ELSEVIER

Expert Systems with Applications 27 (2004) 1–10

Expert Systems with Applications

www.elsevier.com/locate/eswa

A multiagent approach for diagnostic expert systems via the internet

Khaled Shaalan[a,*], Mona El-Badry[b], Ahmed Rafea[c]

[a]*Department of Computer Science, Faculty of Computers and Information, Cairo University, 5 Tharwat Street, Orman, Giza 12613, Egypt*
[b]*Central Laboratory for Agricultural Expert Systems (CLAES), P.O. Box 100, Dokki, Giza, Egypt*
[c]*Department of Computer Science, American University, 113, Sharia Kasr El-Aini, P.O. Box 2511, 11511 Cairo, Egypt*

Abstract

In recent years there has been considerable interest in the possibility of building complex problem solving systems as groups of co-operating experts. This has led us to develop a multiagent expert systems capable to run on servers that can support a large group of users (clients) who communicate with the system over the network. The system provides an architecture to coordinate the behavior of several specific agent types. Two types of agents are involved. One type works on the server computer and the other type works on the client computers. The society of agents in our system consists of expert systems agents (diagnosis agents, and a treatment agent) working on the server side, each of which contains an autonomous knowledge-based system. Typically, agents will have expertise in distinct but related domains. The whole system is capable of solving problems, which require the cumulative expertise of the agent community. Besides to the user interface agent who employs an intelligent data collector, so-called communication model in KADS, working on the client sides. We took the advantage of a successful pre-existing expert systems—developed at CLAES (Central Laboratory for Agricultural Expert Systems, Egypt)—for constructing an architecture of a community of cooperating agents. This paper describes our experience with decomposing the diagnosis expert systems into a multi-agent system. Experiments on a set of test cases from real agricultural expert systems were preformed. The expert systems agents are implemented in Knowledge Representation Object Language (KROL) and JAVA languages using KADS knowledge engineering methodology on the WWW platform.

Keywords: Expert systems; Agents; WWW

1. Introduction

As computer hardware and software become increasingly powerful, so do applications, which used to be considered beyond the scope of automation. To cope with increased demands, software systems are becoming correspondingly larger and more complex (Genesereth & Ketchpel, 1994; Jennings & Varga, 1993; Jennings & Wooldridge, 1995). In recent years, there has been considerable interest in the possibility of building complex problem solving systems as groups of co-operating experts. Distributed Artificial Intelligence (DAI) is the study of how such systems might be built. Historically, empirical research in DAI has focused on three main areas: blackboard architecture (Engelmore & Morgan, 1988; Nwana, 1996), systems based on negotiation (Smith, 1980), and multi-agent planning systems (Durfee and Lesser, 1988). More recently, co-operating expert systems have emerged as a research area of some importance (Nwana & Ndumu, 1999).

Agent-based computing represents an exciting new synthesis both for Artificial Intelligence (AI) and, more generally, Computer Science. It has the potential to significantly improve the theory and the practice of modeling, designing, and implementing computer systems (Jennings, 2000). Since the 1980s, software agents and multi-agent systems have grown into what is now one of the most active areas of research and development activity in computing generally (Jennings & Wooldridge, 1998). There are many reasons for the current intensity of interest, but certainly one of the most important is that the concept of an agent as an autonomous system, capable of interacting with other agents in order to satisfy its design objectives, is a natural one for software designers (Wooldridge & Ciancarini, 2001). Just as we can understand many systems as being composed of essentially passive objects, which have state,

* Corresponding author. Tel.: +2-012-2223377; fax: +2-02-761-7628.
E-mail addresses: shaalan@mail.claes.sci.eg (K. Shaalan); mona@mail.claes.sci.eg (M. El-Badry); rafea@aucegypt.edu (A. Rafea).

0957-4174/$ - see front matter © 2003 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2003.12.018

and upon which we can perform operations, so we can understand many others as being made up of interacting, semi-autonomous agents.

There are several benefits of using the intelligent agent paradigm for software systems. Agents can provide a high level of abstraction for dealing with intelligent systems and distributed systems. The agent abstraction is a natural extension of object-oriented technology, encapsulating the agents knowledge within an active process and providing a standard interface for intercommunications. This leads to another benefit of agent-based systems, interoperability (Genesereth & Ketchpel, 1994). With a common agent communication Language, such as KQML (Finin, Fritzon, Mckay, & McEntire, 1994; Finin, Labrou, & Mayfield, 1997; Labrou & Finin, 1997), programs written as agents can communicate and cooperates with other such programs. In other words, agent-based system architecture provides a consistent interface for intelligent systems to interact with. Finally, agent systems often provide a high-level 'human-like' interface to take the GUI revolution one step further (Moore et al., 1997).

Multi-agent systems are the best way to characterize or design distributed computing systems (Huhns & Stephens, 1999). Multi-agent systems are commonly intended as computational systems, where several (semi-) autonomous entities interact or work together to perform some tasks. There are several motivations for having multiple agent systems (Nwana, 1996). They include:

- To solve problems that are too large for a centralized single agent to do due to resource limitations or the sheer risk of having one centralized system;
- To allow for interconnecting and interoperation of multiple existing legacy systems such as expert systems and decision support systems;
- To provide solutions to inherently distributed problems such as distributed sensor networks (Durfee and Rosenschein, 1994) or air-traffic control;
- To enhance modularity (which reduces complexity), speed (due to parallelism), reliability (due to redundancy), flexibility (i.e. new tasks are composed more easily from the more modular organization) and reusability at the knowledge level (hence shareability of resources).

The rationale for interconnecting computational agents and expert system is to enable them to cooperate in solving problems, to share expertise, to work in parallel on common problems, to be developed and implemented modularly, to be fault tolerant through redundancy, to represent multiple viewpoints and the knowledge of multiple experts, and to be reusable (Huhns & Stephens, 1999). Nevertheless, With the advent of the Internet, many researchers have been taking a closer look at distributed software systems. Recently, a large share of this research has focused on intelligent distributed systems, which have come to be known as multi-agent systems. As a result, new development methodologies specifically designed for multi-agent systems have been introduced (Iglesias, Garijo, & Gonzalez, 1998) and several tools are now available for building multiagent systems (Raphael & Deloach, 2000).

At the Central Laboratory for Agricultural Expert System (CLAES), at the Agriculture Research Center of Ministry of Agriculture and Land Reclamation in Egypt, a number of successful agricultural expert systems were developed and deployed to a large number of users (Rafea, 1995, 1998; Rafea, El-Azhari, & Hassan, 1995; Rafea, El-Azhari, Ibrahim, Soliman, & Mahmoud, 1995; Rafea, Hassan, & Hazman, 2003; Rafea & Mahmoud, 2001; Rafea & Salah, 1994; Rafea, Warkentin, & Ruth, 1991, 1992). These applications are traditional standalone systems. In these expert systems, the knowledge acquired from multiple domain experts that belongs to different disciplines are implemented as an integrated system. The aim of this research is to harness the potential of the agent technology for constructing a community of cooperating agents capable of diagnosing disorders in the agriculture domain. By implementing expert systems as multi-agents that perform their tasks remotely, the expertise can be published on the Web. Expert systems running on the Internet can support a large group of users who communicate with the system over the network.

This paper is organized as follows. Section 2 gives an overall structure of the proposed architecture of our system. Section 3 describes the user interface agent. Section 4 introduces the coordination agent. Section 5 presents the selected expert systems agents. Section 6 shows the ability of the system in producing diagnosis of disorders based on a set of input observations from real-world cases in the multi-agent environment. Section 7 concludes the paper.

2. Overview of the proposed system

Our proposed system considers the two most common synergetic expert system applications—diagnosis and treatment. From the architectural point of view, the system provides a framework to coordinate the behavior of several specific agent types. The society of agents in our system consists of *expert system agents* (diagnosis agents, and a treatment agent) working on the server side, The diagnosis knowledge is distributed among several agents, each agent is an autonomous expert for a certain domain knowledge. Typically, agents will have expertise in distinct but related domains. The whole system is capable of solving problems, which require the cumulative expertise of the agent community. In addition to the *user interface agent* who employs an intelligent data collector working on the client sides. Those agents communicate by passing messages to the *coordination agent*. The clients communicate with the server through socket connection via the Internet. At the client side the user interface agent send requests for

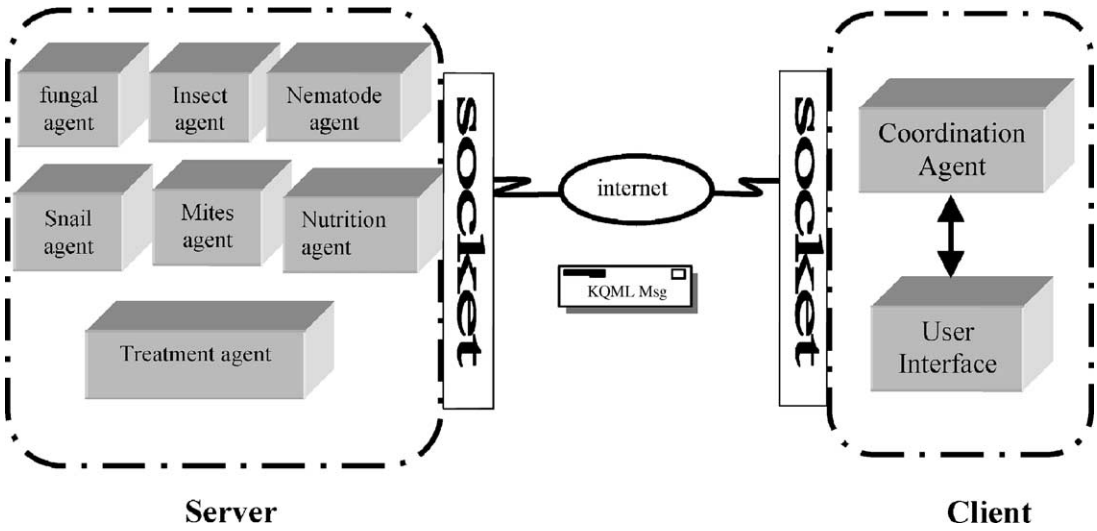

Fig. 1. The society of agents.

diagnosis/treatment services to the server. Answers are sent back to clients. We have applied our system in the domain knowledge of diagnosing disorders of the grapes crop, see Fig. 1.

A diagnosis agent is responsible for producing diagnosis of disorders based on a set of input observations. The treatment agent, who is responsible for finding a suitable recommendation to treat a certain disorder taking into consideration the possible causes of this disorder, provides a set of operations to be performed in order to treat the disorder and remove the causes. The treatment agent can only be consulted after a successful diagnosis session.

In the early stage of this research, we have designed the system with only two expert agents, diagnosis and treatment. But later on, we discovered that the diagnosis agent combines different specialization that are well defined in the agricultural domain. Consequently, we improved our architecture by furtherly decomposing the diagnosis agent. In the agent community decomposition concerns the partitioning of the problem domain into agents. The key problem lies in deciding how the domain is to be partitioned. This decision can only be reached by the extensive consultation with the domain experts. In our case, we figured out that the grape diagnosis consists of six agents, namely: fungal agent, insect agent, nematode agent, snail agent, mites agent, and nutrition deficiency agent. The new architecture concerning diagnosis has the following advantages:

- It tries to simulate what may happen in real world in a very fine-grained manner. Each agent corresponds to a domain expert specialization.
- Partitioning of domain knowledge makes it easy to maintain knowledge, and to improve performance.

In real life, the domain expert who diagnoses a certain disorder can also give the treatment method of this disorder. The treatment methods suggested by different domain experts may conflict with each other such that a decision is needed to decide which one is applied first, and so forth. So, we proposed in our architecture to separate the treatment process from the diagnosis process, and collectively kept into the treatment agent in order to resolve conflicts that may arise in the treatment application.

Expert systems agents include a bilingual KQML translator module. The message transferred through the system takes the KQML format. Expert system agents are implemented in KROL (Shaalan, Rafea, & Rafea, 1998), while the user interface agent is implemented in Java. KQML message needs to be translated to/from Prolog. So we have designed a module which performs this bilingual translation. Similarly, another module for translating user interface messages is designed, which translates KQML to Java and the vice versa.

3. The user interface agent

The user interface agent provides access to the expert systems agents. For portability, this program is a Java applet. When a user browses the WWW page of the system, this applet is downloaded. This is responsible for providing means through which the user could initiate problem-solving sessions, by activating the communication model, as well as handling the presentation of the expert system results. The communication model acts as an intelligent data collector. The user interface agent includes a Java-KQML bilingual translator.

3.1. The communication model

Common-KADS project has addressed many issues related to the development of expert system. One of the most important issues is known as the communication

model (Hoog et al., 1992). One of the best advantages of the communication model is that it completely separates the user interface model from the knowledge model. However, dividing an integrated expert system into a two-component architecture—the problem solving component and an interface or a front-end component—is not usually a straightforward task since the control of the interface is usually managed by the problem-solving component itself. In what follows we show the importance of the communication model in our system.

For instance, in many expert systems, the questions to be asked are determined by answers to previously asked questions. In this case several solutions are possible. The first and the simplest of these, is devising an application specific interface where the user is presented with all possible inputs. Needless to say, this approach will not meet the needs of any reasonably large application, and in addition, will confuse the user. Another approach entails keeping control embedded within the problem solving server application. In that case, the interface will be used to present the user with an input request upon receiving such a request from the server. While this approach might seem reasonable, it suffers from major limitations. First, it relies on heavy communication between the server application and the client front-end, so the user may have to wait for prolonged periods of time depending on the network traffic and bandwidth. In case of synchronous communication, it can engage the server in one connection for an indefinite amount of time, making it impossible for other users to make use of that same server. Although time-out operations could be implemented to avoid indefinite postponement, the application server will still not be fully utilized. In case of asynchronous communication the server will have to maintain extra knowledge such that data inputs could be mapped to application clients. This would be necessary in order to maintain data values that are consistent with its clients.

If however, the interface component employed a communication model that had just enough knowledge about which inputs it should ask about and in which cases, then the client could use this knowledge to collect all needed inputs in an intelligent fashion. Next, send them in one batch to the server for processing. In this case the connection between the client and the server will only be open for the period of sending the inputs, processing them at the server and receiving the output. For most practical applications, this period is usually reasonably short. Meantime, if other clients need to service a request, the request will be placed in a wait queue where the waiting time will be short enough to make that wait transparent. However, care must be taken in selection of the queue size. Otherwise, if the number of the clients for the application grows, then wait times might also grow to unsatisfactory figures.

The implementation of the communication model is inspired by the hierarchical classification model (HC). HC is a problem solving method identified by Chandrasekran (1988) for solving diagnostic types of problems as part of his Generic Task approach to expert system development. In HC, knowledge is represented as hypotheses hierarchically organized in a tree structure such that a general hypothesis is always above more specific ones in the tree. Using a control strategy known as ‘establish and refine’, hypothese are explored top down. If a hypothesis at the top level succeeds (establishes), its immediate descendants are required to be established themselves one by one. This process of attempting to establish the descendants is referred to as ‘refining’ the parent hypothesis. If, on the other hand, a hypothesis fails, then it is said to be ruled out and so are all the hypothese beneath it in the tree (El-Beltagy et al., 1995). In our model, knowledge components for which input is desired, are also organized in a hierarchical fashion.

4. The coordination agent

This component acts as the interface between the user interface agent and other agents. It is responsible for establishing the communication link with the desired expert system agent based on the user request, through the socket connection. In the proposed agent society, TCP/IP is chosen as the low-level transport mechanism for agent-to-agent communication.

In our implementation, a general-purpose client sockets class ESClient is defined that inherits from Socket class of the `java.net` package. Each time the user interface agent needs to communicate with any of the other agents, it sends to the coordination agent a message. The coordination agent in turns creates an instance of this class giving it both the host names of the target agent, and the port number on which the agent’s server socket resides. Host names and port numbers are passed in the first place to the coordination agent as parameters from the applet’s initiating HTML page.

5. The expert system agents

Each expert system agent consists of two basic components, a mediator and the expert system itself. As indicated by its name, a mediator is the module responsible for sending and receiving messages through the network connection. This communication can be established through a server socket. All agent communication takes place through the coordination agent. The expert system consists of three sub-components:

Domain knowledge: it contains the static knowledge of the domain, concepts, properties, and two types of relations. A concept is identified through its name. Concepts can have properties that are defined through their names and descriptions of the values that the property can take. The relations may be relations

between concepts or relations between property expressions.

Inference knowledge: describes what inferences can be made on the basis of the knowledge in the domain knowledge. It describes what inference can made, but not how or when they are made. In KADS the selected knowledge engineering methodology, the following terms are used to denote the various aspects of a primitive inference: inference step, and role. An inference step performs an act that operates on some input data and has the capability of producing a new piece of information as its output. The elements on which inference steps operate or produce are called roles. A role can be an input role or an output role. It acts as a placeholder for domain objects. Domain objects can be linked to more than one role.

Task knowledge: actually describes the steps of executing the knowledge sources in the inference knowledge.

In a previous work (Shaalan et al., 1998), we described the implementation of expert systems in KROL. In KROL, *Webkrol* is the library that handles all the network necessary function.

5.1. The diagnosis agent

As mentioned above there are six diagnosis agents. The user interface agent is responsible for loading the diagnosis communication model in the client side through a WWW browser. The diagnosis session proceeds as follows, see Fig. 2.

1. The diagnosis communication model asks the user about all the observations and collects his responses.
2. The user interface agent calls the Java-KQML translator module to convert the collected observation into a KQML message format.
3. The user interface agent sends this message to the coordination agent.
4. The coordination agent initiates the socket connection with the server and broadcast this message to all diagnosis agents at the server site.
5. At the server side, for each agent, after receiving the diagnosis data, the Prolog-KQML translator translates the KQML message into a prolog term. This message is directly interpreted and executed by the diagnosis expert system.

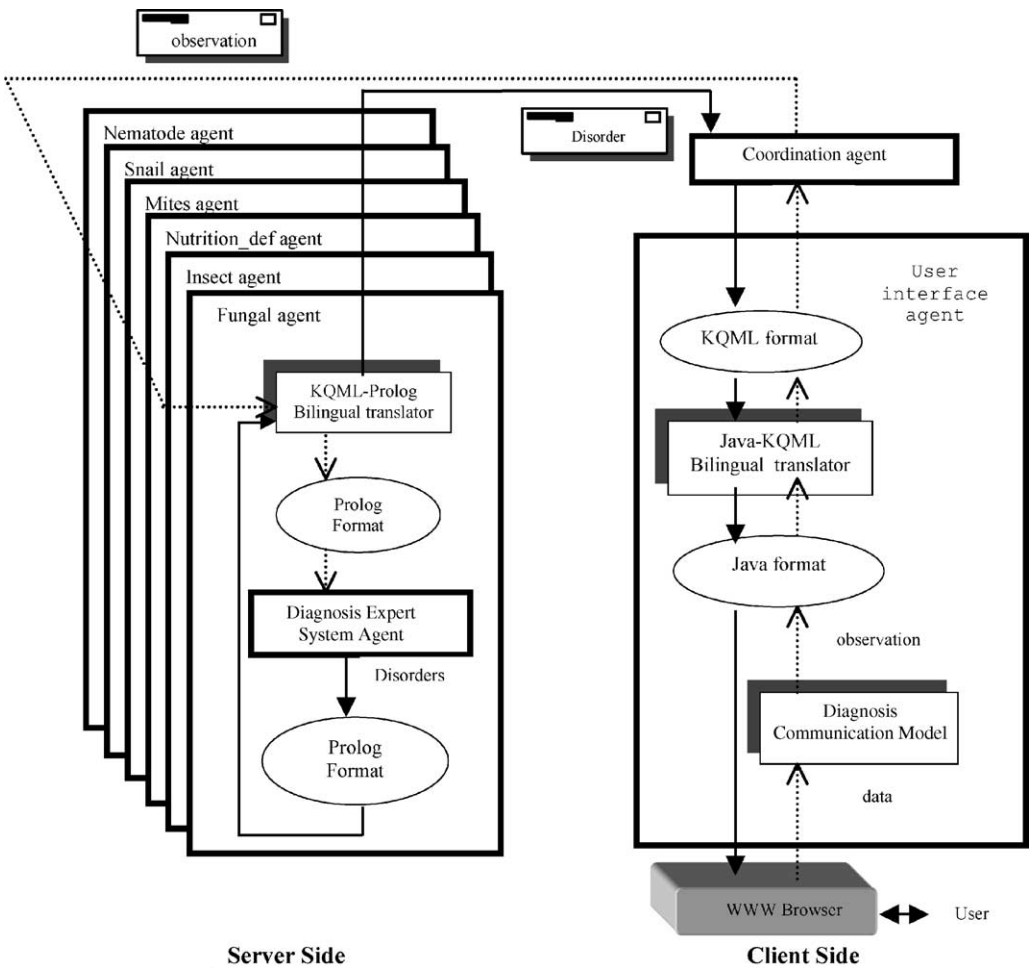

Fig. 2. Observation and disorder flow in the diagnosis session.

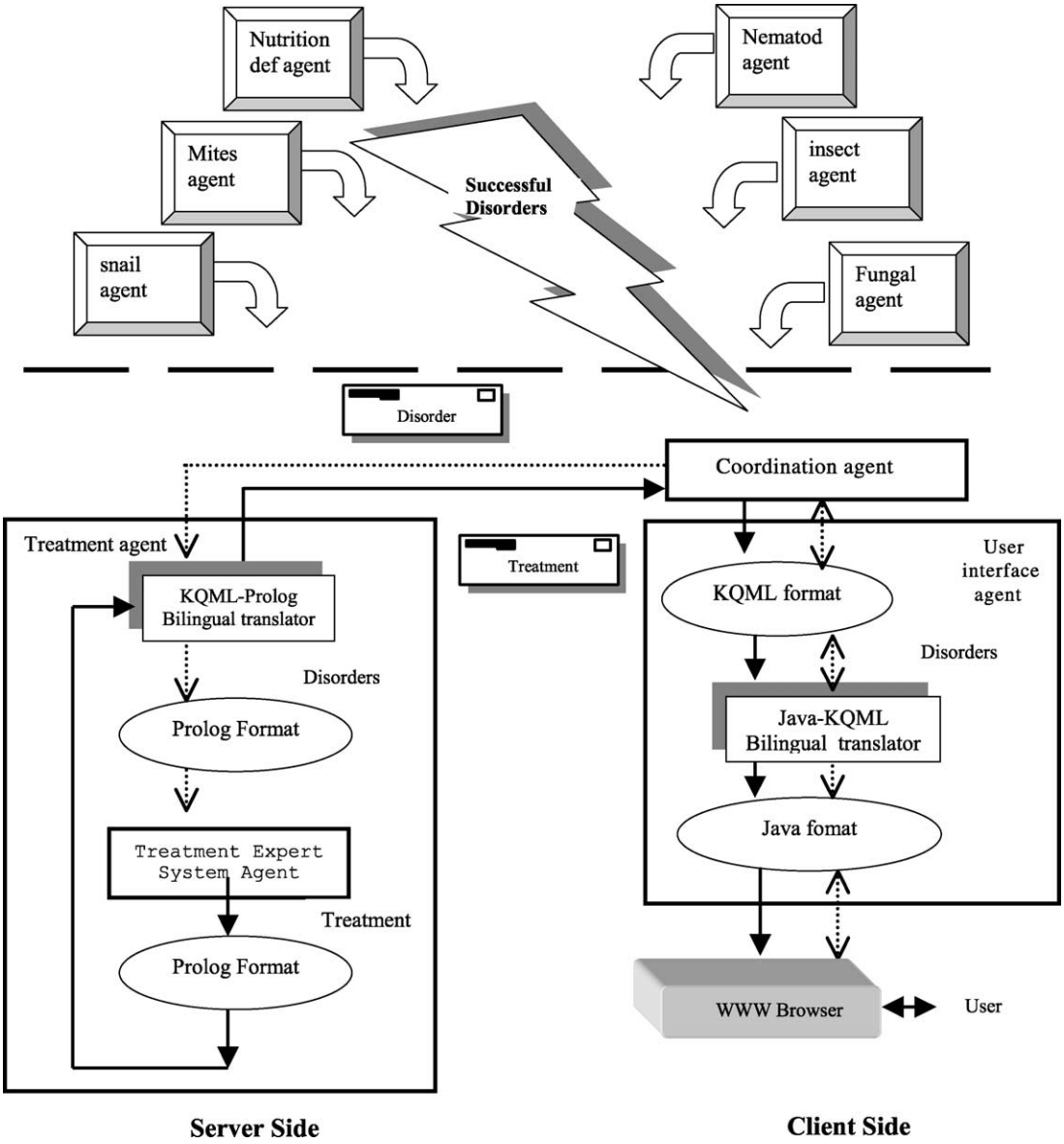

Fig. 3. Disorder and treatment flow from a diagnosis session to a treatment session.

6. The results are converted again to a KQML format and sent back to the coordination agent.
7. The coordination agent collects all the answers and sends it to the user interface agent.
8. The user interface in turn formulates the output and displays it.

5.2. The treatment agent

Treatment agent session can be conducted after a successful diagnosis session. The treatment sessions that correspond to this behavior is described as follows. After the diagnosis session reaches a successful disorders (i.e. at least one of the diagnosis agent find disorders). This information is sent to the coordination agent that transfers it to the user interface agent. If the user requests a treatment for these disorders, the user interface agent calls the treatment communication model and start collecting data related to treatment. Both the treatment input data and the diagnosis result (the confirmed disorders) will transfer to the coordination agent. The coordination agent in turn opens the connection with the treatment agent and mediates this information to it. The treatment agent in turn tries to get a treatment for these disorders and sends the results back to the coordination agent that will transfers it to the user interface agent that will display its contents through the WWW browser, see Fig. 3.

6. An example of utilization and testing

To bring out the advantages of employing multi-agent technology and expert systems, it was necessary to look at some real test cases from their conception to realization. We

have generated test cases that trigger mostly knowledge of the six diagnosis agents. For each case a brief description of its objective is given as well as its related input/output screens in diagnosis session and in treatment session. The function of coordination agent is shown through screens at run time. It is recommended that the end user of our system use a java enabled browser (e.g. internet explorer). Throughout this section we show screens taken from running our system on the MS Internet Explorer. When the user connects to the server site through any available web browser, the front page shown in Fig. 4 appears. This page initiates the user interface agent and carries the necessary information about the address and the port numbers of all expert system agents.

We have adopted a testing methodology based on manually generated test cases taken from the diagnosis and treatment documents of grapes crop. The objective is to cover all the agents, verify the proposed system by considering the number of expert systems agents that succeed in diagnosis: 0-agent succeed, 1-one agent succeed, and many agent succeed. The methodology also trace down the function of the coordination agent in each case. The main steps in our methodology are:

1. Walk through the diagnosis design document and manually generate random test cases. It is worth noting that the generated test cases are by no means exhaustive. The test case is generated by walking backward through the inference chain, starting from a target disorder(s) and randomly selecting values that conclude the selection reached at any given point in the inference chain.
2. For each disorder obtained from step 1, walk through the treatment design document and manually generate random test cases. This time the test cases are generated by walking through the inference chain to get the treatment operation(s).
3. Merge two or more test cases in order to test the behavior of the multi-agent environment. This will try to simulate real life situations, when it may happen that one or more disorders appear at the same time (this step is used for testing the possibilities of many-agent succeeded).
4. Run the generated test cases on the system.
5. Compare the results with the target disorder.

For time and space constraints, we will show a test case that tries to diagnose and treat disorders in case of three diagnosis agents respond.

6.1. Verify three diagnosis agents: nutrition deficiency, mite, and snail

Objective. The objective of test case is to conduct an experiment that demonstrates the capabilities of the system to diagnose and treat disorders in case of three diagnosis agents respond. In this case we choose disorders that belong to nutrition deficiency, mite, and snail expert agents. This test case covers the diagnosis and the treatment for the disorders due to iron deficiency, zinc deficiency, manganese deficiency, mite, and land snail disorders. The treatment of iron deficiency will be on date 26/05/2001, the material used is iron_chelate. The treatment of zinc deficiency will be on date 20/05/2001, the material used is zinc_sulphate. The treatment of manganese deficiency will be on date 23/05/2001, the material used is manganese_sulphate. The treatment of mite will be on date 01/06/2001, the material used is vertamic. The treatment of land snail will be on date 29/05/2001, the material used is iron_sulphate.

6.1.1. Diagnosis session

Diagnosis input:

Growth stage: vegetative
Leaves color: yellow
Leaves status: small, drop
Color of spots: brown
Position of leaves color: whole leaf
Type of infection: new leaves
Range of infection: adjacent spots
Depth of water: medium
Snail exist: yes

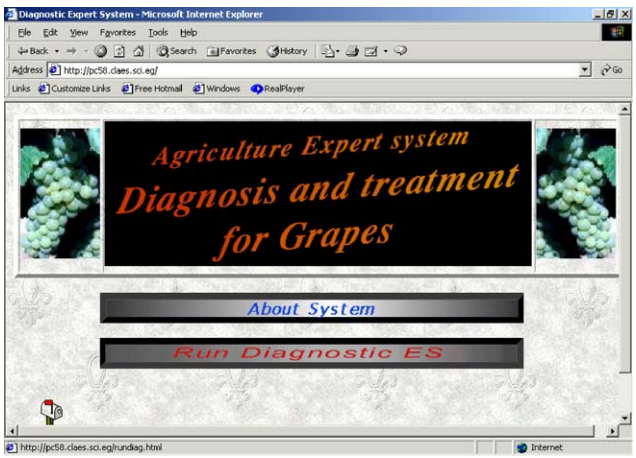

Fig. 4. Front page screen.

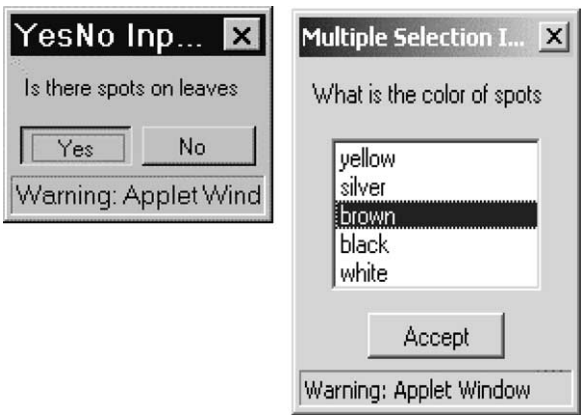

Fig. 5. An example of input screens in the diagnosis session.

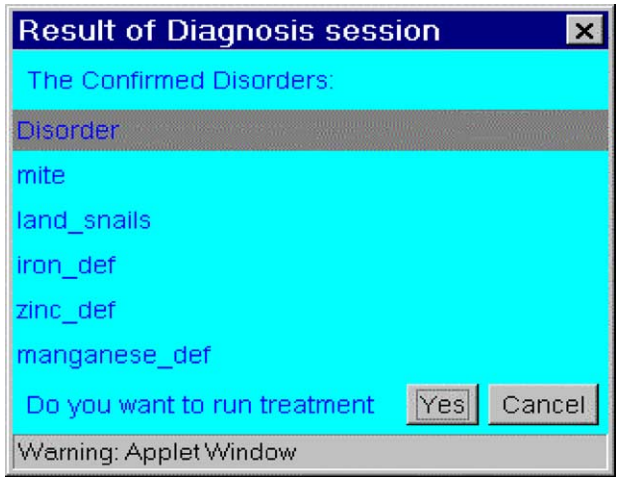

Fig. 6. Output in the diagnosis session.

An example of input screens is shown in Fig. 5.

After all the required symptoms are collected, the user interface automatically sends them to the coordination agent. Which in turn will establish the connection and broadcast the collected data to all the diagnosis agents. The answers from those agents are sent back to the user interface agent.

Diagnosis output: The output from the diagnosis agents of this case is shown in Fig. 6, which indicates that the three agents that are involved in the test case have found that the input symptoms match a certain disorder in their knowledge base.

The user interface asks the user whether or not it may call the treatment agent. Assume we continue with the treatment agent.

6.1.2. Treatment session

The user interface agent will call the communication model of treatment. Then the treatment communication model start asking about some information concerning treatment, such as the infection range, and the material used. After that the coordination agent starts the connection with the server and calls the treatment agent.

Treatment input: Zinc used material: zinc sulphate.

Treatment output: The result from treatment agent again returns back to the user interface agent through the coordination agent. The recommended treatment includes the date of the treat operation, the material used, the material quantity, the method if any, and the treatment advice, see Fig. 7. The right screen, is the detailed screen of the disorder shown on its left.

6.1.3. Coordination agent

Fig. 8 shows a trace of the flow of control of the output from coordination agent to the diagnosis agents, and the input from each agent at run time. Each agent located in a server with server name listen to a port number. The input shows that there are three agents answer with *reply* (at port 1095, 1070, 1075) and all the other three agents answer with *sorry*. The coordination agent will then send to the treatment agent.

7. Conclusions

In this paper we have described our experience with decomposing the diagnosis expert system into a multi-agent system. Our approach consists of a number of cooperating agents capable of solving the diagnosis and treatment problem in the agricultural domain. It has been applied to the grapes crop for providing services to a large group of users over the internet. The system can be geared towards any other related system or application. The widespread use of the internet and WWW provides an opportunity for developing ES that fit into the agent technology widely available.

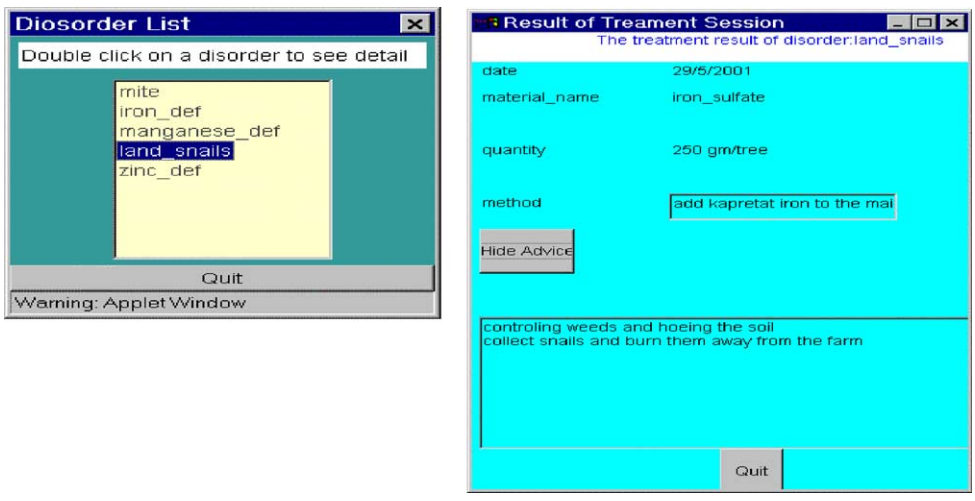

Fig. 7. An example of output screens in the treatment session.

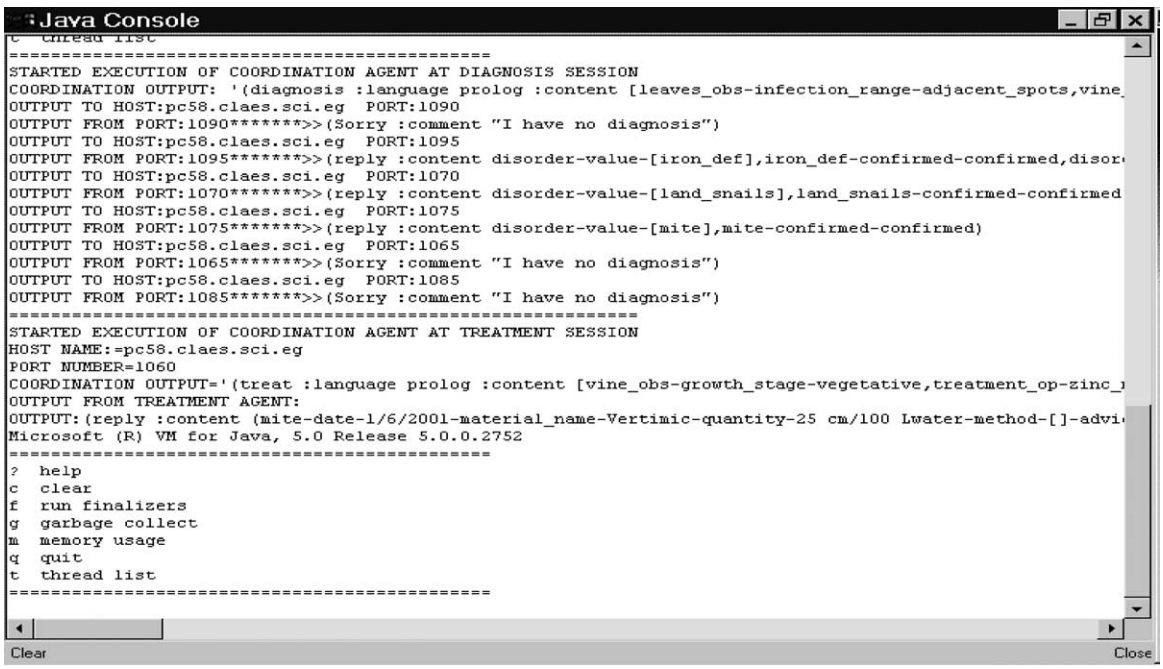

Fig. 8. Output/input to the coordination agent.

The new architecture concerning diagnosis has the following advantages:

- It tries to simulate what may happen in real world in a very fine-grained manner. Each agent corresponds to a domain expert specialization.
- Partitioning of domain knowledge makes it easy to maintain, easy to understand, and improve performance.
- The distribution of responsibilities among agents, achieves modularity and reduced complexity.
- Re-usability at the knowledge level was achieved since the proposed agents are capable of providing their services to other expert systems as well as to other applications provided that these other applications use the same ontology and understand the same communication language.

The expert systems agents are implemented in KROL using KADS knowledge engineering methodology. The user interface agent and the coordination agent are implemented in JAVA. Experiments on a set of test cases from real agricultural expert system were performed. The results observed in our experiments were satisfactory and proved that the system is robust. The system can diagnose and treat 26 disorders and the domain ontology consists of 66 concepts and more than 100 rules.

References

Chandrasekaran, B. (1988). Generic tasks as building blocks for knowledge-based system: the diagnosis and routine design examples. *The Knowledge Engineering Review*, *3*(3), 183–210.

Durfee, E. H., & Lesser, V. R (1988). Using partail global plans to coordinate distributed problem solvers. In A. H. Bond L. Gasser, & Kaufmann, (Eds.), *Readings in Distributed Artificial Intelligence* (pp. 285–293). San Mateo, CA.

Durfee, E. H., & Rosenschein, J. S (1994). Distributed problem solving and multi-agent systems: Comparisons and examples. In *Proceedings of the 13th international DAI workshop*.

El-Beltagy, S., Rafea, A., Kamel, A., Sticklen, J., Schulthess, U., & Ward, R (1995). An expert system for wheat disorders diagnosis and treatment using a hierarchical classification problem solver. *Second IFAC/IFIP/EurAgEng workshop on artificial intelligence in agriculture, Wageningen, Netherlands*. New York: Pergamon Press.

Engelmore, R. S., & Morgan, T. (Eds.), (1988). *Blackboard systems*. Reading, MA: Addison-Wesley.

Finin T., Fritzon R., Mckay D., & McEntire R (1994). KOML as an agent communication language. In *the proceeding of the third international conference on information and knowledge management*. New York: ACM Press. Available at http://www.cs.umbc.edu/KQML/papers.

Finin, T., Labrou, Y., & Mayfield, J. (1997). KQML as an agent communication language. In J. M. Bradshaw (Ed.), *Software agents* (pp. 291–316). Cambridge, MA: The MIT Press.

Genesereth, M. R., & Ketchpel, S. P. (1994). *Software Agents Communications of the ACM*, *37*(7), 48–53.

Hoog, R. de., Martil, R., Wielinga, B. J., Taylor, R., Bright, C., & Velde, V. D. W (1992). The CommonKADS Model Set. *Deliverable ESPRIT project P5248, KADS-II/WP I-II/RR/UvA/018/4.0*. Amsterdam: Universiteit van Amsterdam.

Huhns, M. N., & Stephens, L. (1999). Multiagent systems and societies of agents. In G. Weiss (Ed.), *Multiagent systems. A modern approach to distributed artificial intelligence* (pp. 79–120). Cambridge, MA: MIT Press.

Iglesias, C., Garijo, M., & Gonzalez, J. (1998). A survey of agent-oriented methodologies. In J. P. Müller, M. P. Singh, & A. S. Rao (Eds.), *Intelligent agents: Agents theories, architectures, and languages* (*Vol. 1555*). *Lecture Notes in Computer Science*, Berlin: Springer.

Jennings, N. R. (2000). On agent-based software engineering. *Artificial Intelligence*, *117*, 277–296.

Jennings, N. R., Varga, L. Z., Aarnts, R. P., Fuchs, J., & Skarek, P. (1993). Transforming standalone expert systems into a community of

cooperating agents. *Engineering Applications of Artificial Intelligence*, *6*(4), 317–331.

Jennings, N. R., & Wooldridge, M. (1995). Applying agent technology. *Applied Artificial Intelligence*, *9*(4), 357–369.

Jennings, N. R., & Wooldridge, M (1998). Applications of intelligent agents. Chapter 1. *Agent technology: foundations, applications, and markets.* Berlin: Springer.

Labrou Y., & Finin T (1997). *A proposal for a new KQML specification.* Online at http://www.cs.umbc.edu/kqml/papers/.

Moore, R., Dowding, J., Bratt, H., Gawron, J. M., Gorfu, Y., & Cheyer, A (1997). CommandTalk: A spoken-language interface for battlefield simulations. In *Proceedings of the fifth conference on applied natural language processing, Washington, DC* (pp. 1–7). Association for Computational Linguistics.

Nwana, H. S. (1996). Software agents: an overview. *The Knowledge Engineering Review*, *11*(3), 205–244.

Nwana, H. S., & Ndumu, D. T. (1999). A perspective on software agents research. *The Knowledge Engineering Review*, *14*(2), 1–18.

Rafea, A (1995). On integrating agricultural expert systems with databases and multimedia. In *Proceedings of the first international conference on multiple objective decision support systems for land, water, and environmental management: concepts, approaches, and applications, Honolulu, Hawaii, USA*.

Rafea, A. (1998). Agriculture. In J. Liebwitz (Ed.), *A chapter in the handbook of applied expert systems*. Boca Raton, FL: CRC Press.

Rafea, A., El-Azhari, S., & Hassan E (1995). Integrating multimedia with expert systems for crop production management. In *Proceeding of the second IFAC/IFIP/EnrAgEng workshop on artificial intelligence in agriculture, Netherlands*.

Rafea, A., El-Azhari, S., Ibrahim, I., Soliman, E., & Mahmoud, M (1995). Experience with the development and deployment of expert systems in agriculture. In *Proceeding of IAAI-95, Montreal, Canada*.

Rafea, A., Hassan, H., & Hazman, M. (2003). Automatic knowledge acquisition tool for irrigation and fertilization expert systems. *Expert System with Applications (ESWA): An International Journal*, (4), 49–57.

Rafea, A., & Mahmoud, M (2001). The evaluation and impact of NEPER wheat expert system. *Fourth international workshop on artificial intelligence in agriculture IFAC/CIGR, Budapest, Hungary*.

Rafea, A., & Salah, A. (1994). Guiding object-oriented design via the knowledge level architecture: the irrigated wheat testbed. *Mathematical and Computer Modeling*, *20*(8), 1–16.

Rafea, A., Warkentin, M., & Ruth, S (1991). An expert system for cucumber production in plastic tunnels. In *Proceeding of the world congress on expert systems, Florida, Orlando, USA* (pp. 909–916).

Rafea, A., Warkentin, M., & Ruth, S. (1992). Knowledge engineering: Creating expert systems for crop production management in Egypt. In C. Mann, & S. Ruth (Eds.), *Expert systems in developing countries: Practice and promise* (pp. 89–103). Boulder, CO: Westview Press.

Raphael, M. J., & Deloach, S. A (2000). A knowledge base for knowledge-based multiagent system construction. *National aerospace and electronics conference (NAECON), Dayton, OH*, October 10–12.

Shaalan, K., Rafea, M., & Rafea, A. (1998). KROL: a knowledge representation object language on top of prolog. *Expert System with Applications (ESWA): An International Journal*, *15*, 33–46.

Smith, R. G. (1980). The contract net protocol. *IEEE Transactions on Computers*, *C-29*(12).

Wooldridge, M., & Ciancarini, P. (2001). Agent-oriented software engineering: The state of the art. In P. Ciancarini, & M. Wooldridge (Eds.), *Agent-oriented software engineering*. *Lecture Notes in AI Volume*, Berlin: Springer.