12-1-1990

Integration of Distributed Expert Systems: An Open System Approach.

Semir Al-Schamma

Integration of Distributed Expert Systems:

An Open System Approach

A Thesis

Presented to the

Department of Mathematics and Computer Sciences

and the

Faculty of the Graduate College

University of Nebraska

In Partial Fulfillment

of the Requirements for the Degree

Master of Arts

University of Nebraska at Omaha

by

Semir Al-Schamma

December 1990

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 - 1346

THESIS ACCEPTANCE

Accepted for the faculty of the Graduate College, University of Nebraska, in partial fulfillment of the requirements for the degree of Master of arts, University of Nebraska at Omaha.

Committee

Name	Department
Stanley A. Wileman Jr.	MATHEMATICS/COMPUTER SCIENCE
[signature]	Decision Sciences
Zhengjun Chen	Mathematics/Computer Science

Zhengjun Chen

Chairman

Dec. 5, 1990

Date

To my Wife Liz

and

My Parents Agnes and Hassan

ABSTRACT

The integrated use of expert systems distributed on a network is a topic of practical importance. Through the proper integration methods, powerful expert systems could emerge. Several approaches exist for distributed problem solving, but most of them assume that the individual agents possess sufficient knowledge and skills to communicate and negotiate results and plans. In practice, however, what is needed is a simple, easy-to-implement, approach that allows the integrated use of a distributed (probably existing) set of expert systems or agents.

In this thesis the OSDES approach (which stands for Open System of Distributed Expert Systems) is presented. It entails the open systems perspective (originated from Hewitt), as well as the centralized version of multiagent planning. This approach sets a minimum requirement which specifies the types of interfaces that OSDES can handle. Almost any expert system can interface with OSDES through input and output redirection at the user interface level.

The heart of the integrated system is the Experts Directory Assistance (EDA) which is a service that all the agents in the system can utilize. The EDA keeps all the information about all the expert systems currently contributing to the system. Whenever an agent is added or removed, the EDA is notified to update its database.

Another major part in the integrated system is the communicator, which acts as the mediator between the individual agents, as well as between the agents and the EDA. Each agent in the system has a communicator associated with it. The communicator also provides

a user interface, and a simple scheduler to plan the execution sequence of the remote agents. The communicator incorporates a Generic User Interface (GUI), a Generic Agent Interface (GAI), a Distributed System Interface module (DSI), and a Kernel (or planner) module.

The OSDES approach was implemented on the IBM-PC/AT running MS-DOS 3.3. The communicator and the EDA were written using Microsoft 'C' compiler version 5.1, and 1st-Class (an expert systems building tool) was used to develop the sample expert systems. To implement the communication protocols, a Remote Procedure Call (RPC) development tool (Netwise RPC) was used. The underlying network is an Ethernet based Novell 386 Local Area Network (LAN), but given the proper RPC compiler and libraries, any other LAN can be used. Further improvement includes developing new agent interfaces using memory mailboxes or interactive remote input and output. It also includes eliminating the current DOS limitations by adapting OSDES in a multi-threaded environment such as UNIX or OS/2. environment.

Table of Contents

CHAPTER 1

Introduction

The integration of distributed expert systems is a topic that is still under research. Through the proper integration methods, powerful expert systems could emerge with an integrated knowledge base far beyond the individual knowledge of the individual experts.

Several attempts have been made to develop approaches to distributed problem solving, but most of these systems assume that the individual members (or agents) possess sufficient knowledge and skills to communicate, negotiate results, and plans with each other.

In reality though, expert systems that are being built today are not as powerful, and do not possess all these additional skills. These expert systems are developed using a generic high level language, such as 'C' or Pascal, but more commonly specialized artificial intelligence languages such LISP and PROLOG are used. Recently the approach has been to use Expert System Building Tools (ESBTs) such as 1st-Class, KEE, and Picon, which simplify the expert system building process. In any case, the expert systems generated are network unaware, not to mention other experts on the network, in other words communication skills do not exist in these systems. To integrate these systems using one of the existing approaches would

require major modifications (if not complete rewrites) of these systems, and this is not an acceptable solution. The available approaches are useful in cases where domain dependent expert systems are developed, so that the additional skills can be built-in at the development phase.

What is needed is a simple approach that allows the integration of a distributed, probably existing, set of expert system or agents. This approach should consider simple expert systems as well as complicated ones.

To be able to develop such an approach which investigates the existing distributed problem solving approaches, a literature survey is performed in chapter 2. The conclusion is that these works are either to complicated to implement, or place severe restrictions on the agents that can be integrated into the system. In chapter 3 the problem and the objectives of distributed problem solving are extended, and a new approach to the integration of distributed expert systems (OSDES) is presented. The actual implementation, environment, and tools used are discussed in chapter 4. Finally, the problem is reassessed and OSDES is compared with the other approaches in chapter 5. A list of future enhancements and development opportunities is also presented in this chapter.

CHAPTER 2

Distributed Problem Solving Approaches

This chapter will present the basic terminology used in this approach and in other approaches to distributed problem solving. Next some of these approaches are presented and their advantages and disadvantages are highlighted.

2.1 Expert Systems

An *expert system* is a system that has sufficient information to mimic the ability of an expert in a specified field. It uses a knowledge base supplemented by an inference engine to analyze given facts [Borl87]. It is a computer program containing knowledge and reasoning capability that imitates human experts problem solving in a particular domain. [Hu 89]

An expert system usually consists of a *knowledge base* which holds the expert knowledge in the domain of expertise and an *inference engine* which uses the knowledge base and information provided by the client to arrive at its conclusion. Finally, the *user interface* asks the questions, and converts answers into a form the inference engine can understand [Covi88, Luge89].

Early expert systems were built for specific domains with high level languages, these programs

were highly interwoven. "... although efficient, once these expert systems were constructed they could not be easily adopted to altered views of the domain". This led to the development of shells or *expert system building tools (ESBT)* [Gaag88]. "The idea behind an expert system building tool is that the user can produce a true expert system for whatever problem domain he wants by filling the shell with expert knowledge needed for his application" [Covi88, Cohe89].

2.2 The Distributed Environment

A system of distributed experts is one in which the expert systems are physically distributed. These systems are sometimes referred to as distributed problem solvers or multiple agents. Both distributed problem solvers and multiagent systems comprise the basis for that portion of artificial intelligence studies called "Distributed artificial intelligence (DAI)" [Gass89].

In *Distributed Artificial Intelligence* the emphasis is not on providing performance improvement, but on using the expertise of multiple agents to find a solution that fits the problem, or handles uncertainty through triangulation or models natural systems [Gass89].

Distributed Problem Solving (DPS) considers how the work of solving a particular problem can be divided among a number of modules that cooperate by dividing the problem, sharing the knowledge about it, and developing the solution [Gass89, Smit81, Stee86].

Multiagent systems comprise intelligent behavior among a collection of possibly preexisting, autonomous, intelligent agents, when they can coordinate their knowledge, goals, skills, and

plans to take action or solve problems. The agents in a multiagent system may be working toward a single global goal or separate individual but interacting goals.

The knowledge required for a distributed group of problem solvers differs from the one needed by a single agent, since the distributed agents need extra capability of planning, negotiation, and communication. While a single problem solver would detect any changes to it's world (knowledge and environment) almost instantaneously, the changes applied in a distributed problem solving environment may not be detected by everyone immediately [Stee86] (in fact some of the approaches in distributed problem solving, expect the agents to be able to anticipate and plan all the world changes).

2.3 Distributed Problem Solving: Objectives and Phases

The different approaches have several objectives [Gass89, Durf89, Stee86]. The major ones are listed below:

1. Boost task completion rate through parallelism.
2. Expand the set or scope of achievable tasks by sharing resources (including information, knowledge, and devices).
3. Increase certainty, reliability of solutions, and the probability of finding a solution, through redundancy.
4. Utilize DPS approaches in many AI problems that are inherently distributed e.g., interpretation of sensory information.
5. Build and maintain complex expert systems by using a structure of self contained modules.

6. Use a DPS system to simulate and analyze social sciences.

7. Accommodate open systems and multiple perspectives.

8. Allow agents to use global knowledge to make local decisions.

Smith and Davis identified the following phases in any distributed problem solving process [Smit81]:

1. Problem decomposition into subproblems.

2. Solution of kernel subproblems, which might require inter-communication between the agents.

3. Answer synthesis, which integrates the subproblems results. This phase is not always necessary.

2.4 Cooperative Distributed Problem Solving

Cooperative Distributed Problem Solving (CDPS) is a special case of distributed problem solving. It is the study of how a loosely coupled network of problem solvers can work together to solve problems that are beyond their individual capabilities. Each node can work independently and might possess various skills, thus each agent has different appropriateness to solve an assigned task. Each uses its expertise, resources, and information to solve subproblems, and then integrates the results with the others. Each has limited knowledge of the tasks assigned to the group and the intentions of the others [Stee86, Durf89].

CDPS differs from some of the DPS frameworks such as ACTOR [Jack86], HEARSAY II, ETHER language, BEINGS, CAOS, Poligon and connectionism where each agent is a

specialist in an aspect and there are predefined interactions between tightly coupled elements. In these systems each expert has no knowledge of the problem solving task or general strategies, and scheduling is centralized. An element in these networks is like a small piece of the brain, by itself it is not intelligent, but intelligence emerges from a well structured and tightly connected collection of such elements [Durf89].

Some of the cooperative distributed problem solving applications are:

1. Distributed interpretation such as distributed sensor networks. Agents constantly exchange their partial views [Durf89].
2. Distributed Planning Control such as distributed Air Traffic Control, cooperating robots, and remotely piloted vehicles [Durf89].
3. Cooperative expert systems such as navigation, and vehicle control. Multiple corporate expert systems negotiate to decide on product prices. Agents might have different goals, but common language. These are complicated systems in which agents must model each other.
4. Computer supported human cooperation such as intelligent command and control systems, multiuser project coordination, and meeting scheduler or messages router.
5. Cognitive models of cooperation such as emulating human methods of cooperation.

2.5 Constraints and Concerns

This section list some of the major issues of concern presented in other approaches [Gass89, Durf89, Stee86]:

1. How to decompose an individual problem, and allocate it to a collection of problem

solvers with different skills and resources? (sometimes called the *connection problem*). Solutions can include having a central node acting as clearing house, having nodes decompose tasks locally and announce them to the network, or having some nodes produce plans, while others evaluate and adopt them.

2. How to resolve subproblem interactions?

Nodes have to send copies of partial results to each other. Since the nodes' interactions are not predictable in terms of which nodes talk to each other; the partial results may be incompatible. To enforce compatibility the nodes must first recognize each other and second communicate with each other regarding the results.

3. How to exploit parallelism?

If a centralized scheduling mechanism is used, then a powerful scheduler is needed which knows everything in the system. However if decentralization is preferred, where each node has it's own scheduler, then the schedulers either have to cooperate, or the nodes must predict and anticipate each other's behaviors and actions.

4. Where and when to integrate the results?

The integration can be performed at one node or at several, it can be done with partial results, or with the complete final results, but how can partial results (arriving synchronously) with different certainties be integrated into meaningful wholes?

5. How to corroborate when nodes realize that they have formed mutually exclusive views?

They must reason about the information and decide which information to ignore. This type of decision will require additional intelligence.

6. How to keep all the agents up-to-date?

One approach suggests reflecting the changes made by an agent only to a subset of the agent, or only at specified time intervals, but inconsistencies may arise due to the incomplete reflections of the world changes to all the agents. On the other hand keeping every agent up-to-date with all actions and intentions of all the other agents is very difficult and can pose as a bottle neck to the communication channels. Another approach suggests that solutions must be coherent locally and globally, but this coherence must be achieved solely by local computation (i.e. nodes must predict the intentions and actions of others).

7. How to recover from a node crash?

None of the research done indicates how to handle crash and recovery, they suggest replication partially to resolve inconsistencies and uncertainties, and to assure that someone can respond in the case of node crash.

2.6 The Coordination Problem

Most of the considerations and issues mentioned above can be combined in a problem known as the *coordination problem*. It addresses how the agents should reason about the coordination process among them [Gass89]. In the remaining parts of this chapter, the coordination problem is explained and the different solutions to the problem are presented.

Effective coordination requires structure, flexibility, knowledge reasoning ability, and cooperation between the agents[Durf89]. This might be an impossible task in situations where global control, global consistent knowledge, or globally shared goals are impossible. [Gass89]

Node coordination approaches [Durf89, Gass89] include negotiation and contracting, a functionally accurate approach, organizational structuring, multiagent planning, open systems, sophisticated control, and blackboards.

2.6.1 Negotiation

Negotiation uses dialogue between the nodes to resolve inconsistent views and to reach an agreement on how they should effectively cooperate. This is "the process of improving agreement (reducing inconsistency and uncertainty) on common viewpoints or plans through the structured exchange of relevant information" [Durf89].

Negotiation is exercised in the Air Traffic Control (ATC) Problem [Stee86, Durf89] where several agents cooperate to direct aircrafts in ways to avoid collisions. The system is object-centered, an agent is associated with each aircraft, and each aircraft has only limited world knowledge, and is constantly gathering information about the other aircrafts. The major tasks of the agents are detection and resolution. They also provide some generic tasks e.g., sensing, input communication, output communication, initial plan generation, plan evaluation, plan fixing, and plan execution. An agent will request help if it can not resolve a conflict.

Goal interactions between the agents come in the form of shared aircraft conflict which happens, according to their current plans, if two (or more) aircrafts are going to meet in the future. When conflict is foreseen the agents must communicate and negotiate a solution. If (due to lack of knowledge) the group fails to assign the tasks to the most appropriate agents, the problem of optimal tasks assignment arises (also called the connection problem).

The coordination approach used here is called Task Centralization where conflicting planes choose a node to solve and resolve. A plane is then called to change direction and only it's local data is changed. Since it is centralized there is no inconsistent data. On the other hand it might be the bottle neck, in which case there is a reliability risk since there are two rounds of negotiations involved to decide who qualifies for planning and who qualifies for acting. This process could require time that is not available to the conflicting planes.

Even if the agents come to a conclusion, other coordination problems may arise. The action of moving to avoid a conflict might cause another one.

2.6.2 Contracting

"Contracting involves an exchange of information between interested parties, an evaluation of the information by each member from its own perspective, and final agreement by mutual selection" [Durf89]. Problem solvers use bidding, contracting and information exchange protocols to allocate work or resolve conflicts [Gass89]. The bidding protocol is the process of announcing the tasks by the manager and finding contractors.

An agent can be a contractor or a manager, nodes are free to act and unlike voting, agents are free to exit the process rather than being bound by the decision of the majority. The contract contains, task abstraction, eligibility specifications, and bid specifications [Durf89].

The Contract-Net-Protocol can be summarized as follows [Durf89].

1. The manager forms the task to be allocated.

2. The manager announces the existence of the task.

3. Available nodes evaluate the task announcement.

4. Suitable nodes bid for the task.

5. After the announcement of the task, the manager waits for *some time* then evaluates the bids.

6. The manager awards the contract to the most appropriate node(s).

7. The manager and selected contractor(s) communicate privately during the contract execution.

A basic messaging system is used in which the nodes can announce their availability and in special cases the manager can award a contract without bidding [Durf89]. The process of contracting can result in two or more results that need to be synthesized.

2.6.3 Functionally Accurate Cooperation

Functionally accurate cooperation involves the exchange of tentative results to overcome errors and coverage on problem solutions [Durf89]. Gasser [Gass89] defines it as the triangulation and convergence on useful results. It deals with the issue of getting nodes with inconsistent views to effectively cooperate.

Inconsistencies may arise from incomplete, out-of-date, contradictory, or conflicting knowledge, different views, and errors in hardware and software. They can be managed in several ways:

1. Do not allow inconsistencies, in other words precheck every modification to the system.

2. Resolve inconsistencies, if they occur, through explicit negotiation.

3. Build CDPS networks that perform accurately despite inconsistencies.

In a functionally accurate approach the network solution is structured so nodes cooperatively exchange and integrate partial results to construct consistent and complete results. "Some say that in complex systems it is impossible to guarantee that knowledge among nodes will remain consistent". [Gass89]

Unlike negotiation, functionally accurate cooperation is bottom-up. It relies on the nodes' ignorance which might lead to excessive communication. The problem in negotiation is that the nodes in the beginning don't know how their local subproblems fit within the overall problem.

2.6.4 Organizational Structuring

Organizational structuring is a compromise between negotiation (top-down) and functionally accurate cooperation (bottom-up). "An organizational structure of a CDPS network is the pattern of information and control relationships that exist between the nodes and the distribution of problem solving capabilities among the nodes" [Durf89].

Organizational structuring uses common knowledge about general problem solving roles and common patterns to reduce nodes' uncertainty. It provides control frameworks to work as a team. This approach allows nodes to work in parallel, assigns the roles to suitable nodes, permits overlapping roles to increase network reliability, and empowers each node to

determine for itself whether there is an interaction. Because the network might be able to solve problems in several different ways, it must have nodes that have the authority to decide and enforce a particular approach.

2.6.5 Sophisticated Local Control

Sophisticated local control involves reasoning about other agents in addition to reasoning about the actual problem, hence coordination is done locally. The information communicated has to satisfy relevance, timeliness, and completeness characteristics. The communication can be achieved by sending all the partial information all the time, sending when a node is locally completed, or sending the first information and the last [Durf89].

2.6.6 Open Systems

Gasser defines open systems as "systems with no complete representation and with dynamically changing boundaries" [Gass89]. An open system consists of a network of micro theories in which an agent or a small set of agents can reason logically and maintain consistent knowledge within a micro theory [Durf89]. Hewitt noted that "Internal operation organization and state of one computational agent may be unknown and unavailable to another agent" [Hewi86]. The open systems approach is further explained in the next chapter where a solution based on it is presented.

2.6.7 Multiagent Planning

Multiagent planning involves developing a plan on how agents should work, distribute it and follow it [Durf89]. A single agent or a group of agents may be used to form a coherent plan

for solving a multiagent problem. Dependencies and conflicts among the actions and knowledge of different agents are identified in advance. Communication and synchronization acts are inserted into each agent's plan to prevent conflicts when the plan is executed. In this approach one (or more) node possess(es) a plan that indicates exactly what actions and interactions each node will take for the duration of the network activity. This approach differs from contracting where nodes make pair wise agreement and there is no complete view of the network coordination represented. More computation and communication than other approaches may be required if multiagents are used [Gass89].

In Centralized multiagent planning [Degr87], plans of nodes are formed (announced), then a central node collects and analyzes them, its duties involve detecting and avoiding inconsistencies. Nodes form a plan for all their future actions and interactions.

2.6.8 Blackboards

A black board is a central global database for the communication of independent synchronous Knowledge Sources (KS) focusing on related aspects of a particular problem [Luge89]. Gasser defines blackboards as a collection of KSs which rely upon a global scheduler and a central shared data structure for communication, consistency and control e.g., Distributed Vehicle Monitoring Testbed (DVMT) [Gass89].

2.7 Summary

This chapter presents some of the approaches to distributed problem solving. Each approach is unique in its own way and has its application areas, but the must important observation is

that none of these approaches exists in an operational system. Durfee states that "It is important to remember that the implementations are prototypes and simulations; to date no CDPS networks have actually been used in real world applications." [Durf89]. The other major observation is that all these approaches require advanced intelligence to cooperate, but most of today's systems do not have this intelligence, and would have to be changed severely to be incorporated into a distributed system.

In the next chapter an approach will be presented that tries to solve these problems. It is based on open systems and multiagent planning approaches, and has a few similarities with the contracting approach.

CHAPTER 3

An Approach to the Integration of Distributed Expert Systems

As discussed in the previous chapter, there have been several attempts to develop distributed problem solving approaches, each claiming to be reliable, fast, and intelligent. As Durfee states, most of these systems are simulations or primitive prototypes. They are very complex and require that agents have additional expertise and communication skills. In addition most of these systems are domain dependent.

This is not very encouraging, given that expert systems have penetrated today's office environment [Will87], and could be potential agents in an integrated system. An approach must be developed that is domain independent, and that does not require any additional intelligence or skills by the agents to be part of the systems.

This chapter will first redefine the problem and set the objectives. Next an approach based on the open systems and multiagent planning approaches will be presented and the protocol described. Finally the approach will be compared to the other approaches described in chapter 2.

3.1 Redefining the Problem

To be able to present a solution or approach, the problem has to be further defined, and the objectives extended to meet the requirements of integrating expert system such as the ones developed in today's office environment.

The problem is the need for a simple approach that allows the interconnection of a distributed, probably existing, set of expert systems or agents. These expert systems may vary in size and complexity. A large percentage of today's expert systems are small and simple, most of them do not involve a major development effort since they are created using expert system building tools (ESBT), e.g., Du Pont's engineers developed over 300 such systems in 1988 using an ESBT called 1st-Class [Will87]. Thus a generic approach must be simple enough to encourage adding such systems to the network of distributed problem solvers. This ,rather unique, point of view distinguishes this solution from the others, since it covers a larger set of agents. Note that the solution presented here will be able to integrate small expert systems as well as large and complicated ones.

3.2 Integrated System Objectives

Based on the consideration described above, the objectives described earlier have to be extended to include the following:

1. Most of the expert systems are small programs, generally unaware of the existence of other experts or even the existence of a network. It is thus unlikely that these experts possess the advanced (or for that matter even primitive) communication and

negotiation expertise described in other solutions.

2. Furthermore it is unrealistic to expect the developers of such systems to spend more time developing a communication interface than they spent on the actual expert system. The developers need a generic and simple interface to the rest of the agents in the network.

3.3 The OSDES Approach: an Open System of Distributed Expert Systems

The approach used to integrate a distributed set of expert systems is a combination of an open systems approach and the multiagent planning approach. It also features some common attributes with the other approaches to distributed problem solving such as contracting.

The open system approach is chosen since it can accommodate preexisting systems (i.e. systems that were not built specifically for the integration process such as standalone systems). Hewitt assumes that the "internal operation, organization and state of one computational agent may be unknown and unavailable to another agent" [Hewi86]. This is the case with most the standalone systems. Multiagent planning similarly assumes preexisting agents. In addition the special case where only one agent is planning, called centralized multiagent planning, assumes that the one node develops the complete plan before having the group help in solving the problem.

In general an open system features the following [Bond88]:

1. They are composed of independently developed parts in continuous evolution.

2. They are concurrent and synchronous, and they have decentralized control based

on debate and negotiation.

3. They exhibit many local inconsistencies.

4. They consist of agents with bounded knowledge and bounded influence.

5. They have no fixed global boundaries visible to the agents constituting the system.

The OSDES approach presented here will satisfy these objectives as follows:

1. Any expert system can be added to the integrated system at any time, thus the system consists of independently developed parts and is in continuous evolution.

2. These expert systems are synchronous and concurrent, they are based on an approach similar to contracting. Control is decentralized in the sense that any agent can issue a problem to be solved, but that agent will be the center of coordination for that specific problem. Using this semi-centralized approach reduces the load on the communication channels, as well as the level of negotiation and communication intelligence required from every agent.

3. Since the experts do not modify each other's knowledge bases, they do exhibit local inconsistencies. These inconsistencies are resolved at the integration level (if it exists).

4. Every expert system is restricted to its own knowledge base, thus it has bounded knowledge; it uses its own inference engine and thus has bounded influence.

5. There is no global boundary for either the knowledge base or inference engine.

3.4 Agent to System Interaction

To integrate an agent in the system, the agent must be able to interact with the system. There are three possible levels of interaction.

At the Knowledge Base level, a global knowledge representation scheme can be developed which converts all the requests of the local knowledge base to the global scheme and vice versa [Howa89]. This involves modifications to the agent which contradicts with the original objective of not excessively modifying the agent.

At the Inference Engine level, the task implied in the user query can be distributed to various inference engines, using methodologies similar to those considered in distributed artificial intelligence and then the partial results synthesized [Durf89, Smit81]. However, this approach also requires modifications to the existing expert systems.

At the User Interface level, which can be viewed as the part issuing the consolidations to the inference engine. This part seems to be the easiest accessible part of the integrated system, as well as the easiest to modify (if necessary). Integration through the user interface means incorporating one or multiple expert systems and providing them with the input data as if they were provided by the user. Thus an expert system can be invoked and supplied with input data similar to that which the actual user would provide. The output that the expert system thinks is going to the user will be captured and redirected to the agent initiating the original query. This type of interfacing is sometimes called a non-intrusive type of interface since the ignorance of an agent to the world around it is preserved by capturing and redirecting the input and output.

Integration realized through the user interface will be referred as a conceptual integration, because it allows the user to envision the set of expert systems as a conceptual whole. The

term *conceptual integration* also implies the little amount of software development effort required by this approach, since basically no modifications to the inference engine and knowledge base for an existing agent is needed.

A side benefit from interfacing at the user level is that agents are not restricted to being expert systems, in fact, any agent that requires input from the user or provides output to the user (or both) can be integrated into this open system.

3.5 A General Overview

The heart of this system is the *Experts Directory Assistance (EDA)* which is a service that all the agents in the system can utilize. The EDA keeps all the information about all the expert systems currently contributing to the system. Whenever an agent is added or removed the EDA is notified to update its database.

The other major part in this system is the *Communicator*, which acts as the mediator between the individual agents, as well as between the agents and the EDA. Each agent in the system has a communicator associated with it (Figure 3.1). It also provides a user interface, and a simple scheduler. The communicator incorporates a Generic User Interface (GUI), a Generic Agent Interface (GAI), a Distributed System Interface module (DSI), and a Kernel module.

3.5.1 General User Interface (GUI)

The GUI module interacts with the user. It receives the requests for advice, asks the user all

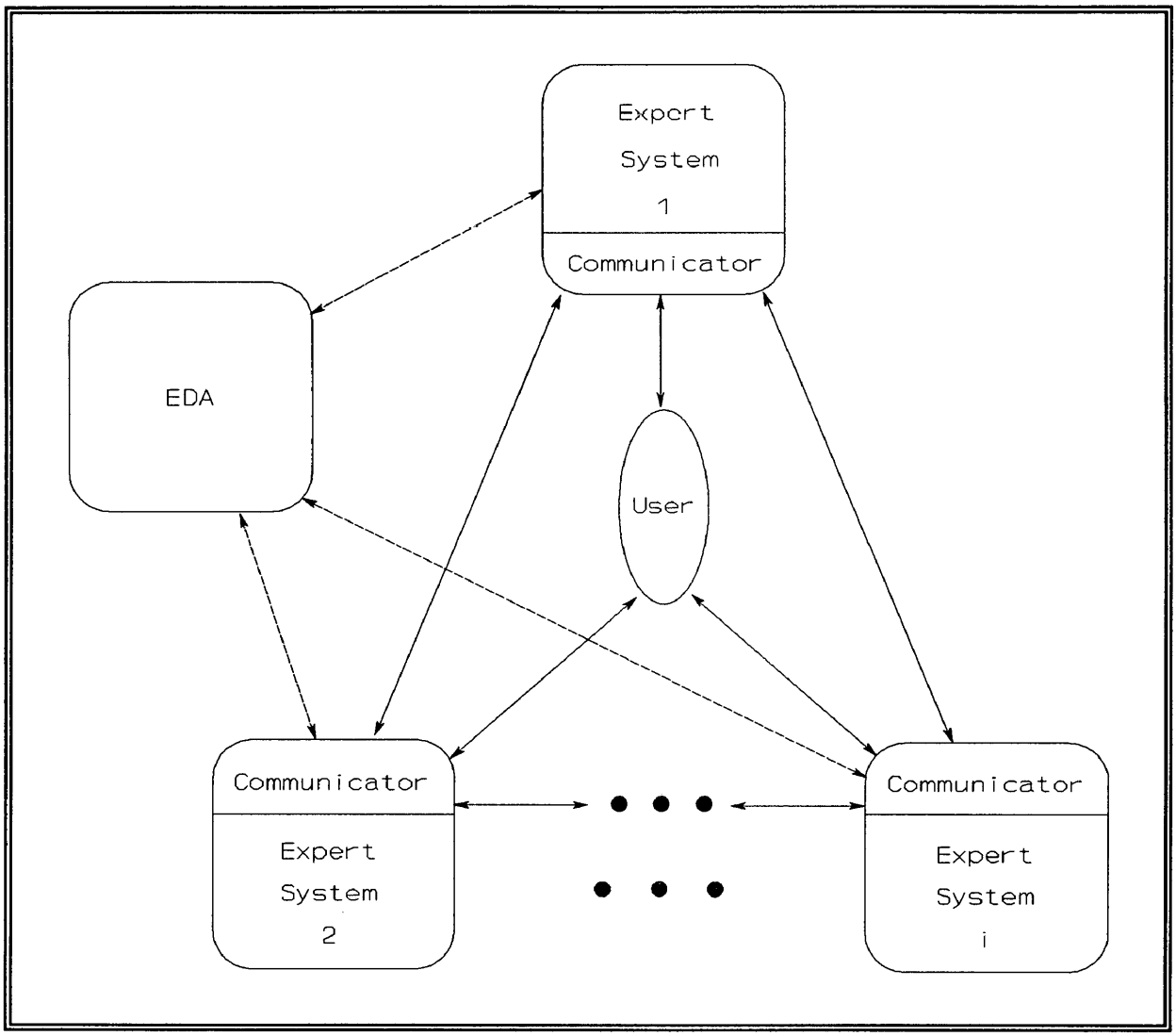

Figure 3.1 System Components Interaction

the necessary questions, and returns the final results to the user.

3.5.2 Generic Agent Interface Module (GAI)

The GAI wraps itself around the expert system. It is responsible for executing the local expert system, providing it with input, and obtaining the output. The conceptual integration method

described above is used.

The communicator can interface with the agent by simple input/output redirection, file passing [Degr87], mailboxes [Luge89], or interactive methods which require modifications to the expert's user interface. Using the I/O redirection or file passing methods, the agent is almost not changed, it still thinks it is running in standalone mode. The communication skills and responsibilities are shifted up to the communicator, thus the expert system will suffer minor to no modifications.

3.5.3 Distributed System Interface Module (DSI)

This module is responsible for the communication with the EDA, and the other communicators in the system. The DSI can receive a request to solve a problem or it can send a request to solve a problem.

3.5.4 Kernel/Planner Module

This is a simple scheduler that prepares the plan of execution (or a dependency list), then calls the agents in the order specified in the dependency list. In this document the kernel is sometimes referred to as the planner.

The knowledge possessed by the kernel includes knowledge about the existence of the EDA and the rules to apply in building a dependency list at the time when it communicates with remote communicators to solve a problem.

This is also the module called when the expert system is added to the system. It calls the EDA requesting to add an expert (add_expert), then it goes to sleep and waits for user input or a remote request.

3.6 Components Interaction in the OSDES Approach

In this section the two major components of this approach are detailed and their interaction is explained (Figure 3.2).

3.6.1 Experts Directory Assistance

In order for the communicators to communicate they have to possess certain knowledge about each other, such as, names, topics of expertise, and the facts other agents need to know before being able to provide a solution. Building this kind of knowledge into each communicator is very inconvenient and inefficient, it implies keeping all these knowledge bases up-to-date with every change to the integrated system.

A better option is to keep all this knowledge about all the experts in a centralized knowledge (or data) base, the Experts Directory Assistance (EDA). All the communicators are aware of the existence of the EDA, thus the knowledge requirements of each communicator is reduced. If a problem is to be solved by the integrated system, the communicator will retrieve from the EDA all the information it needs about all the agents in the system that deal with the specific problem topic. This information is updated every time an agent is added or removed from the system.

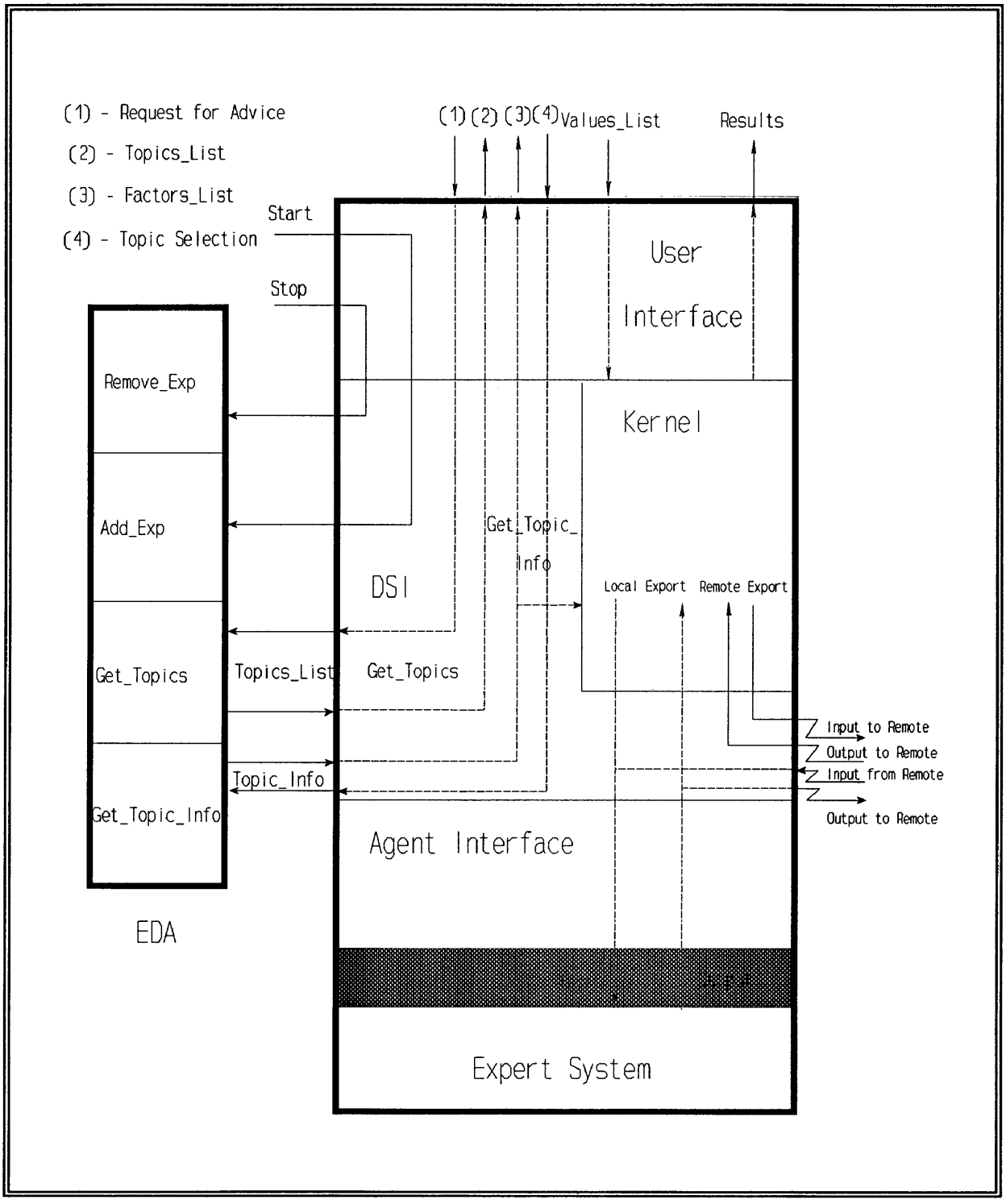

Figure 3.2 EDA/Communicator Interaction

For each expert or agent in the system there will be an entry in the EDA that contains information about it. This information includes name, topics of expertise, facts required to solve a problem and the associated question issued to the user and domains. Information about the results generated by an agent is also kept in the EDA. The contents of such an entry are provided to the EDA at the time when the agent is added to the system. At that time the communicator obtains this information from a parameter file (contents of this file and how it is provided to the communicator are described in Appendix A).

The EDA acts as a global directory assistance. Whenever an agent is added to the system an entry in the EDA is created for it. Similarly when an agent is removed (or crashes) its corresponding information is removed from the EDA. Since this a non-static data base, the system can grow and shrink dynamically, thus the system resembles an open system by being in a continuous evolution.

3.6.2 Communicator

This section will describe the two possible states for a communicator, Server or Client.

3.6.2.1 Server State: In this state the communicator is waiting for a remote request to execute the local expert system. The communicator is activated via its DSI module (Figure 3.3) which receives a remote request along with the necessary input data to the expert. The GAI is next activated which starts the local expert system and provides it with the input data. The output from the expert system is captured by the GAI and forwarded to the DSI, which in turn returns the output data to the communicator that issued the remote request.

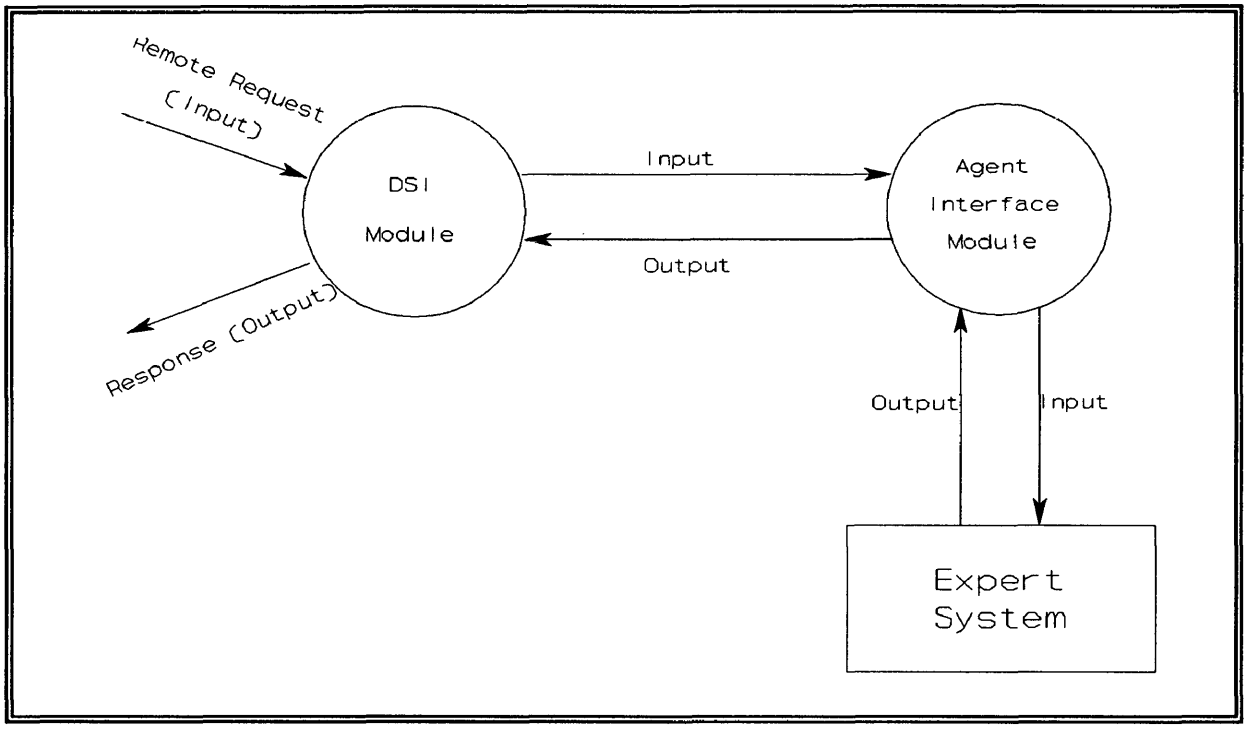

Figure 3.3 Communicator Server State

3.6.2.2 Client State: While in client state the communicator is passive, it does not generate any load on the network. The communicator is activated via a user request of advice to the GUI module(Figure 3.4.a). The request for advice is translated to a request by the DSI module to the EDA for a list of available topics (get_topics). The list of topics returned by the DSI module (topics_list) is presented to the user to make a selection for the topic of interest. Once the user has selected a topic, the DSI module is invoked with a request to get the detailed information about the selected topic. This request is translated into a (get_topic_info) to the EDA (Figure 3.4.b) The detailed information about a certain topic is the name(s) of the expert(s) that deal with this topic, along with the set of factors, questions, and domains needed to generate the result. Having received all the information the GUI module asks the user all the questions (facts), and forwards all the answers (values) to

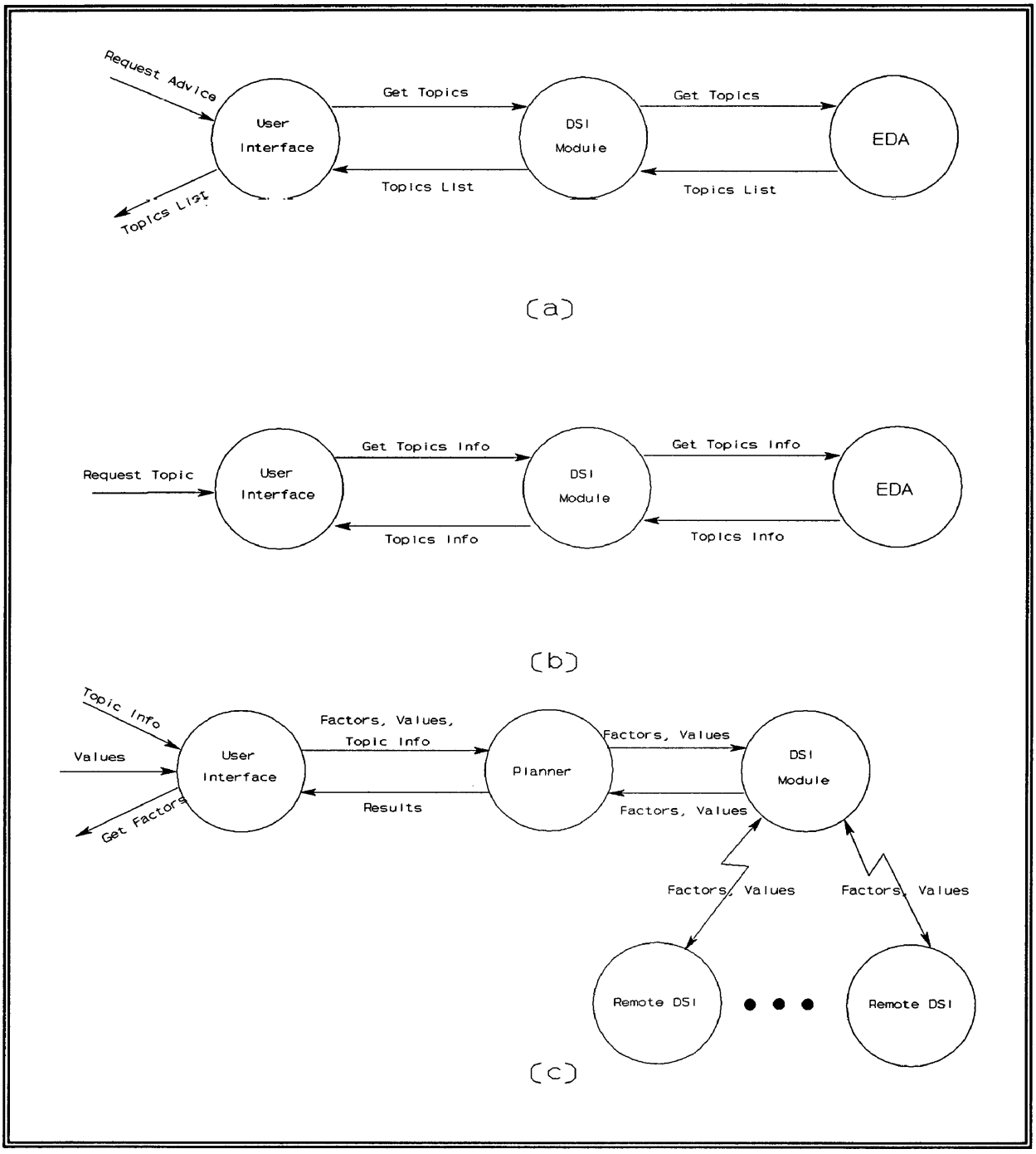

Figure 3.4 Communicator Client State

the kernel (Figure 3.4.c).

The kernel, which has already received a copy of the topic_info, generates a dependency list based on input and output factors of each agent (See the section Generating Dependency Lists). Next the kernel calls the remote agents in the order specified in the dependency list. The final result(s) is/are returned to the GUI to be displayed to the user.

3.7 Dependency Lists

In the client state the user provides all the answers (values) which are forwarded to the kernel along with a copy of the topic_info. From the topic_info the kernel can identify the agents involved in the problem solving process and also all the input and output factors to each agent. Based on this information the kernel can build a dependency list which will specify the order of calling the remote agents.

Building the dependency list is based on the following set of rules:

1. If an agent 'A' produces output that is required by an agent 'B' then 'A' must be called before 'B'. In the diagrams (Figure 5) this would be an arrow from 'A' to 'B' (Figure 3.5.a).
2. If 'A' and 'B' produce output that is required by 'C', then 'C' can not be called until 'A' and 'B' returned their results (Figure 3.5.b)
3. In a similar manner if 'A' and 'B' are independent and neither requires the output of the other one, then 'A' and 'B' can be called in parallel (Figure 3.5.c).
4. A dependency list can have a single input to one or more agents (Figure 3.5.b). It also could have multiple entry points (Figure 3.5.c).
5. A dependency list can have one output (Figures 3.5.a and 3.5.b), or multiple

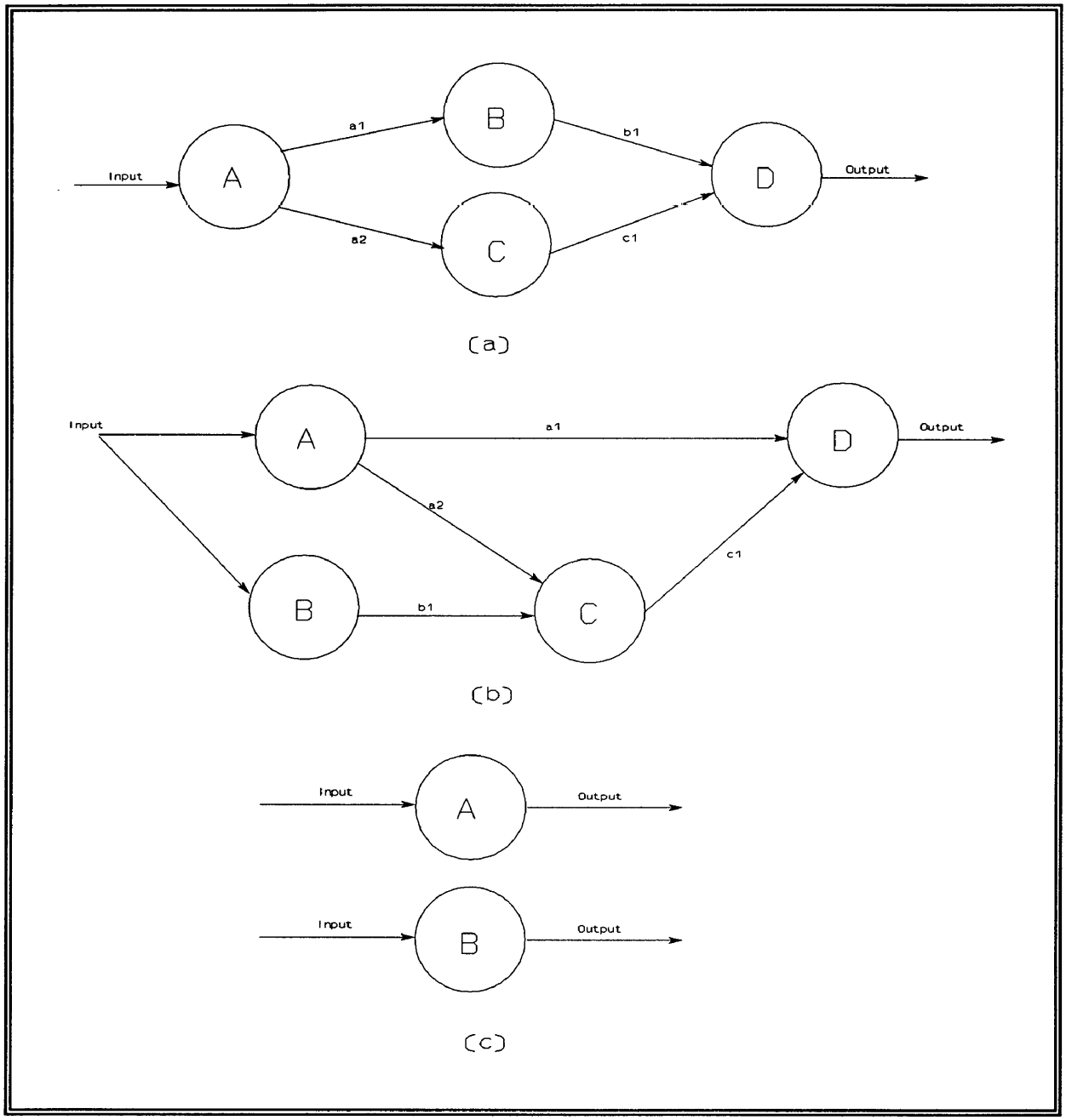

Figure 3.5 Dependency Lists

outputs (Figure 3.5.c). Results integration is exercised in the case where one result is produced. If there is no results integration agent in the dependency list, the user

will see several results (or advices).

3.8 Crash and Recovery

Crash and recovery are areas given inadequate consideration in other approaches. When a member of the integrated system crashes the system is left in an inconsistent state, since the EDA indicates that this expert exists when in fact it has crashed.

If a communicator sends a request to another communicator that has crashed, it will obviously not receive any response and it will time out. In that case the communicator will call the EDA and place a so called *service call.* There are two modes for a service call, regular and urgent.

In the regular mode, the communicator places the service call but does not wait for a response from the EDA. A regular mode service call is placed when according to the dependency list there is another expert that was called at the same time and that fulfills the requirements for calling the next expert in the list (i.e., it returns all the necessary results).

In the urgent mode the communicator has to wait for the EDA to perform the service call since it can not rely on other experts. If the EDA finds that the remote expert has left the network it updates its data base to reflect the new state of the network and return a negative result to the communicator indicating the crash of the remote expert. Upon a negative result the communicator will try to recover by asking the user the questions associated with the failed agent's results and forwarding them to the next agent. If the user is unable to answer

the questions the process is stopped and the user is informed of the temporary downfall of the system.

3.9 OSDES versus other Distributed Problem Solving Approaches

OSDES performs the same phases as the DPS in terms of problem decomposition, subproblem solution, and answer synthesis. Unlike the other approaches the decomposition process does not require any negotiation and little intelligence to build the dependency list. Parallelism is achieved by concurrently calling those experts that do not depend on each other. The answer synthesis is left to the integrated system, if an agent uses the results of two or more agents and operates on them then it is called. In other words if the last stage of the dependency list consists of one node then this node performs the answer synthesis, otherwise a user might be presented with several results. The idea is that building the answer synthesis intelligence into the communicators, which has to be generic, will be complicated and not very efficient. It is more efficient to build that knowledge in separate agents that deal with a specific domain.

The OSDES approach is similar to the multi agent approach, since both prepare the plans before hand. OSDES builds a dependency list which represents the plan. OSDES is a special case of multi agent planing where the plans are developed by one node.

Unlike the original open systems approach all the nodes do not have to reason about inconsistencies, this skill is built into the nodes that integrate the results (if they exist).

The OSDES approach is similar to the contracting approach, where there are managers and contractors, except that in OSDES the contractors announce their availability first thus eliminating the need for the bidding process. The manager (coordinating communicator) gets only the bids from the EDA that relate to the topic of interest. By eliminating the need for broadcasting a bid the load on the network is reduced. In addition the manager in OSDES does not have to wait until all the contractors have responded to it's bid. Finally, the contracting approach requires domain specific knowledge in the manager to be able to decompose a problem and evaluate the bids.

3.10 Summary

This chapter presented an alternative approach to the integration of distributed expert systems. This approach tries to accommodate the current generation of unsophisticated expert systems developed in office environments. Unlike the other approaches OSDES does not make any assumption about the different members in the system. It does not require any extraordinary intelligence or communication skills from the individual agents. Instead it sets a minimum requirement which specifies the types of interfaces that OSDES can handle. Almost any program can interface with input/output redirection, file passing, or interactive I/O.

While the other approaches had assumed that the agents in the system should possess knowledge about each other, OSDES shifted this responsibility to the communicator and the experts directory assistance. Finally, this approach produces less traffic on the network due to minimum knowledge exchange needs.

CHAPTER 4

Implementation

4.1 Program Development Environment

The OSDES approach was implemented on the IBM-PC/AT running MS-DOS 3.3. The communicator was written in Microsoft 'C' compiler version 5.1, and 1st-Class is an Expert Systems Building Tool (ESBT) used to develop the sample expert systems. To implement the communication protocols a Remote Procedure Call (RPC) development tool called Netwise RPC was used. The underlying network is an Ethernet based Novell 386 Local Area Network (LAN), although given the proper RPC compiler any other LAN could have been used (discussion will follow).

4.1.1 IBM-PC/AT

The IBM-PC/AT running MS-DOS was chosen as the development and implementation platform due to it's availability and the availability of the other system components for the PC platform (e.g., 1st-Class, Netwise RPC, MS 'C' 5.1, Novell).

4.1.2 Microsoft 'C' 5.1 Compiler

C was chosen, since it provided an interface to the system, the network, and the RPC compiler. The RPC compiler usually generates 'C' routines, that get compiled using the MS 'C' compiler.

4.1.3 1st-Class

An Expert Systems Building Tool (ESBT) makes it possible to build an expert system in a order of magnitude less time than it is possible in regular AI languages (e.g., LISP and PROLOG) [Geva87]. Since the purpose of this study was to integrate expert systems and not to develop them, it did not matter whether the experts were developed in PROLOG or 1st-Class. It saved a lot of effort in developing the actual expert systems, and it demonstrated that integrating expert systems developed using an ESBT is just as easy as integrating ones that were developed using conventional languages.

There exists a large variety of commercial ESBTs in the market including: ART, KEE, Knowledge Kraft, Picon, S.1, Personal Consultant+, EXSYS, Expert Edge, ESP Advisor, Insight 2+, TIMM, OPS5, Rulemaster, and 1st-Class [Geva87]. 1st-Class was the tool of choice at Du Pont [Will87] due to its simplicity and wide acceptance.

1st Class is an induction system that generates decision trees, which are elaborate rules, from examples given in spreadsheet form. Rules can be individually built or edited in graphical form on the screen. The rules are compiled, thus are very fast. The 1st-Class system is designed to interface readily with other software. Interfacing techniques include file passing, memory mailboxes, or interactive input and output. The examples developed in 1st-Class use the file passing method to interact with the communicator. The file format for these examples will be discussed later in this chapter (See Appendix A.,OSDES Parameter Files).

4.1.4 Netwise RPC

In the conventional programming environment, a procedure call involves passing arguments to, and returning values from, a procedure within a program. The RPC model suggest that this process can be carried out between two machines on a network. Conceptually, a Remote Procedure Call differs from a conventional local call only in that the caller and the called procedure occupy disjoint address space. In practical terms the caller and the called procedure are separated by one or more networks.

The Remote Procedure Call model is a client-server model in which the caller (client) passes arguments and control to the called procedures (server). However unlike regular local calls communication software intervenes. The client communication software manages the transfer of the request and any associated data (arguments) to a remote computer where similar software receives the information and simulates a local call to the intended procedure. When this procedure finishes its work, the server communication software returns control and any output data to the caller. By declaring a procedure as remote, the RPC compiler generate the necessary code for the server, the server dispatcher, and the client and server stubs for that procedure.

In the specific case of OSDES, each communicator is a server that waits for remote requests to execute the local expert system program. The Experts Directory Assistance (EDA) is also a server program that allows remote calls to add, remove, and retrieve records from the directory.

Due to some DOS and RPC for DOS limitations, only one communicator with one expert

system can run on a machine at a time. This limitation disappears with operating systems that allow threaded execution such as OS/2 and UNIX where each communicator runs as a separate process.

4.1.5 Novell 386

Novell Netware is a file server based Local Area Network (LAN) Operating System (OS). It allows services such as file and printer sharing.

Applications can interface directly with Netware via its Sequenced Transport Protocol (SPX). A set of library function call allow a 'C' program to interface with the network. RPC uses this facility to send and receive the packets across the network. RPC can utilize similar interfaces provided by other network operating systems such as NetBIOS for IBM networks, and TCP/IP for UNIX based networks. This means that porting the OSDES from a Novell network (using SPX) to a UNIX based network (using TCP/IP) is as simple as re-compiling and re-linking the client and server code with a different set of RPC network libraries.

Usually RPC relies on the file server to keep information about all the available 'RPC servers' or remote functions in the network. In OSDES the need for the file server was eliminated by using the EDA for that purpose. Each client knows the network address of the EDA which in turn can provide the address of any expert system on the network.

4.2 The Experts Directory Assistance (EDA) Setup

Before adding any expert system to OSDES the Experts Directory Assistance (EDA) must

be started. This is a server program that runs on separate PC (or in UNIX and OS/2, a separate process). Once activated it goes into server mode and waits for remote requests. Requests can be one of the following :

Adding an Expert System.

The EDA program checks the topic of expertise then decides whether to merge this expert with an existing group, or to create a new one.

Removing an Expert System.

The EDA program finds the expert system (by its name) and removes it from the list of experts contributing to OSDES.

Finding available Topics of Expertise.

Whenever a user requests assistance (on a communicator), the list of currently available topics is retrieved from the EDA and displayed to the user to choose from.

Retrieving Group Information about a Topic

Given a topic name the EDA program finds and returns information about the group of expert systems that claim to know about this topic.

4.3 The Integration Procedure

Adding a new member (expert system) to OSDES is very simple. The communicator, which is a generic shell, accepts a parameter file that contains all the information about the new

member. Given this information a communicator can pass information to the expert system, execute the expert system, and capture information from the expert system. The communicator interacts with the local expert system by passing 'facts' and their 'values' back and forth. In most of the cases no changes are required to the actual expert system.

The communicator is started from the command line with an argument indicating the name of the expert system that needs to be added to the system (e.g., SHIPPER is the name of the expert system which provides package shipping advice). The communicator looks for a parameter file with the same name as the expert and a '.DAT' extension (e.g., SHIPPER.DAT). It then reads the parameter file, and performs a remote procedure call "ADD to EDA" to add this information to EDA (Note that the communicator is acting as a client at this stage). Next the communicator switches into server mode and wait for remote requests form other communicators to execute the local expert system.

When a communicator is awakened by a remote request, it is usually supplied with all the factors and their respective values that the local expert system needs to solve a problem. The communicator responds by acknowledging the receipt of the request and data (done internally by RPC), and then executes the local expert system passing all the data it received. Once the execution is completed control is returned to the communicator with the results in a result file. The communicator reads the data from this file, returns it to the requestor, and goes back to sleep.

When a communicator is approached by the user, it calls the EDA and gets the latest list of

available topics. The user chooses a topic, and the communicator calls the EDA again to get specific information about the group of experts providing expertise in this domain. This information includes the names of all of the experts in the group, and all the questions that must be answered by the user before invoking the experts.

After obtaining all the answers from the user, the communicator prepares to call all the experts involved requesting their assistance. The communicator starts by calling the expert system whose input data (factors) does not depend on any output data from any other expert (Note that several experts could be called at this stage). The resulting factors from this stage determine which is to be executed next. At any stage if more than one communicator is to be called, an asynchronous RPC method is used where all the experts are called first and then the group is polled for the results. Finally the user is shown the result(s) returned by the system.

Result integration and synthesis is not the responsibility of OSDES, but that of the experts in the network. For every group of experts a new expert should be created that combines all the results. This does not apply to single expert groups, or groups where the last stage of calling involves only one expert.

When a communicator terminates it calls the EDA and requests the removal of the information about the local expert system.

4.4 A Note on the Expert Systems Requirements

To be able to interface the communicator with any expert system, the expert systems have to satisfy the following requirements :

1. The expert system must be able to interface with one of the interfaces described in appendix A (or the communicator has to be modified).

2. If the expert system allows file passing, then the developer might be asked to add one or two commands to the expert system code that read or write the file and instantiates the variable. For the example in 1st-Class the following command had to be added in the beginning of the expert system program: {READ filename ALL} which reads all the values from a file named 'filename'.

3. If the Input Redirection Method is used then the expert system must be able to ask the questions in a fixed order. Usually an option in the expert system development tool allows building the expert system to run in an exhaustive mode (i.e ask all the questions first). This is because all the answers are passed at once into the expert system, hence the order of questions can not be anticipated at the time of the file creation.

4. The expert system should be able to provide a plain text output report, or output redirection.

4.5 A Sample System and Scenario

In this section a sample system will be described and a scenario presented. This setup was used to test the system. First the actual components are described, then the scenario is presented.

4.5.1 Integrated System Setup

This system includes the EDA, and a communicator (Client Communicator) which will serve as the user interface to the system. There is no expert system associated with this communicator due to the DOS/RPC limitation where the communicator can run only in one state (Client or Server).

The other components of this system are the actual expert systems and the communicators associated with them. The expert systems are:

1. SHIPPER, a shipping advisor provides advice about which shipper to choose for a given type of service. The factors in this system are WEIGHT specifying the parcel weight, SPEED indicating number of days to delivery, 10:30_AM specifying AM delivery, SAT_DELV indicating if saturday delivery is required. Accordingly the expert produces a result factor SHIP_BY which is the shipping method.
2. SHIPCOST, a shipping advisor that provides the cost of shipping given the shipping method. Input factors are the same as for SHIPPER and the resulting factor is COST.

3. CREDIT, a credit advisor that can decide the person's credit status. Input factors are ANNUAL_INCOME, YEARS_EMPLOYED, and LIVING_STD which is urban or suburban. The output is a CREDIT_STATUS.

Obviously the first two expert systems belong to a common pool of experts since they both deal with the shipping topic. They can be called in parallel since there are no dependencies between them.

4.5.2 A Sample Scenario

This scenario assumes that the user will request an advice regarding shipping a parcel. It is divided into five phase beginning with the system startup and ending when the user receives the advice.

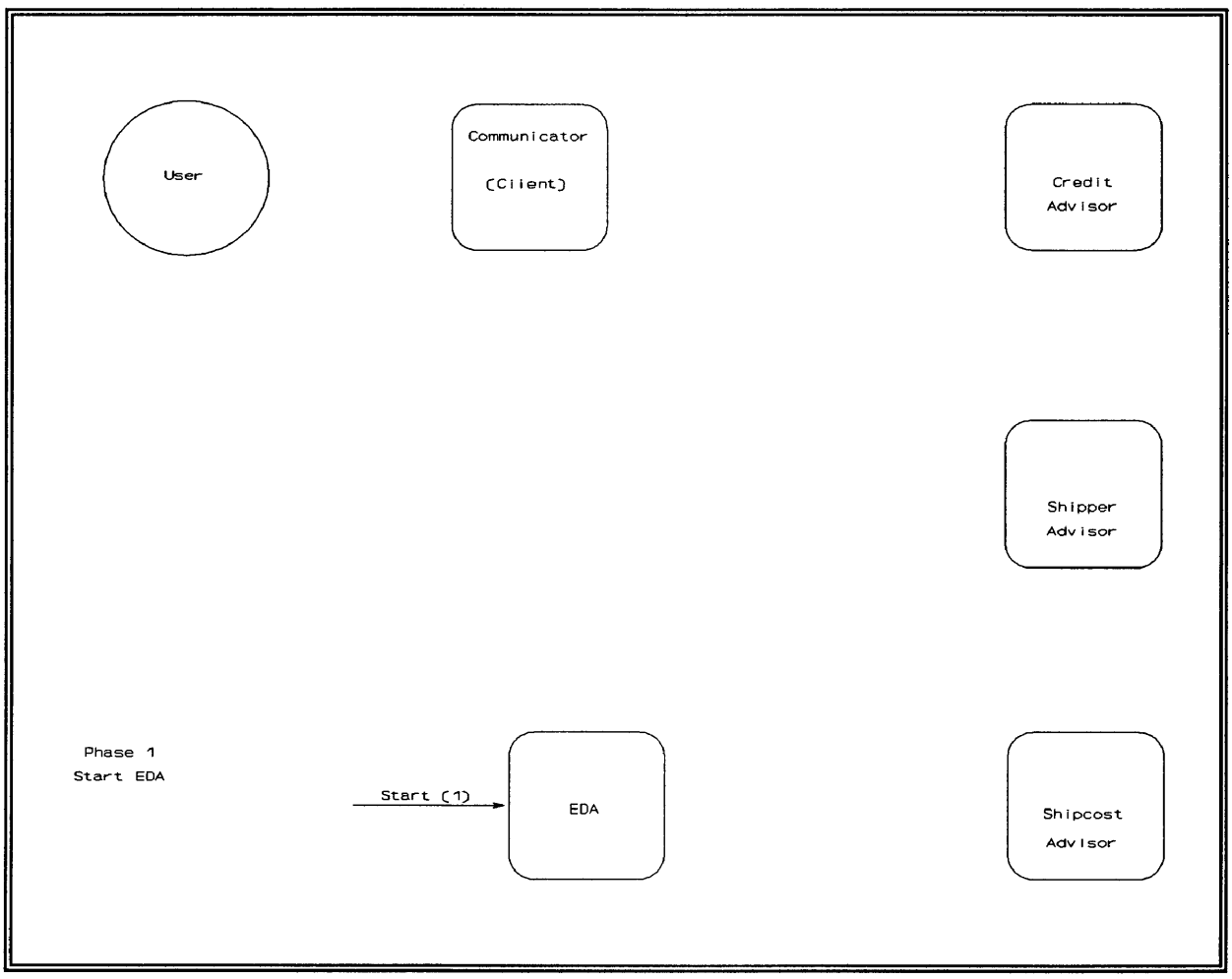

Figure 4.1 Phase 1 : Start EDA

Phase 1 : The EDA is started by running the EDA from the DOS prompt at the machine holding the EDA program (Figure 4.1). Once started the EDA goes to sleep awaiting a

remote request from one of the communicators. Note that the only communicator at this stage is the client communicator which has not been started yet.

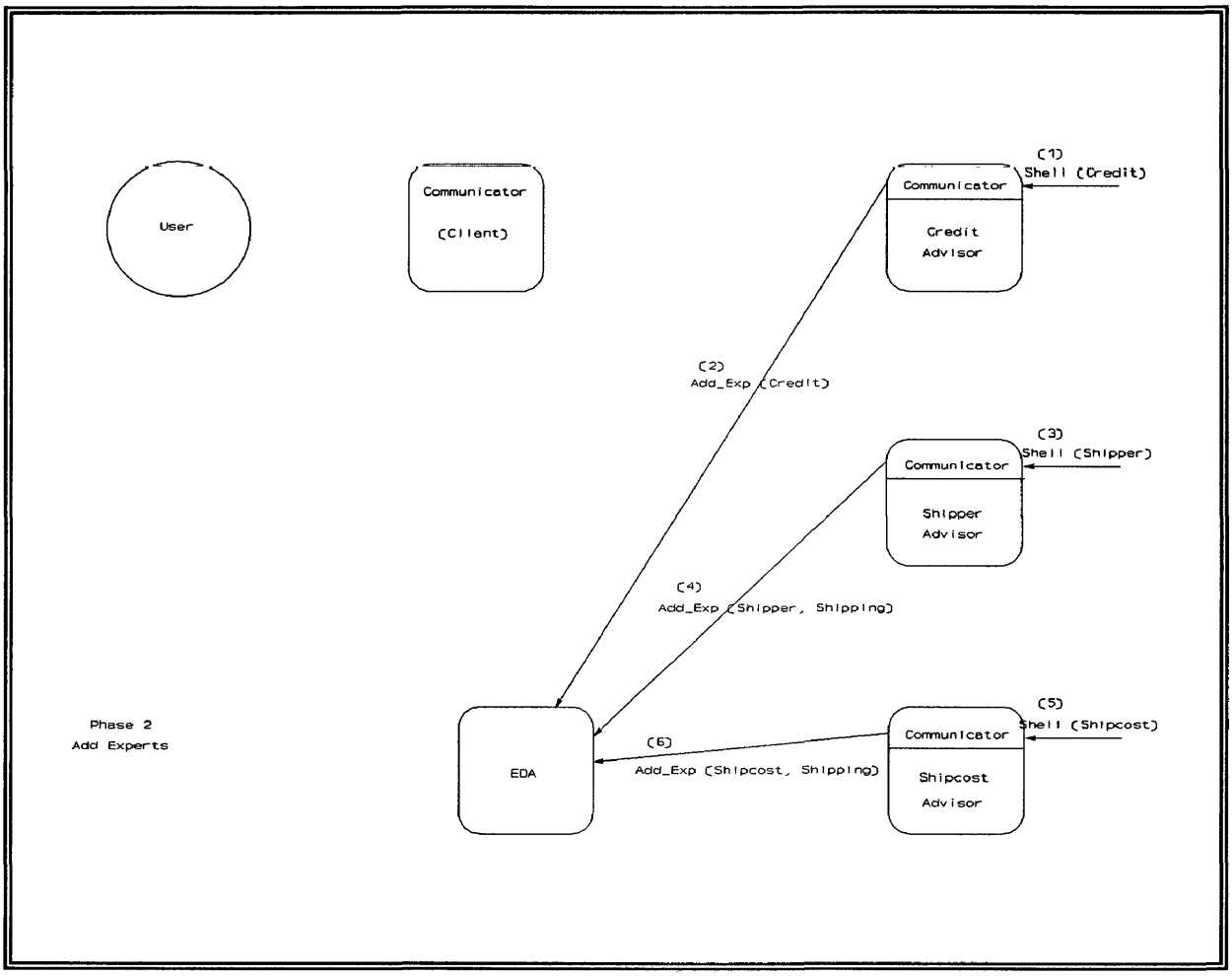

Figure 4.2 Phase 2 : Add Experts

Phase 2 : The individual expert systems can be added now to the system by loading the communicator on each machine and supplying it with the local expert name that it should represent. This is achieved by running SHELL shipper, SHELL shipcost, and SHELL Credit at the respective machines. The SHELL is the program which executes the remote call add_Exp when the communicator is started and, then starts the communicator in server mode. Each communicator (SHELL) will load the parameter file describing the local expert system,

and forwards it to the EDA with an Add_Exp request (Figure 4.2).

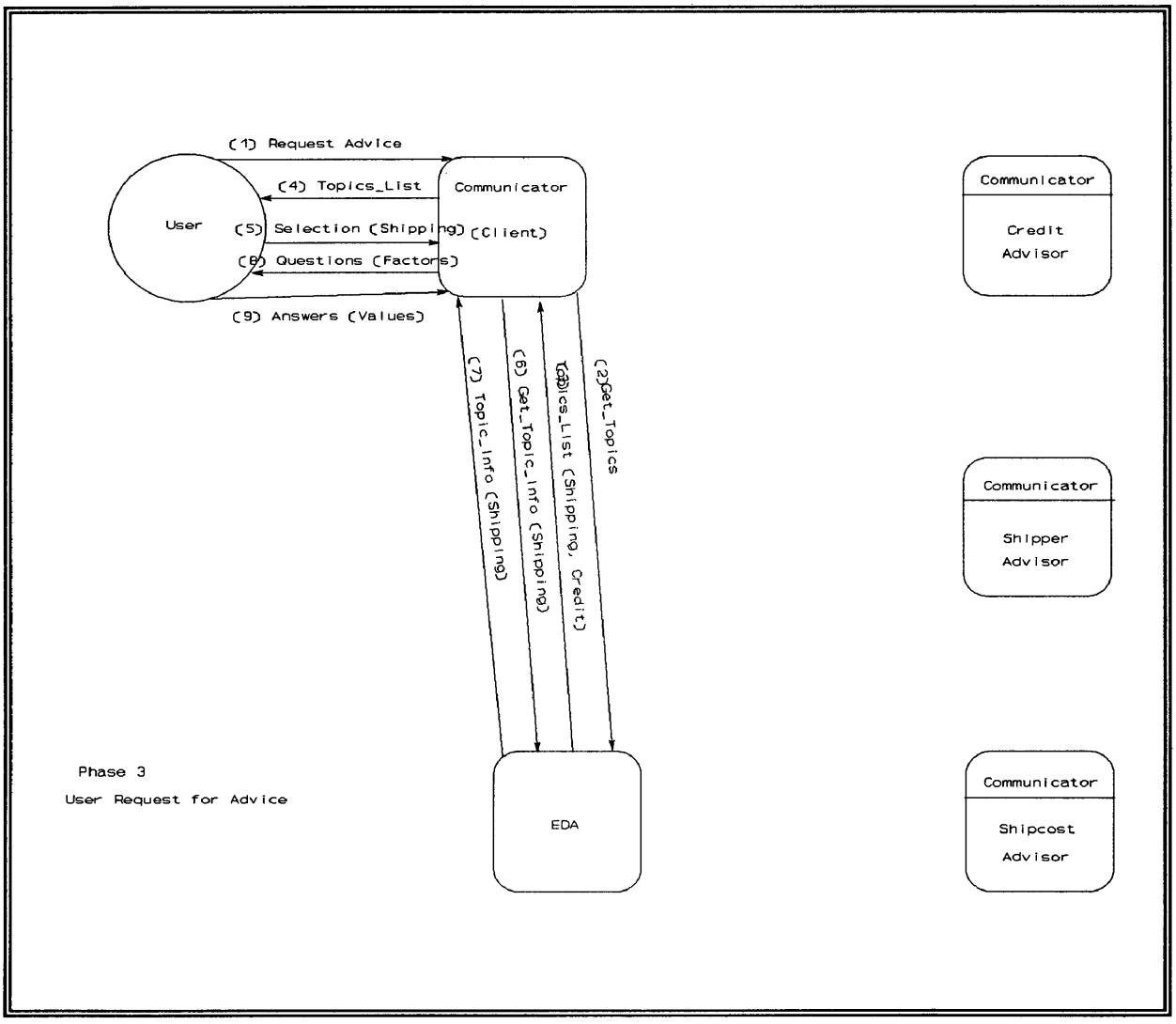

Figure 4.3 Phase 3 : User Request for Advice

Phase 3 : Next the user can request advice from the Client communicator by running CLIENT. The communicator immediately, contacts the EDA with a Get_Topics request (Figure 4.3). The returned Topics_List, which is Shipping and Credit, is presented to the user to make a selection. The user will select Shipping causing the communicator to call the EDA

again with a request for detailed topic information Get_Topic_Info about the topic Shipping. The EDA will return SHIPPER and SHIPCOST as the contributing experts names, along with list of factors used in the two systems, and the questions to asks and the expected results. For example for the factor WEIGHT, the associated question will be 'Specify the weight of the Parcel in Pounds (.5_TO_2, 2_TO_20, and OVER_20) ?', and the expected results would be one of the items listed between the parentheses. These questions will be presented to the user, and the responses (values) validated against the expected ones.

Phase 4 : Now that the Client communicator has acquired all the values from the user, it will call the experts systems according to their dependencies. In this case there are no dependencies between the two, so it calls (Try_Solve) in both remote experts at the same time (Figure 4.4).

Phase 5 : After calling all the remote experts, the communicator will start pooling them waiting for the results. The results will be returned in the form of factors and their values along with the text advice that the user would see. Since there are no more experts to call the communicator decides to return the results to the user and will display both text results (Figure 4.5)

4.6 Summary

In this chapter the actual implementation, environment and tools are described. The system runs on the IBM-PC/AT running DOS 3.3, on a Novell 386 network. The expert systems are developed in 1st-Class but the communicators are generic enough to interface with expert

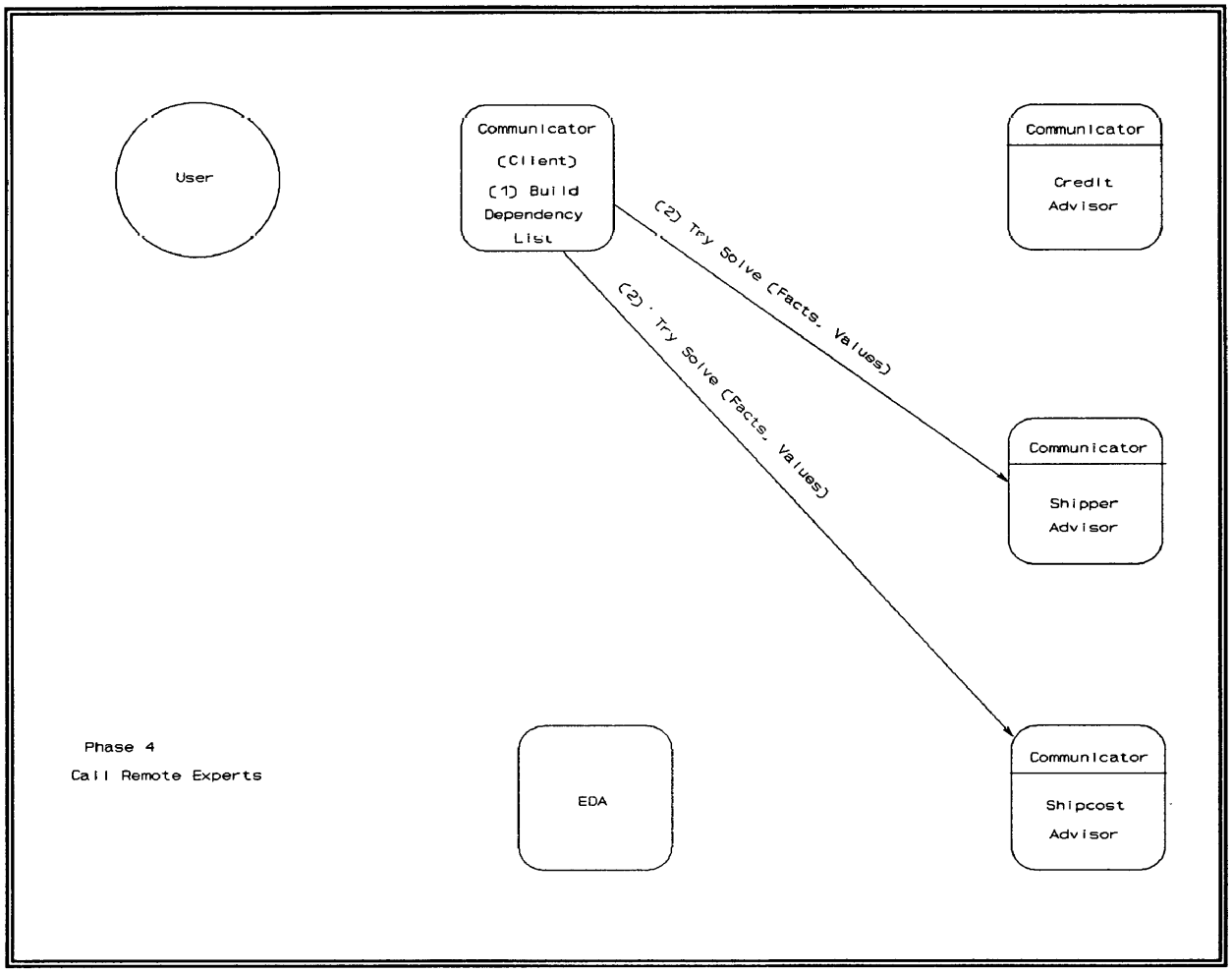

Figure 4.4 Phase 4 : Call Remote Experts

systems written in PROLOG, LISP or any other language. In fact any program that accepts standard input and output direction can be added to this system. The communication interface is based on a Remote Procedure Calls library called Netwise RPC. Using RPC the underlying Novell network can be replaced by almost any other network for which there are RPC libraries, such as Banyan VINES, 3COM, PC-NFS and DECNet.

The actual implementation of the EDA is described followed by a discussion on how to setup

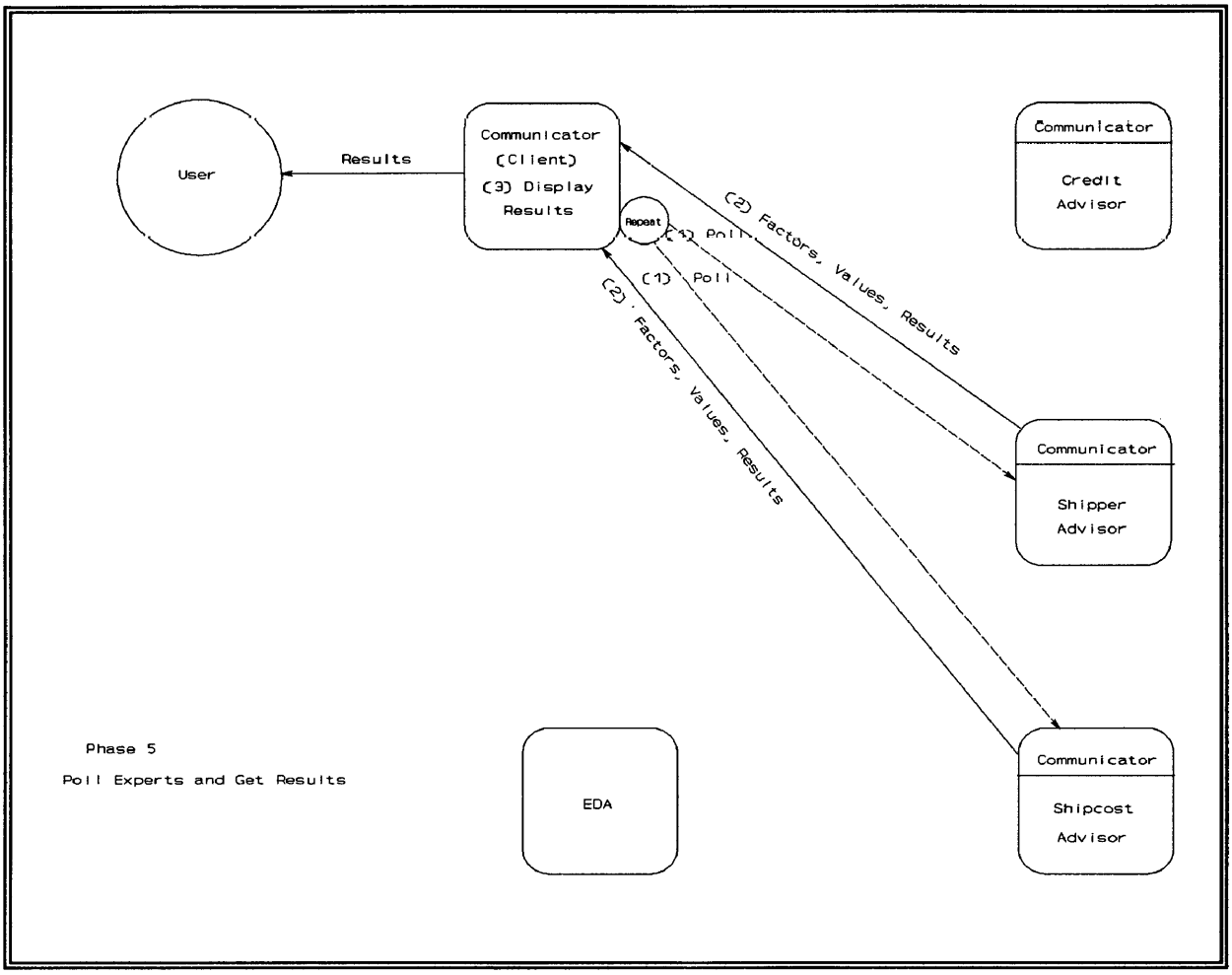

Figure 4.5 Phase 5 : Poll Experts and Get Results

the integrated system. Next, the limitations imposed by the actual implementation are listed, and finally a sample system setup and scenario is presented.

This is an implementation of the OSDES approach described in chapter 3. Even though it is a limited version due to the DOS restrictions, it substantiates the idea presented in this project, that small expert systems can be integrated in a simple fashion.

CHAPTER 5

Discussion and Conclusion

In this project an Open System approach to integrate Distributed Expert Systems (OSDES) is presented. This approach tries to integrate the individual agents in a manner that is transparent to them. Transparency is required to reduce the effort needed to add the agents to the distributed system. Thus almost any agent (not necessarily an expert system) can be integrated. This approach has been implemented and unlike other approaches, places minimum skill requirements at the individual experts.

As demonstrated in chapter 4, the Open Systems approach to Distributed Expert Systems (OSDES) is a viable way to integrate distributed expert systems. It represents a unique contribution to the field of distributed problem solving. Some future enhancements and development opportunities, which would convert this system into a fully functional and operational system, remain open.

The OSDES approach utilizes the original Open Systems Approach, and the centralized version of Multiagent Planning discussed in chapter 2. In addition this approach provides an easy interface to accommodate today's expert systems.

5.1 Advantages Of OSDES

This section will re-emphasize the advantages and strengths of the OSDES approach which satisfies most of the distributed problem solving objectives:

1. Any expert system can be added to the system if it uses one of the interfaces described earlier.

2. A larger set of expert systems is addressed here, since there is no extra intelligence or skills (such as negotiation skills) required in the individual expert systems.

3. Unlike most of the distributed problem solving approaches, OSDES is an actual implementation. It is not a simulation, it uses everyday networks and everyday expert systems.

4. The approach used in OSDES can be used to integrate any program (or agent which does not have to be an expert system) on the network, as long as it satisfies the general interface requirements.

5. Most of the objectives of distributed problem solving approaches are implemented in OSDES. These objectives are:

(i) Increasing performance and throughput through parallelism.

(ii) Increasing the set or scope of achievable tasks.

(iii) Increasing the probability that the solution can be found through redundancy.

(iv) Implementing a modular approach, where each expert system feeds the next with viable information.

(v) Simplifying problem decomposition.

(vi) Implementing crash and recovery protocols.

(vii) Using any network or expert system. That is, it is network and application independent.

5.2 Issues of Concern in OSDES

Even though OSDES solves the issues that the other approaches ignore, it has some limitations in terms of functional intelligence and application areas. These considerations are listed below along with ideas on how to overcome these limitations.

1. By simplifying the decomposition problem, the system does not have the intelligence to negotiate a solution like some other approaches. On the other hand this extra intelligence, say to integrate results, is left to the developer. The developer of an expert system (or the administrator of the integrated system) can add another expert system to each group that is only responsible for integrating the results, thus shifting the responsibility away from the existing systems.

2. There is no corroboration in the OSDES approach. Experts can not verify each others results. Considering the ignorance of most of the expert systems, this is not surprising. The administrator, however could add new experts to system that perform these tasks.

3. OSDES is very useful for applications where the system interacts with a user, since the communicator can always fall back on the users response (say in the case of a crash of an agent). Applications such as Distributed Air Traffic Control require higher levels of intelligence and performance.

4. Although OSDES is not a centralized system in the traditional sense, it still has a few bottlenecks. The communicator at the user side acts as a scheduler, and can potentially be overloaded. Similarly the machine running the Experts Directory Assistance, can be overloaded or (even worse) crash, in which case the whole system is halted. Depending on the level of security (safety) required, some of these

components can be duplicated or mirrored.

5.3 Future Opportunities for Improvements

The current implementation of OSDES is operational and can be used to integrate a set of distributed expert system. Just like any other system, there are functions that can be enhanced or added to make the system more useable and user friendly.

1. The system as it is now allows two types of interfacing, I/O redirection and file passing. One improvement would be to allow other types of interfacing e.g., memory mailboxes and interactive remote I/O. In memory mailboxes the communicator interacts with the expert system through passing the parameters in predefined memory locations. Remote interactive I/O should provide the expert systems with a set of function calls (or executables) that the expert could use to interact with the remote user.

2. Due to the current DOS limitations, the communicator had to be split into two major parts, the communicator itself and the user interface module. The user can not use a system running the communicator program, since the communicator is an RPC server process. Adopting this system to a UNIX or OS/2 environment should solve this problem.

3. Neither the crash recovery portion of the OSDES nor the process of building dependencies at the query time was implemented.

APPENDIX A

Parameter Files

Adding a new member to OSDED requires a parameter file that contains all the information a communicator needs to know about that member. The expert system developer has to provide the communicator with this file which holds all the information about the expert system, and about the methods of interfacing. In the following paragraphs the format of the parameter file will be presented, each entry in the parameter file will be described in a separate paragraph.

Expert System Name :

The name used by the communicator to add the new expert system to the EDA. This name is also used when a communicator calls another one.

Topics Covered :

The list of topics that the expert system deals with. These topics are shown to the user as the available topics.

Input File Name :

The name of the file which holds the information needed by the expert system (only for

certain interfaces such as file passing and standard I/O redirection).

Output Results File Name :

The name of the file which holds the final results, generally a text file. (only for certain interfaces)

Output Factors File Name :

The name of the file that holds the output factors and their values. These factors and values are used by the calling communicator as input to the next expert system in the dependency list.

Memory Location :

A memory address to hold the address of the input/output buffer between the GAI and the agent. (only for certain interfaces)

List of Factors Questions and Domains :

The list of the factors that have to be passed to the expert system. The questions are the ones that the communicator should ask, and the domain specifies what type of answers are expected.

Invocation method :

The command the communicator should execute when activating the local expert system.

Interface style :

There are several ways to interface the communicator with an existing expert system. Since this is the part that heavily depends on the expert system architecture, the expert system developer (Administrator) has to research and investigate the most appropriate method (if any). Following is a list of possible interface styles, there are many others. Note that, there might be some communicator code modifications involved if it does not support that interface yet.

1. Input/Output Redirection : This is the simplest method, and applies to those expert systems that read their input from the standard input device and produce their output to the standard output device.

2. Input variables/Output Reports : Some of the expert systems can read any of the - yet unbound- variable and their values from a file. Similarly the output variables or reports can be placed in some file. This is the method utilized by the expert system building tool (1-st Class) used in this projects. For example for an Insurance application processing system, the input variables can be passed in file as :

```
AGE=18
STATUS=SINGLE
ACCIDENT=3
```

The result can then be read from a file in the form :

```
INSURANCE=DENIED
```

or, the result can be place in a report as follows :

THE APPLICANTS INSURANCE APPLICATION CAN NOT BE APPROVED

AT THIS TIME, CALL YOUR AGENT FOR FURTHER ASSISTANCE.

Sample Parameter File

```
CREDIT
FRUN CREDIT /IYXRB
CREDIT.IN
CREDIT.RPT
CREDIT APPROVAL ADVISOR
#
CRED_RATING
WHAT IS YOUR CREDIT RATING ? (one only of: excellent. good, poor)
YEARSATJOB
HOW MANY YEARS HAVE YOU BEEN EMPLOYED BY YOUR CURRENT COMPANY ? (Number)
INCOME
WHAT IS YOUR ANUAL INCOME IN DOLLARS ? (Number)
ADDRESS
IN WHAT TYPE OF SETTING DO YOU LIVE ? (only one of: suburban, urban)
#
```

APPENDIX B

Source Code

```
 1 +--%h{
 2 |  #define  MAX_EXPERTS   3
 3 |  #define  MAX_TOPICS    (MAX_EXPERTS * 3)
 4 |  #define  MAX_FACTORS   (MAX_EXPERTS * 7)
 5 |  #define  STRLEN        30
 6 |  #define NAME      20
 7 |  #define  LINELEN       80
 8 |  #define  REPORT_SIZE   512
 9 +--%}
10
11 +--TYPEDEF STRUCT Factor {       /* FACTOR
   |  consists of      */
12 |     CHAR    fName[NAME] (*$0 =='\0');
   |       /* Factor's Name      */
13 |     CHAR    fDomain[LINELEN] (*$0 =='\0');
   |       /* Decription of domain      */
14 |     CHAR    fValue[NAME] (*$0 =='\0');
   |       /* Factor's Value       */
15 +--} FACTOR ;
16
17 +--TYPEDEF   STRUCT ExpertInfo {       /* Expert
   |  Description      */
18 |     CHAR    eName[NAME]  (*$0 =='\0');
   |       /* How to invoke the Expert   */
19 |     CHAR    eInvoke[STRLEN] (*$0 =='\0');
   |       /* How to invoke the Expert   */
20 |     CHAR    eInput[NAME]  (*$0 =='\0');
   |       /* Input file Name      */
21 |     CHAR    eOutput[NAME] (*$0 =='\0');
   |       /* Results file Name      */
22 |     CHAR    (eTopics[MAX_TOPICS][STRLEN] (
   |     *$0 =='\0'));  /* Expert's Topics   */
23 |     FACTOR  eFactors[MAX_FACTORS] ;
   |        /* Expert's Factors  */
24 +--} EXPERT_INFO ;
25
26 +--TYPEDEF   STRUCT expertsGroup {
27 |     INT    expertsCount ;
28 |     CHAR   eName[MAX_EXPERTS][NAME];
29 |     CHAR   eInvoke[MAX_EXPERTS][STRLEN];
30 |     CHAR   eInput[MAX_EXPERTS][NAME];
31 |     CHAR   eOutput[MAX_EXPERTS][NAME];
32 |     CHAR   eTopics[MAX_TOPICS * STRLEN];
33 |     FACTOR    eFactors[MAX_FACTORS];
34 +--} EXPERTS_GROUP ;
35
```

```
36    /*
37     * File: experts\common\experts.rpc
38     *
39     * This RPC file runs in the following environment
       :
40     *  RPC TOOL 2.2.0, Novelle SPX, DOS 3.3
41     *
42     * This is the RPC Specification file for the
       experts example.
43     */
44
45    /*
46     * A global variable called sname is identified
       as the variable containing
47     * process binding information.  Because this
       variable is only
48     * used for input process binding information
       and because no
49     * output process binding information is
       specified, the client
50     * stubs open and close connections for each
       remote procedure
51     * call (i.e., this is a non-persistent connection
       ).
52     */
53    EXTERN server_name [bind in] sname;  /* server_nam
      e is a predefined binding
54                      type (typedef char *server_name
                        ) */
55
56    /* The following procedures use the global
      variable sname implicitly
57     * for obtaining process binding information.
       The procedures add the
58     * elements of vec and return the sum.
59     */
60
61    INT
62    announce(
63    EXPERT_INFO [in out] *anExpert ,
64    CHAR [in] *cfgName (*$0 == '\0')
65    ) ;
66
67    INT
68    trySolve(
69    EXPERT_INFO [in] *expertInfo ,
```

```
70    CHAR [in out] resultText[REPORT_SIZE]
71    ) ;
72
```

```
1   #
2   # File: experts\client\client.mk
3   # Makefile FOR the 'experts' client program
4   # This makefile runs in the following environment:
5   #   RPC TOOL 2.2.0, Novelle SPX, DOS 3.3
6   #
7
8   COMMON=..\common
9   CC=cl
10  MODEL=S
11  MLIB=C_LIBS
12  CFLAGS=/A$(MODEL) /Od
13  LFLAGS=/SE:256 /STACK:0X4800  /CO
14  RPCC=rpcc
15  NOBJS=
16  LIBS=$(MODEL)rpc $(MODEL)nwspx fdr_5$(MODEL)
    samlib
17  INCS=
18  RINCS=
19
20  experts.h: $(COMMON)\experts.rpc
21  $(RPCC) $(RINCS) /c $(COMMON)\experts.rpc /o
    cstubs.c
22
23  cstubs.c: $(COMMON)\experts.rpc
24  $(RPCC) $(RINCS) /c $(COMMON)\experts.rpc /o
    cstubs.c
25
26  cstubs.obj: cstubs.c experts.h
27  $(CC) $(CFLAGS) /c $(INCS) cstubs.c
28
29  user_io.obj: user_io.c experts.h
30  $(CC) $(CFLAGS) /c /Zi $(INCS) user_io.c
    >user_io.err
31
32  client.obj: client.c experts.h
33  $(CC) $(CFLAGS) /c /Zi $(INCS) client.c
    >client.err
34
35  client.exe: client.obj cstubs.obj user_io.obj
36  # Create the response file FOR the linker
37  echo client.obj cstubs.obj user_io.obj >client.lnk
38  echo client.exe  >>client.lnk
39  echo $(LFLAGS)  >>client.lnk
40  echo $(LIBS) >>client.lnk
41  link @client.lnk
```

```
42      erase client.lnk
```

```
1   /*
2    * Generated by Netwise C/RPC TOOL
3    * MS/DOS - Microsoft C 5.1, Version 2.02.03
4    */
5   #ifndef EXPERTS_H
6
7   #define EXPERTS_H 1
8   #include "rpchdr.h"
9
10   #define   MAX_EXPERTS    3
11   #define   MAX_TOPICS     (MAX_EXPERTS * 3)
12   #define   MAX_FACTORS    (MAX_EXPERTS * 7)
13   #define   STRLEN         30
14   #define NAME       20
15   #define   LINELEN        80
16   #define   REPORT_SIZE    512
17  +--TYPEDEF STRUCT Factor {
18  |     CHAR fName[NAME];
19  |     CHAR fDomain[LINELEN];
20  |     CHAR fValue[NAME];
21  +--} FACTOR;
22  +--TYPEDEF STRUCT ExpertInfo {
23  |     CHAR eName[NAME];
24  |     CHAR eInvoke[STRLEN];
25  |     CHAR eInput[NAME];
26  |     CHAR eOutput[NAME];
27  |     CHAR eTopics[MAX_TOPICS][STRLEN];
28  |     FACTOR eFactors[MAX_FACTORS];
29  +--} EXPERT_INFO;
30  +--TYPEDEF STRUCT expertsGroup {
31  |     INT expertsCount;
32  |     CHAR eName[MAX_EXPERTS][NAME];
33  |     CHAR eInvoke[MAX_EXPERTS][STRLEN];
34  |     CHAR eInput[MAX_EXPERTS][NAME];
35  |     CHAR eOutput[MAX_EXPERTS][NAME];
36  |     CHAR eTopics[MAX_TOPICS * STRLEN];
37  |     FACTOR eFactors[MAX_FACTORS];
38  +--} EXPERTS_GROUP;
39    EXTERN server_name sname;
40    EXTERN INT  announce( );
41    EXTERN INT  trySolve( );
42
43   #endif
```

```
 1   /*
 2    * File: experts\client\client.c
 3    *
 4    * This example runs in the following environments
      :
 5    *  RPC TOOL 2.2.0, Novelle SPX, DOS 3.3
 6    *
 7    * Client code for the experts example. This
      driver program calls two server
 8    * programs, opening and closing the connection
      for each call, i.e., a
 9    * non-persistent connection.
10    */
11
12   #include <stdio.h>
13   #include <fdrlib.h>
14   #include "experts.h"  /* client header file,
     created by RPC compiler */
15
16
17   /* Server_Name is used to set the process-
     binding variable.
18    * It must be defined as the name the server
      registers under.
19    */
20
21   #define Server_Name "EXAMPLE"
22
23   EXTERN INT _rpcerr_;  /* declare RPC error code */
24
25   /* declare variable of type server_name for
     process binding */
26   server_name sname;
27
28   STATIC VOID  place(EXPERT_INFO *, /* in */
     EXPERTS_GROUP []) ;
29   STATIC VOID  merge(EXPERT_INFO *, /* with */
     EXPERTS_GROUP *);
30
31   main(argc, argv)
32   INT argc ;
33   CHAR   **argv ;
34  +--{
35  |    STATIC CHAR          resultText[REPORT_SIZE] ;
36  |    STATIC EXPERT_INFO    anExpert ;
37  |    STATIC   EXPERTS_GROUP  experts[MAX_EXPERTS] ;
```

```
38  |
39  |     INT   i , index , num_experts ;
40  |
41  |     fdclr(0,0,24,79, ' ', fdpen(P_NORMAL)) ;
42  |     IF (argc < 2)
43  |  +--{
44  |  |     printf("\n\n USAGE : client <expert1>
    |  |     [expert2] [expert3]\n");
45  |  |     exit(0) ;
46  |  +--}
47  |
48  |     memset((CHAR *) experts, '\0', SIZEOF(
    |     experts)) ;
49  |
50  |      // fill the array of experts
51  |     FOR ( i = 1 ; i < argc ; i++)
52  |  +--{
53  |  |     memset(&anExpert, '\0', SIZEOF(anExpert)) ;
54  |  |     sname = argv[i] ;
55  |  |     announce(&anExpert, sname) ;
56  |  |     place(&anExpert, experts) ;
57  |  +--}
58  |
59  |      // count the number of expert groups
60  |     FOR (num_experts = 0 ; experts[num_experts].
    |     eTopics[0] ; num_experts++ );
61  |
62  |     DO
63  |  +--{
64  |  |     index = getTopic(experts, num_experts) ;
65  |  |     IF (index < 0)
66  |  |        BREAK ;
67  |  |
68  |  |     IF (getFactors(&experts[index]) == 0)
69  |  |        BREAK ;
70  |  |
71  |  |     memcpy(anExpert.eFactors, experts[index].
    |  |     eFactors,
72  |  |     SIZEOF(anExpert.eFactors)) ;
73  |  |
74  |  |     fdclr(0,0,24,79, ' ', fdpen(P_NORMAL)) ;
75  |  |     fdcurset(0,0);
76  |  |     FOR (i = 0 ; experts[index].eName[i][0] ;
    |  |      i++)
77  |  |  +--{
78  |  |  |     strcpy(anExpert.eName   , experts[index]
```

```
   | | |     .eName[i]) ;
 79 | | |     strcpy(anExpert.eInvoke , experts[index]
   | | |     .eInvoke[i]) ;
 80 | | |     strcpy(anExpert.eInput  , experts[index]
   | | |     .eInput[i]) ;
 81 | | |     strcpy(anExpert.eOutput , experts[index]
   | | |     .eOutput[i]) ;
 82 | | |     sname = anExpert.eName ;
 83 | | |     printf("CALLING ==> %s \n", sname) ;
 84 | | |     fdstfill(resultText, REPORT_SIZE - 1 ,
   | | |     ' ') ;
 85 | | |     trySolve(&anExpert, resultText) ;
 86 | | |     IF (_rpcerr_)
 87 | | |  +--{     /* check RPC return code */
 88 | | |  |     printf("CLIENT: RPC error %d \n",
 89 | | |  |     _rpcerr_);
 90 | | |  |     exit(1);
 91 | | |  +--}
 92 | | |     printf("ADVICE IS :\n%s\n", resultText) ;
 93 | |  +--}
 94 | |
 95 | |     printf("\n\n press any key when ready ");
 96 | |     fdkbget(&resultText[0]) ;
 97 |  +--}
 98 |     WHILE (1);
 99 |
100 |     exit(0);
101  +--}
102
103     STATIC VOID  place(anExpert, /* in */ allExperts)
104     EXPERT_INFO     *anExpert ;
105     EXPERTS_GROUP   allExperts[] ;
106  +--{
107  |     INT   i = 0 , k = 0 ,
108  |     found = 0;
109  |
110  |     FOR (i = 0 ; allExperts[i].eTopics[0] ; i++)
111  |  +--{
112  |  |     FOR (k = 0 ; anExpert->eTopics[k][0] ; k++)
113  |  |  +--{
114  |  |  |     IF (strstr(allExperts[i].eTopics,
   |  |  |     anExpert->eTopics[k]))
115  |  |  |  +--{ // if found the merge
116  |  |  |  |     merge(anExpert, &allExperts[i]) ;
117  |  |  |  |     found = 1 ;
118  |  |  |  |     BREAK ;     // continue with the
```

```
    |  |  |  |      next expert
119 |  |  |  +--}
120 |  |  +--}
121 |  +--}
122 |
123 |     IF (!found)
124 |        merge(anExpert, &allExperts[i]) ;      //
    |        add to a new group
125 +--}
126
127    STATIC  VOID merge(anExpert, /* with */ eGroup)
128    EXPERT_INFO     *anExpert ;
129    EXPERTS_GROUP   *eGroup ;
130 +--{
131 |     INT   j = 0, k = 0 ;
132 |
133 |     strcpy(eGroup->eName[eGroup->expertsCount],
    |     anExpert->eName) ;
134 |     strcpy(eGroup->eInvoke[eGroup->expertsCount],
    |     anExpert->eInvoke) ;
135 |     strcpy(eGroup->eInput[eGroup->expertsCount],
    |     anExpert->eInput) ;
136 |     strcpy(eGroup->eOutput[eGroup->expertsCount],
    |     anExpert->eOutput) ;
137 |
138 |      // Merge the Topics
139 |     FOR (k = 0 ; anExpert->eTopics[k][0] ; k++)
140 |  +--{
141 |  |      // if Topic does not exists then added to
    |  |      the list
142 |  |     IF (!strstr(eGroup->eTopics, anExpert->
    |  |     eTopics[k]))
143 |  |  +--{
144 |  |  |     IF (strlen(eGroup->eTopics) > 50)
145 |  |  |        strcat(eGroup->eTopics, ",\n") ;
146 |  |  |     ELSE
147 |  |  |        strcat(eGroup->eTopics, ", ") ;
148 |  |  |     strcat(eGroup->eTopics, anExpert->
    |  |  |     eTopics[k]) ;
149 |  |  +--}
150 |  +--}
151 |
152 |      // Merge the factors
153 |     FOR (k = 0 ; anExpert->eFactors[k].fName[0] ;
    |     k++)
154 |  +--{
```

```
155  |  |     FOR (j = 0 ; eGroup->eFactors[j].fName[0] ;
     |  |      j++)
156  |  |  +--{
157  |  |  |     IF (!strcmp(anExpert->eFactors[k].fName,
     |  |  |      eGroup->eFactors[j].fName))
158  |  |  |        BREAK ;     // Factor already exists
     |  |  |        in the group
159  |  |  +--}
160  |  |
161  |  |     IF (!(eGroup->eFactors[j].fName[0]))   //
     |  |     end of factors -> add factor
162  |  |        memcpy(&eGroup->eFactors[j], &anExpert->
     |  |        eFactors[k],
163  |  |        SIZEOF(FACTOR));
164  |  +--}
165  |
166  |     eGroup->expertsCount++ ;          // increment
     |     # of experts in the group
167  +--}
168
```

```
 1   #include    <stdio.h>
 2   #include    <stdlib.h>
 3   #include    <string.h>
 4   #include    <dos.h>
 5   #include    <fdrlib.h>
 6   #include    <experts.h>
 7
 8   USHORT   keylist[] = { KB_ENTER, KB_ESC, NULL };
 9
10   STRING str =
11  +--{
12  |    NULL,                    // special
    |    option flag
13  |    NULL,                    // maximum
    |    length of string
14  |    20,                     // width of
    |    field display
15  |    0,                      // current
    |    scrolling offset
16  |    0,                      // cursor
    |    position in field display
17  |    NULL,                    // value to
    |    be edited
18  |    NULL,                    // valid/
    |    invalid input characters
19  |    NULL,                    // formatted
    |    input mask
20  |    20, 20,                 // absolute
    |    row, column
21  |    P_REVERSE, P_NORMAL,          // attribute
    |    pen numbers
22  |    keylist,                // array of
    |    keys that terminate
23  |    NULL                     // override
    |    of fdkbget() function
24  +--};
25
26   INT getTopic(experts, num_experts)
27   EXPERTS_GROUP   experts[] ;
28   INT num_experts ;
29  +--{
30  |    INT i , j ;
31  |    CHAR   selection[3] ;
32  |
33  |    fdclr(0,0,24,79, ' ', fdpen(P_NORMAL)) ;
34  |    fdcurset(0,0);
```

```
35  |     FOR (i = 0 ; i < num_experts ; i++)
36  |       printf("\n%d. %s", i, experts[i].eTopics);
37  |
38  |     fdstfill(selection, 3 - 1, ' ') ;
39  |     IF (get_string(&str, selection, 20, 1,
    |     STR_UPPER,
40  |       "Select a Topic Number") == KB_ESC)
41  |       RETURN(-1) ;
42  |
43  |     RETURN (atoi(selection)) ;
44  |
45  +--}
46
47    INT getFactors(eGroup)
48    EXPERTS_GROUP   *eGroup ;
49  +--{
50  |     INT  i , j,
51  |     sts = 0 ;
52  |
53  |     fdclr(0,0,24,79, ' ', fdpen(P_NORMAL)) ;
54  |     fdcurset(0,0);
55  |     FOR (i = 0 , j = 0; eGroup->eFactors[i].fName[
    |     0] ; i++, j+=2)
56  |   +--{
57  |   |     fdstfill(eGroup->eFactors[i].fValue, NAME
    |   |     - 1, ' ') ;
58  |   |     IF (get_string(&str, eGroup->eFactors[i].
    |   |     fValue, j, 0, STR_UPPER,
59  |   |       eGroup->eFactors[i].fDomain) == KB_ESC)
60  |   |       RETURN(sts) ;
61  |   +--}
62  |
63  |     RETURN(1) ;   // all ok
64  +--}
```

```
1   #
2   # File: experts\server\server.mk
3   # Makefile FOR the 'experts' server program
4   # This makefile runs in the following environment:
5   #   RPC TOOL 2.2.0, Novelle SPX, DOS 3.3
6   #
7
8   COMMON=..\common
9   SCPS=$(RPCSCP)
10  CC=cl
11  MODEL=S
12  MLIB=C_LIBS
13  CFLAGS=/A$(MODEL) /Od
14  LFLAGS=/SE:256 /STACK:0X4000
15  RPCC=rpcc
16  NOBJS=
17  DEFINES=/DSERV_DEF
18  LIBS=$(MODEL)rpc $(MODEL)nwspx fdr_5$(MODEL)
19  INCS=
20  RINCS=
21
22  experts.h: $(COMMON)\experts.rpc
23  $(RPCC) $(RINCS) /s $(COMMON)\experts.rpc /o
    sstubs.c
24
25  sstubs.c: $(COMMON)\experts.rpc
26  $(RPCC) $(RINCS) /s $(COMMON)\experts.rpc /o
    sstubs.c
27
28  sstubs.obj: sstubs.c experts.h
29  $(CC) $(CFLAGS) /c $(INCS) sstubs.c
30
31  # Note that main_sc.c is in the server control
    procedure directory rather
32  # than in the local example directory
33  scp.obj: serv_def.h
34  $(CC) $(CFLAGS) /c /Foscp.obj /I. $(INCS) $(
    DEFINES) $(SCPS)\main_sc.c
35
36  rproc.obj: rproc.c experts.h
37  $(CC) $(CFLAGS) /c $(INCS) rproc.c
38
39  server.exe: scp.obj rproc.obj sstubs.obj
40  # Create the response file FOR the linker
41  echo scp.obj rproc.obj sstubs.obj >server.lnk
42  echo server.exe >>server.lnk
```

```
43    echo $(LFLAGS) >>server.lnk
44    echo $(LIBS) >>server.lnk
45    link @server.lnk
46    erase server.lnk
```

```
1    /*
2     * File: experts\server\serv_def.h
3     * This include file is used to define the
      constants needed by the
4     * server control procedure.
5     */
6
7    #define Dispatcher experts_Proc
8    #define Server_Name "shiper"
9    #define DEBUG
```

```
 1   /*
 2    * Generated by Netwise C/RPC TOOL
 3    * MS/DOS - Microsoft C 5.1, Version 2.02.03
 4    */
 5   #ifndef EXPERTS_H
 6
 7   #define EXPERTS_H 1
 8   #include "rpchdr.h"
 9
10    #define   MAX_EXPERTS    3
11    #define   MAX_TOPICS     (MAX_EXPERTS * 3)
12    #define   MAX_FACTORS    (MAX_EXPERTS * 7)
13    #define   STRLEN         30
14    #define NAME       20
15    #define   LINELEN        80
16    #define   REPORT_SIZE    512
17  +--TYPEDEF STRUCT Factor {
18  |     CHAR fName[NAME];
19  |     CHAR fDomain[LINELEN];
20  |     CHAR fValue[NAME];
21  +--} FACTOR;
22  +--TYPEDEF STRUCT ExpertInfo {
23  |     CHAR eName[NAME];
24  |     CHAR eInvoke[STRLEN];
25  |     CHAR eInput[NAME];
26  |     CHAR eOutput[NAME];
27  |     CHAR eTopics[MAX_TOPICS][STRLEN];
28  |     FACTOR eFactors[MAX_FACTORS];
29  +--} EXPERT_INFO;
30  +--TYPEDEF STRUCT expertsGroup {
31  |     INT expertsCount;
32  |     CHAR eName[MAX_EXPERTS][NAME];
33  |     CHAR eInvoke[MAX_EXPERTS][STRLEN];
34  |     CHAR eInput[MAX_EXPERTS][NAME];
35  |     CHAR eOutput[MAX_EXPERTS][NAME];
36  |     CHAR eTopics[MAX_TOPICS * STRLEN];
37  |     FACTOR eFactors[MAX_FACTORS];
38  +--} EXPERTS_GROUP;
39    EXTERN INT  announce( );
40    EXTERN INT  trySolve( );
41
42    #endif
```

```
1    /*
2     * File: experts\server\rproc.c
3     *
4     * This example runs in the following environment:
5     *  RPC TOOL 2.2.0, Novelle SPX, DOS 3.3
6     *
7     * This file contains the remote procedures for
       the array example.
8     */
9
10    #include    <stdio.h>
11    #include     <stdlib.h>
12    #include     <string.h>
13    #include     <dos.h>
14    #include     <fdrlib.h>
15    #include    "experts.h"  /* server header file,
      created by RPC compiler */
16
17    INT trySolve(expertInfo, resultText)
18    EXPERT_INFO     *expertInfo ;
19    CHAR            *resultText ;
20  +--{
21  |    STATIC   CHAR  line[80] ;
22  |    FILE     *fp ;
23  |    INT      i ;
24  |
25  |    printf("\nTRY SOLVE\n");
26  |     // 1.  first create the input file holding
    |     all the factors in the form of
27  |     // FAVTOR = value (FACTOR is as decribed in
    |     the EXPERT, & value is as
28  |     // filled by the user.
29  |    fp = fopen(expertInfo->eInput, "wt") ;
30  |     // while there are factors we need to put
    |     them to the file
31  |    FOR (i = 0 ; (expertInfo->eFactors[i].fName[0]
    |     != '\0') ; i++ )
32  |       fprintf(fp,"%s = %s \n",expertInfo->
    |       eFactors[i].fName,
33  |       expertInfo->eFactors[i].fValue ) ;
34  |    fclose(fp) ;
35  |
36  |
37  |     // 2. next zap the output file
38  |    remove(expertInfo->eOutput) ;
39  |
```

```
40  |     // 3. Now Invoke the Expert
41  |     system(expertInfo->eInvoke) ;
42  |
43  |     // 4. Now prepare the result text
44  |     resultText[0] = '\0' ;
45  |     fp = fopen(expertInfo->eOutput, "rt") ;
46  |     WHILE (! feof(fp) && fp)
47  |  +--{
48  |  |     IF (!fgets(line, SIZEOF(line) -1 ,fp))
49  |  |        BREAK ;              // nothing more to
    |  |        read
50  |  |     fdstftrim(line) ;
51  |  |     IF ((line[0] == '\n') || ((INT)line[0] <
    |  |     32))
52  |  |        CONTINUE ;           // skip if new
    |  |        line or non-print
53  |  |     printf("\n%s", line) ;
54  |  |     strcat(resultText, line) ;
55  |  +--}
56  |     fclose (fp) ;
57  |
58  |     printf("\n\n\n ------------------------------
    |     ------");
59  |     printf("\n%s", resultText) ;
60  |     printf("\n\n\n ------------------------------
    |     ------");
61  |
62  +--}  // end trySolve()
63
64
65
66     INT announce(anExpert, cfgName)
67     EXPERT_INFO     *anExpert ;
68     CHAR            *cfgName  ;          // Expert
       configuration file
69  +--{
70  |     STATIC CHAR  line[LINELEN] ;
71  |     FILE  *fp ;
72  |     INT   i,
73  |     sts = 0 ;         // 0 => did not announce, 1
    |     => announced OK.
74  |
75  |     printf("\nANNOUNCE\n");
76  |     sprintf(line, "%s.%s", cfgName, "dat");
77  |     fp = fopen(line,"rt") ;
78  |     IF (!fp)
```

```
 79  |       RETURN(sts) ;
 80  |
 81  |     // 1st  line is the Expert system name (
     |     to be announced)
 82  |     IF (fgets(line, SIZEOF(line)-1 , fp) == NULL)
 83  |       RETURN(sts) ;
 84  |     fdstftrim(line);
 85  |     fdsttrim(line) ;
 86  |     strcpy(anExpert->eName, line) ;
 87  |
 88  |     // 2nd  line is the methode of Invoking the
     |     EXpert system
 89  |     IF (fgets(line, SIZEOF(line)-1 , fp) == NULL)
 90  |       RETURN(sts) ;
 91  |     fdstftrim(line);
 92  |     fdsttrim(line) ;
 93  |     strcpy(anExpert->eInvoke, line) ;
 94  |
 95  |     // 3rd  line is the  file name that holds
     |     the input to the EXpert system
 96  |     IF (fgets(line, SIZEOF(line)-1 , fp) == NULL)
 97  |       RETURN(sts) ;
 98  |     fdstftrim(line);
 99  |     fdsttrim(line) ;
100  |     strcpy(anExpert->eInput, line) ;
101  |
102  |     // 4th line is the file name that holds the
     |     output from the expert system
103  |     IF (fgets(line, SIZEOF(line)-1 , fp) == NULL)
104  |       RETURN(sts) ;
105  |     fdstftrim(line);
106  |     fdsttrim(line) ;
107  |     strcpy(anExpert->eOutput, line) ;
108  |
109  |     // get the list of topics each on a line,
     |     terminated by '#'
110  |     // in a line by itself
111  |     memset(line, '\0', SIZEOF(line)) ;
112  |     FOR (i = 0 ; (line[0] != '#') ; i++)
113  |  +--{
114  |  |     IF (fgets(line, SIZEOF(line)-1, fp) == NULL)
115  |  |       RETURN (sts) ;          // unexpected
     |  |       end of file
116  |  |     fdstftrim(line);
117  |  |     fdsttrim(line) ;
118  |  |     IF (line[0] != '#')
```

```
119  |  |       strcpy(anExpert->eTopics[i], line) ;
120  |  +--}
121  |
122  |     // get All the factor names, each Factor
     |     name is followed by
123  |     // domain description used as info for the
     |     user, and terminated by '#'
124  |     memset(line, '\0', SIZEOF(line)) ;
125  |     FOR (i = 0 ; (line[0] != '#') && (!feof(fp)) ;
     |     i++)
126  |  +--{
127  |  |     fgets(line, SIZEOF(line)-1, fp) ;
128  |  |     fdstftrim(line);
129  |  |     fdsttrim(line) ;
130  |  |     IF (line[0] != '#')
131  |  |  +--{
132  |  |  |     IF (i % 2)              // od numbered
     |  |  |     lines
133  |  |  |        strcpy(anExpert->eFactors[i/2].
     |  |  |        fDomain,line) ;
134  |  |  |     ELSE
135  |  |  |        strcpy(anExpert->eFactors[i/2].fName,
     |  |  |        line) ;
136  |  |  +--}
137  |  +--}
138  |
139  |     RETURN (1) ;
140  +--}
141
142
143
144
145
```

BIBLIOGRAPHY

Alsc90 Al-Schamma, S. and Z. Chen. "An Open System Approach To Integration of Expert System.", APPLIED ARTIFICIAL INTELLIGENCE, 1990, 4:3:145-154.

Amir89 Amir, S. "Building Integrated Expert Systems.", AI EXPERT, January 1989, 4:1:26-37.

Amir89 Amir, S. "Building Integrated Expert Systems, Part Two.", AI EXPERT, March 1989, 4:3:42-52.

Bond88 Bond, A. H. and Gasser, L. "An analysis of problems and research in DAI. Readings in Distributed Artificial Intelligence, 1988, 1:3-35.

Borl87 Borland. TURBO PROLOG TOOLBOX, Borland Inc. 1987.

Cohe89 Cohen, B. "Merging Expert Systems and Databases.", AI EXPERT, February 1989, 4:2:22-31.

Covi88 Covington, M. A., Donald Nute, and Velline. PROLOG Programming in Depth, Scott Foresman and Company, London 1988. 9:256-261, 12:380-381.

Degr87 Degroff, L. "Conventional Languages and Expert Systems.", AI EXPERT, April 1987, 2:4:32-36.

Durf89 Durfee, E. H., Victor Lesser, and Corkill. "Trends in Cooperative Distributed Problem Solving.", IEEE Transactions on Knowledge and Data Engineering, March 1989, 1:1.

Gaag88 Van Der Gaag, L.C., and P. J. F. Lucas. "HEPAR:An Expert System in Prolog.", AI EXPERT, June 1988, pp. 34-43.

Gass89 Gasser, L. "Distributed Artificial Intelligence.", AI EXPERT, July 1989, 4:7:26-33.

Geva87 Gevarter, W. B. "The Nature and Evaluation of Commercial Expert System Building Tools.", COMPUTER, May 1987, pp. 24-44.

Hewi86 Hewitt, C. "Offices are open systems". ACM Transactions on Office Information Systems, July 1986, pp. 271-287.

Howa89 Howard, H. C. and Rehak, D. R. "KADBASE:Interfacing Expert Systems with

Databases.", IEEE Expert, Fall 1989, pp. 65-76.

Hu 89 Hu, D. C/C++ FOR EXPERT SYSTEMS, MIS PRESS 1989.

Jack86 Jackson, P. Introduction to Expert Systems, Reading, MA:Addison-Wesley, 1986.

Luge89 Luger, G. F. and Subblefield, William A. ARTIFICIAL INTELLIGENCE AND THE DESIGN OF EXPERT SYSTEMS, Benjamin/Cummings 1989 , 15:561-563, 16:598-600.

Netw89 Netwise RPC TOOL for C Language Reference Manual, Netwise Inc., September 1990.

Pede88 Pedersen, K. "Connecting Expert Systems and Conventional Environments.", AI EXPERT, May 1988, 3:5:26-35.

Smit81 Smith, R. G., and Randall Davis. "Frameworks for Cooperative Distributed Problem Solving.", IEEE Transactions on Systems, Man and Cybernetics, January 1981, 3.1:61-69.

Stee86 Steeb, R. and others. "Distributed Problem Solving for Air Fleet Control:Framework and Implementation.", EXPERT SYSTEMS:Techniques and Applications, Ed. Philip Klahr and Donald A Waterman: Addison-Wesley Publishing Company 1986. 11:391-432.

Thom89 Thomas, W. and Hapgood, W. 1st-Class Reference Manual, Expert Systems Inc. 1989.

Will87 Williamson, M. "At Du Pont, Expert Systems Are Key to AI Implementation.", PC WEEK, January 13, 1987, 4:2.