each service has a unique name, percepts which are the inputs needed to stimulate the service (indicate when that service should be activated), and output of that service.

```
<Message>
<Header>
        <From>Coordinator</From>
        <To>Diagnosis_Agent</To>
        <Service>Generate_Hypothesis</Service>
</Header>
<Body>
<Tuple Cpt="Leaves_observations"Prop="color" Val= "Brown"/>
<Tuple Cpt="Stem_Observations" Prop="Shape" Val= "etiolated"/>
<Tuple Cpt="Stem_Spot" Prop="Exist" Val= "yes"/>
</Body>

<Fault>
</Fault>
</Message>
```

Figure [3] A sample diagnosis agent message.

KBS agents are implemented as COM (Component Object Model) objects, so they can be accessed as web

services. A number of methods have been added to these components so that knowledge base can be manipulated easily.

5. CASE STUDY

As we have mentioned early in this paper, the proposed architecture has been tested in the domain of agricultural expert systems. Two expert system agents have been implemented: Irrigation agent, and Diagnosis agent for cucumber cultivation under tunnels. The irrigation agent calculates water requirement for the plant, and generates partial and full irrigation schedules. It is worth mentioning that some circumstances may drive the irrigation agent to revise its previously suggested irrigation schedule, like changes in the weather or having certain disorders.

The diagnosis agent analyses the input observations and farm data, and provides the user with information about the possibility of having a certain disorder(s), and according to extra information provided by the user, the diagnosis agent may confirm the existence of one or more of these disorders. Other agents (treatment agents) can be developed for providing suitable treatment schedules for the confirmed disorders specified by the diagnosis agent.

As shown in Figure [4], the user provides basic information about the case, in addition to some observations that according to the diagnosis agent's knowledge are manifestation for some disorders. At this point, only diagnosis agent is involved, and replies with the names of suspected disorders.

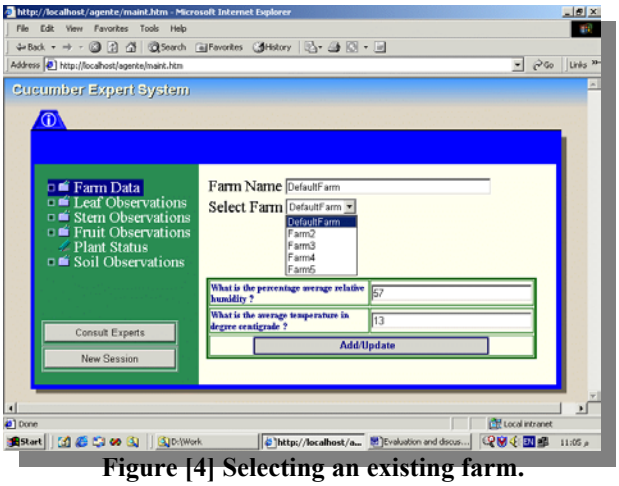

Figure [4] Selecting an existing farm.

As shown in Figure [5], with extra observations provided by the user, the diagnosis agent will have the possibility of confirming one or more of these suspected disorders.

The coordinator agent receives the initial results of the diagnosis agent, and passes the results to other agents, in our case, the irrigation agent.

According to the irrigation agent's knowledge, the confirmed disorder may affect the quantity or schedule of irrigation. So, it revises the proposed schedule, and passes it back to the coordinator agent.

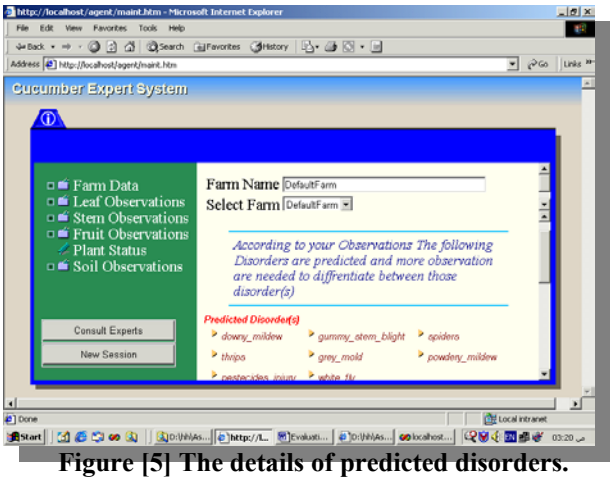

Figure [5] The details of predicted disorders.

The coordinator agent tries to route the new information to expert system agents again, but this time no one replies.

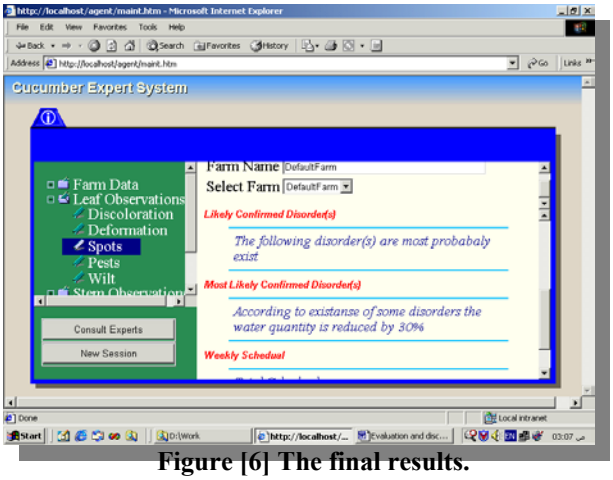

Figure [6] The final results.

The coordinator agent considers this as a final reply, and reports it back to the user interface agent.

As shown in Figure [6] The user interface agent displays the results to the user with a fully detailed report coming from all agents involved in the session.

CONCLUSION

In this paper we have presented an agent-based architecture that enables knowledge-based system users to interact with a number of knowledge-based system applications through an intelligent interface agent.

Through the analysis and evaluation of the

implemented prototype, we have come up with the following conclusions:

Using the presented architecture, the user deals with multiple knowledge-based systems as if they were only one. This approach helped in reducing the interaction between the user and the knowledge-based systems, by making the coordinator agent take over this interaction on behalf of the knowledge-based system user.

Building this standard and unified interface simplifies the design of knowledge-based systems by separating between the interface design and other functions of the system and provides a unified model of interaction between the user and KBSs without having to merge these KBSs into one large, hard to manage KBS.

The proposed architecture is open, in the sense that we can add new knowledge-based system agents without any change in the manner of interaction between the user and the proposed intelligent interface agent.

The proposed architecture is reusable, since we can apply the same environment on different domains of applications.

REFERENCES

[1] Jennings N. R., and Wooldridge M. (2001). "Agent-Oriented Software Engineering", Handbook of agent technology, ed. J. Bradshaw, AAAI/MIT Press.

[2] Chen C., and Roy R. (1998). "Knowledge-based system Technology: Knowledge-based system Interface", in The Handbook of Applied Knowledge-based systems, Chapter 6, edited by Liebowitz, J.

[3] Kuo-Wei Su, hu-Hua Liu, and Sheue-Ling Hwang (2001). "A developed model of expert system interface (DMESI)", Expert systems with Applications (20), pp. 337-346.

[4] Jennings N. R. , and Wooldridge M. (1994). "Agent Theories, Architecture, and Languages: A Survey", proceeding ECAI - workshop on Agent Theories, Architecture, and Languages, pp. 1-32.

[5] Jennings N .R., and Wooldridge M. (1998). "Applications of intelligent agents", Agent technology: foundations Applications and Markets, pp. 3-28.

[6] Chi R. T., Turban E., and King M. Y. (1996). "A Proposed Framework for Distributed Agents in Decentralized Organizations", in Artificial Intelligence in Organizational Design, Modeling, and Control, Eds Robert Blanning and David King, IEEE Computer Society Press, Los Alamitos, CA, pp.172-182.

[7] Finin T., Weber J., Wiederhold G., Genesereth M., Fritzson R., Mckay M., McGuir J., Pelavin R., Shapiro S., and Beck C. (1993). " Specification of the KQML Agent-Communication Language", By The DARPA Knowledge Sharing Initiative External Interfaces Working Group.

[8] Breiter P., and Sadek M. D. (1996). "A rational agent as a kernel of a cooperative dialogue system: Implementing a logical theory of interaction". In ECAI Workshop on Agent Theories, Architectures, and Languages. Springer-Verlag, Heidelberg, Germany, pp.261-276.

[9] Sadek M. D. (1991). "Dialouge acts are rational plans", In Proceedings of the ESCA/ETRW Workshop on the Structure of Multimodal Dialogue, Maratea, Italy, pp. 1-29.

[10] Jennings N. R., Varga L. Z., Aarnts R. P., Fuchs J., and Skarek P. (1993). "Transforming Standalone Expert Systems into a Community of Cooperating Agents ", Engineering Applications of Artificial Intelligence, 6 (4) 1993, 317-331.