NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

ARCHITECTURE AND ALLOCATION CONSIDERATIONS FOR
GROUP EXPERT SYSTEMS

by

Michael Bernard Rattigan

March 1988

Thesis Advisor: Taracad Sivisankaran

Approved for public release; distribution is unlimited

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION Unclassified		1b RESTRICTIVE MARKINGS		
2a SECURITY CLASSIFICATION AUTHORITY		3 DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution is unlimited		
2b DECLASSIFICATION / DOWNGRADING SCHEDULE				
4 PERFORMING ORGANIZATION REPORT NUMBER(S)		5 MONITORING ORGANIZATION REPORT NUMBER(S)		
6a NAME OF PERFORMING ORGANIZATION Naval Postgraduate School	6b OFFICE SYMBOL (If applicable) Code 54	7a NAME OF MONITORING ORGANIZATION Naval Postgraduate School		
6c ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000		7b ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000		
8a NAME OF FUNDING / SPONSORING ORGANIZATION	8b OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER		
8c ADDRESS (City, State, and ZIP Code)		10 SOURCE OF FUNDING NUMBERS		

			PROGRAM ELEMENT NO	PROJECT NO	TASK NO	WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification)
ARCHITECTURE AND ALLOCATION CONSIDERATIONS FOR GROUP EXPERT SYSTEMS

12 PERSONAL AUTHOR(S)
Rattigan, Michael Bernard

13a TYPE OF REPORT Master's Thesis	13b TIME COVERED FROM ___ TO ___	14 DATE OF REPORT (Year, Month, Day) 1988 March	15 PAGE COUNT 82

16 SUPPLEMENTARY NOTATION "The views contained herein are those of the student and not those of the Naval Postgraduate School or the United States Government."

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB GROUP	Distributed or Group Expert Systems; Allocation; Architecture

19 ABSTRACT (Continue on reverse if necessary and identify by block number)
This thesis investigates the design, communication, and allocation considerations for implementing a distributed group expert system on a Local Area Network. A model system called GESP (Group Expert System Prototype) was implemented in Prolog on a microcomputer LAN to be used as a working platform. From observations of the model, conclusions have been drawn concerning: (1) the architecture of the expert system software required to support an interactive group expert system; (2) implications of expert system to expert system communication; and (3) the optimum allocation strategy of expert systems to nodes. Due to the lack of a distributed operating environment in which to implement the model, efficiency has been sacrificed for operability. Although GESP is not a fully practical implementation of a group expert system, it should as a minimum provide a functional framework for understanding, analyzing, and designing interactive group expert systems.

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS		21 ABSTRACT SECURITY CLASSIFICATION Unclassified	
22a NAME OF RESPONSIBLE INDIVIDUAL Prof. Taracad Sivisankaran		22b TELEPHONE (Include Area Code) (408) 646-2637	22c OFFICE SYMBOL Code 54Sj

DD FORM 1473, 84 MAR
83 APR edition may be used until exhausted
All other editions are obsolete

Architecture and Allocation Considerations for
Group Expert Systems

by

Michael Bernard Rattigan
Lieutenant, United States Naval Reserve
B.S., University of North Carolina at Greensboro, 1977

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN INFORMATION SYSTEMS

from the

NAVAL POSTGRADUATE SCHOOL
March, 1988
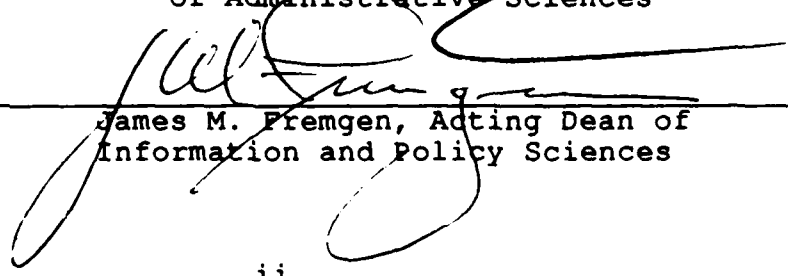
Author: _____
Michael Bernard Rattigan

Approved by: _____
Taracad Sivasankaran, Thesis Advisor

_____
Tung Bui, Second Reader

_____
David R. Whipple, Chairman, Department
of Administrative Sciences

_____
James M. Fremgen, Acting Dean of
Information and Policy Sciences

ii

ABSTRACT

This thesis investigates the design, communication, and allocation considerations for implementing a distributed group expert system on a Local Area Network. A model system called GESP (Group Expert System Prototype) was implemented in Prolog on a microcomputer LAN to be used as a working platform. From observations of the model, conclusions have been drawn concerning: (1) the architecture of the expert system software required to support an interactive group expert system; (2) implications of expert system to expert system communication; and (3) the optimum allocation strategy of expert systems to nodes. Due to the lack of a distributed operating environment in which to implement the model, efficiency has been sacrificed for operability. Although GESP is not a fully practical implementation of a group expert system, it should as a minimum provide a functional framework for understanding, analyzing, and designing interactive group expert systems.

Accession For		
NTIS GRA&I	☒	
DTIC TAB	☐	
Unannounced	☐	
Justification		
By		
Distribution/		
Availability Codes		
Dist	Avail and/or Special	
A-1		

QUALITY INSPECTED 2

iii

THESIS DISCLAIMER

The reader is cautioned that computer programs developed in this research may not have been exercised for all cases of interest. While every effort has been made, within the time available, to ensure that the programs are free of computational and logic errors, they cannot be considered validated. Any application of these programs without additional verification is at the risk of the user.

TABLE OF CONTENTS

I. INTRODUCTION

A. BACKGROUND

Group expert systems offer tremendous potential support to strategic decision making. Top level managerial decisions are often made by a group or by a single person after consultation with a group. Distributed expert systems can facilitate group decisions whether the organizational structure is centralized or decentralized.

The issue of centralization verses decentralization acquires a new dimension for debate when viewed in the context of distributed expert systems. Organizational structure and behavior clearly surface as the central factors in the question of decentralization. The physical location of control of decision making authority and responsibility replaces the traditional arguments of economics verses enduser productivity as the platform debate.

The concept of distributing expert systems implies that some knowledge and responsibility for decision making is distributed. On the smallest scale, all experts within the group may be located in the same building. A Group Expert System (GES) implemented on a local area network could, at the very least, conserve the amount of time spent in meetings solving reoccurring problems. In a decentralized organization implemented on a large scale, such as a wide area network, a GES could realize significant savings in timeliness and effectiveness of organizational decisions.

Distributed expert systems have many of the same design and implementation problems as other types of distributed systems. Attempts to optimize performance within constraints are key considerations. Typical problems include: the physical location of resources, replication of data, interprocess and intermodule communications, and the determination of the optimal number and location of nodes.

1

B. OBJECTIVES

This study will looks at the issues related to analyzing, designing, and implementing a distributed group expert system. Some of the major issues addressed are how domain concepts, rules / heuristics, and control strategies are distributed based on system and communication costs.

C. RESEARCH QUESTIONS

The primary research question is one of basic implementation. Is the implementation of a distributed expert system possible on a Local Area Network?

If LAN implementation is possible, what should be the general architecture?

What are the implications of node to node communication and of interprocess and intermodule communication? Also, what are the implications of expert system to expert system communication including calling processes, rule passing, data passing, and interactive expert systems?

The final question deals with the actual distribution of systems. What is the optimum allocation strategy of expert systems to nodes?

D. SCOPE

A networked prototype distributed expert system has been implemented on six nodes of a Local Area Network. The prototype system is called GESP, Group Expert System Prototype. GESP solves a meta problem which is a security clearance screening for employees. It employs multiple expert systems and multiple knowledge domains. Implementation of GESP was constrained by the operating system of the IBM PC LAN, which does not directly support distributed computing. This limitation was overcome by using a commonly available drive for a blackboard as a means of communication. Although GESP was implemented only on a Local Area Network, design

2

considerations, architecture, allocation, and communications are discussed in broader context.

E. METHODOLOGY

An architecture has been proposed by which expert systems can communicate on a microcomputer Local Area Network using a commonly available drive as a blackboard. A prototype system, GESP (Group Expert System Prototype), has been developed using Prolog. It is capable of supporting three knowledge bases. The model has been implemented on a PC LAN for six nodes. Implications for system allocation and architecture for group expert systems have been induced from the model. Optimization formulas from operations research have been used to conduct studies of cost and allocation problems. Formulas for determining system cost and allocation for expert systems have been derived.

F. SUNNARY OF FINDINGS

This study has shown, by actual implementation of a model system, that implementation of distributed expert systems on a Local Area Network is possible. The implementation requires an architecture consisting of a communication structure, a meta expert system to decompose the problem, consult appropriate expert systems, and synthesize a solution, and individual expert systems to form solutions within their respective domains. It has also been shown that group expert system allocation can be optimized by minimizing system cost. A method of determining expert system cost has been demonstrated.

G. ORGANIZATION OF STUDY

The remaining chapters will describe the thesis research. Specifically, Chapter II presents an introduction to group expert systems. It provides an overview of expert system support for group decision making and supports the use of

3

artificial intelligence in group decision support. Chapter III describes the architecture used in the model group expert system and proposes a structure for group expert systems in general. Chapter IV presents communication and allocation considerations for group expert systems. It details a method for measuring system cost and thereby allocating expert systems to nodes on a network. Chapter V draws conclusions from this study and proposes directions for future research.

II. GROUP EXPERT SYSTEMS

Group expert systems, GES, represent the next generation of intelligent systems which will provide support to management. Current expert systems operate on a stand alone basis. Each expert system has problem solving expertise in only a specific domain. However, strategic decision making in management often requires the coordinated assessment and evaluation of multiple areas of managerial expertise. The development of group expert systems can greatly enhance the quality and timeliness of strategic decision making.

Group expert systems function in a similar manner as do group decision support systems, GDSS. There has been some debate about whether or not an effective decision support system is in fact an expert system. The use of expert systems as decision support systems is the motivation to develop group expert systems. Expert systems represent "tremendous potential in providing the ultimate assistance to decision makers involved in serious business activities". (Isett, 1985, p.21)

Waterman (1986, pp.10-12) suggests that there are five advantages to substituting artificial expertise for human expertise. First of all, artificial intelligence is permanent. Human experts must constantly exercise their skills in order to maintain proficiency. System code does not decay through lack of use. Expert systems are not affected by personnel turnover. If a manager leaves the firm, that expertise also perishes. Once an expert system is coded, expertise becomes corporate knowledge.

Secondly, artificial expertise is easily portable. Transferring expert knowledge from one human being to another is difficult, costly, and time intensive. Porting artificial intelligence from one location to another is as simple and cheap as copying a program. Further more, less experienced managers gain knowledge through the use of expert systems.

5

Documentation of knowledge is another advantage. It is much simpler to document system code than to thoroughly explain an expert's thought process.

Artificial expertise is more consistent and reliable. Artificial intelligence does not perform differently in a crisis situation because of stress, time pressures, or emotional factors.

Finally, artificial expertise is cheaper than managerial expertise. Portability is a major factor contributing to its low cost. It is far more expensive to hire additional managers to satisfy need for expertise at multiple locations.

In addition to the advantages described above, Waterman also lists five drawbacks to using artificial expertise. These disadvantages apply to stand alone expert systems. Some of these will be eliminated by employing GES.

Creativity and adaptability go hand-in-hand. There is still much progress to be made in developing systems that can learn and adapt to new situations. Group expert systems go a long way in adding the dimension of separate multiple problem domains. However, while a system may consult another for input into solving its own problem, it still may not adapt another system's algorithm or heuristic to its own internal solution.

The whole world concept encompasses two more disadvantages, the sensory experience and common sense. The human can look at the whole picture at once and, drawing from a wide range of experiences, see how each piece fits. Group expert systems support human interaction only to the point necessary to solve the problem. If a more extensive human interface is designed for less structured problems, human creativity, adaptability, and common sense can be maximized.

The final drawback to artificial expertise is a narrow problem focus. Waterman not only lists narrow focus as a disadvantage but as a criteria for building expert systems. (Waterman, 1986, p. 26) He states, "An expert system has

6

depth; that is, it operates effectively in a narrow domain containing difficult and challenging problems". This is true for stand alone expert systems. Group expert systems combine multiple domains of expertise to solve a meta problem. In doing so, GES can support broad scope strategic management problems.

A group expert system in its simplest form is composed of a meta expert system, MES, and multiple single-domain expert systems. The meta ES is concerned with problem-domain relationships and domain-domain relationships. The MES identifies the problem to be solved and decomposes it into individual problem domains. It identifies the expert systems whose domains are pertinent to the solution.

The problem solving is then devolved the remote ES. The remote ES reads the problem devolved, develops a solution, and returns the solution to the MES. The MES then synthesizes the solution. Group Expert System Prototype is an example of how a GES solves a meta problem. The problem in this thesis is whether or not to grant a security clearance to a particular employee. There are three problem-domain relationships associated with this question. To conduct a security screen, each employee must successfully complete a financial profile study, criminal profile study, and a psychological study.

There are six expert systems in the GES, including the MES. Three are in the credit domain, CREDIT_1, CREDIT_2, and CREDIT_3. CREDIT_1 and CREDIT_2 have a peer-to-peer relationship within the domain, and CREDIT_3 is independent. CRIME_1 is the criminal domain and PSYCH_1 in the psychological domain. All domain-domain relationships are independent. Any dependency that exists should be captured in the rule base.

The MES, META, calls other expert systems to conduct all or part of the screening process. The criminal expert system looks into the criminal database and generates a criminal

7

record score. Likewise, the other expert systems look into their respective databases to calculate scores. All systems could use a central employee database if it contained adequate information. CREDIT_1 also has the ability to consult CREDIT_2 to conduct a more extensive check. The remote expert systems pass their scores to META which accepts them as inputs to the meta problem.

The assumption is that the problem is partitionable. In a group management decision scenario partitioning is intrinsic to the nature of the problem. If it were not, only one decision maker would be required. Strategic management decisions involve not only multiple experts, but often these managers are at remote locations. Group expert systems provide a cost effective platform by which scarce expert resources may be accessed.

The traditional comparison of expert systems to decision support systems sees two major differences, how the systems are used and the types of problems they solve. The use of expert systems as decision support tools is generally accepted, however the difference is that expert systems typically replace the expert. With group expert systems it is not necessary to eliminate an expert user. The goal is to minimize the involvement of expert management, whose time is a scarce resource.

The greatest advantage of GES over stand alone ES is the type of problems they can solve. Current ES are designed to solve structured, well-defined, and somewhat repetitive problems with a narrow and predictable domain. Data is usually symbolic, factual, and procedural in content. These ES are unlike DSS which solve unstructured and ill-defined problems of a broad, complex, and unpredictable nature. DSS support ad-hoc inquiries handling data which is numerical and factual in content. Modification to a DSS is more difficult than to an ES, as a DSS is more rigid.

Group expert systems provide managers with the best of both DSS and ES. Distributed expert systems can interface with users at all nodes on the network as do GDSS. The most significant factor of GES is their ability to solve broad-scope problems involving multiple knowledge domains. Group decision support systems require interaction with a human expert. Group expert systems support such human interaction when necessary but do not require it. Coding human expertise frees the manager to perform other functions whenever direct human involvement is not required.

III.   ARCHITECTURE FOR GROUP EXPERT SYSTEMS

In the arena of group expert systems, there is the fundamental task of bringing the system to the user. End-user computing puts AI decision support systems within the user's reach both physically and economically, in terms of both system and budget constraints. However, there has been much debate about the feasibility of implementing AI systems on a PC.

The knowledge domain of a meta problem may be too large to fit into a PC, in fact it may be too broad for a single expert system. Factoring the meta domain into discrete sub-domains and distributing the sub-domain expert systems across a network allows multiple PC's to share the problem.

The general architecture required to support this concept consists of a communication mechanism, a meta expert system, and consultant expert systems. Ideally, the communication is problem independent, however, it may be imbedded within the domain expert systems.

The meta expert system manages the solution of the problem. It is responsible for problem acceptance and validation and problem-domain and domain-domain relationships. It decomposes the problem and devolves the problem to consultant expert systems. The meta expert system accepts the results from the consultants and synthesizes a solution. The general architecture is shown in Figure 3.1.

Group expert systems, GES, present an ideal platform for demonstrating how to implement distributed expert systems on PC's. The key to distributing expert systems is structured modular design. There are three levels of modularity applicable to GES. The first level is communication. In order to distribute expert systems, they must be able to talk to one another. That is, one system must call another; pass data, rules, solutions, and make calls to the operating system and the database. At some level, at least one expert
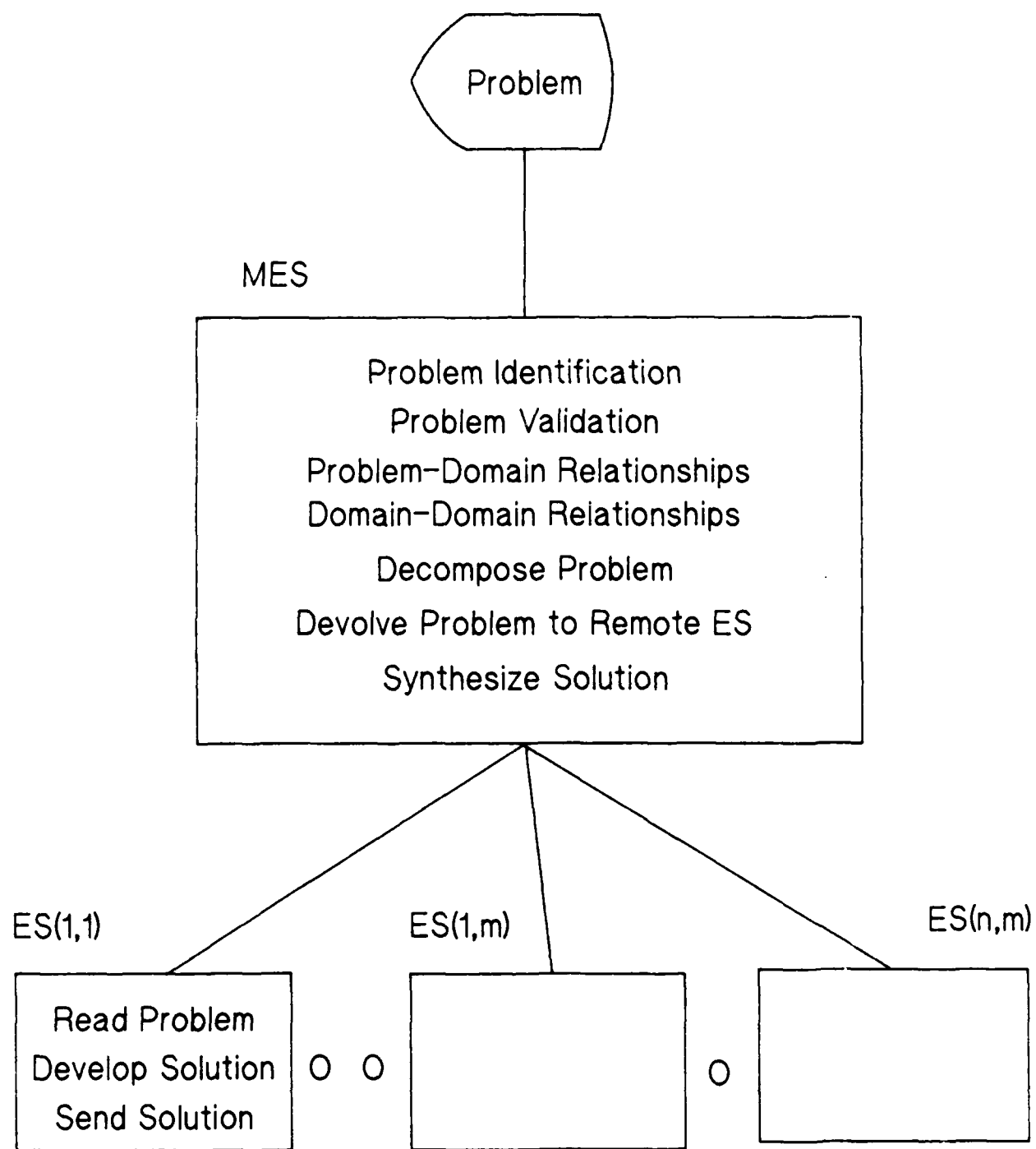
Figure 3.1   META MODELING FOR A GES

11

system must interface with the user. The control communication structure is located at the node where the problem originates. The control takes the problem input from the user, validates it, and decides which expert systems must be called to solve the problem. It accepts the inputs and solutions from the other expert systems and uses them to solve the higher level problem. The communication structures at the lower level expert systems allow them to communicate with the calling system and with each other, as in consulting a peer system. If the problem requires input from a user at remote node, as in a GDSS, a user interface at lower level can also be supported. To the greatest extent possible the communication architecture should be independent of the problem to be solved. The objective is to be able to solve a range of problems with multiple expert systems within one GES. The advantages of distinct separation of the problem from communication will be realized in a multitasking environment. The communication module could call any expert system by issuing an executable command and then reading the expert system output. Such a system should be extendable. It should be able to support solutions to other common expert system problems which are independent of the domain of expertise. (Biegl, 1986, p. 279)

The second level of modularity is the problem itself. The problem is factored by partitioning the knowledge domain. Each expert system represents a distinct part of the knowledge domain of a larger problem, as discussed in the previous chapter. Expert systems may interact vertically or horizontally to solve the problem. They may share a common database, pass rules or information from their own unique database to other systems, or pass their own solution to another system to be used to solve a separate problem. The top-level system can collect solutions from all other experts to solve a meta problem. The architecture proposed herein is one that supports the solution of this meta problem on a PC

12

LAN. Vertical and peer-to-peer system communication are required.

The third level of modularity is within a single expert system. The architecture for distributing a single expert system is best implemented in a distributed operating system environment. The problem to be solved by the system must be one which can be partitioned into distinct goals. Unique rule sets which are paths to goals may be partitioned into modules separate from unrelated rules and distributed to another processor. The initial state rules can be fired from a module located on a remote processor. The resultant of the goal state is then passed back to the remote calling module. This type of intermodule communication across remote processors is not practical to implement on the PC LAN, as the LAN was not designed to support distributed processing. The communications structure required to support an entire expert system would be necessary for each set of modules located on a remote processor. The overhead could only be justified for distributing a single large expert system. Distributing multiple expert systems is far more interesting, and the problem of the single system is solved in much the same way.

GESP, Group Expert System Prototype, is the initial screening of an individual for a security clearance. To pass the screening the candidate must have a satisfactory criminal, psychological, and credit records check. One criminal record database, three credit agencies, and one psychological record database are on the system. The control expert system, META, interfaces with the user. The problem to be solved, that is the type of screening to be conducted, is input by the user, and META calls the appropriate expert systems. There is a single fixed cost associated with consulting the criminal expert system, CRIME_1, and likewise, the psychological expert system, PSYCH_1. However, each credit expert system has a different cost associated with. The top-level system, META, may select either CREDIT_1 or

13

CREDIT_3. CREDIT_3 is the cheaper system, but it is less extensive than CREDIT_1 and only used for lower level clearances. CREDIT_1 is more expensive and is used for more sensitive clearances. It has the ability to consult CREDIT_2 on a peer-to-peer basis in order to obtain a higher quality solution.

The first and most obvious obstacle to implementing GESP is the fact that the IBM PC LAN operating system does not directly support distributed processing. It is not possible to directly call an expert system located on a remote processor or to directly pass it data. To establish system-to-system communication a virtual disk was created. The virtual disk is used as a blackboard to which all expert systems can read and write. In this respect GESP can not adhere to modular independence between the communication structure and the specific problem. A diagram of GESP is presented in Figure 3.2.

META calls other expert systems by writing the file, k_file.inp. This file contains the names of the expert systems to be called and all data necessary to begin execution. The other expert systems use a polling process to look for this file. When k_file.inp appears, each system checks the time stamp to see if the file is current. If the file is current it is opened and read. Each expert system then checks the list for membership to see if it is being called. The check is accomplished by using the member predicate. If the membership rule succeeds, the expert system begins execution of the problem. Each system returns a solution by writing the files crime_1.rep, credit_1.rep, and psych_1.rep respectively. CREDIT_1 calls CREDIT_2 in the same manner using the file, credit_1.inp. CREDIT_2 responds with the file, credit_2.rep.

All systems must constantly poll, checking the time stamp, to see if they are being called. If the META system knows on which processor each expert system is located the
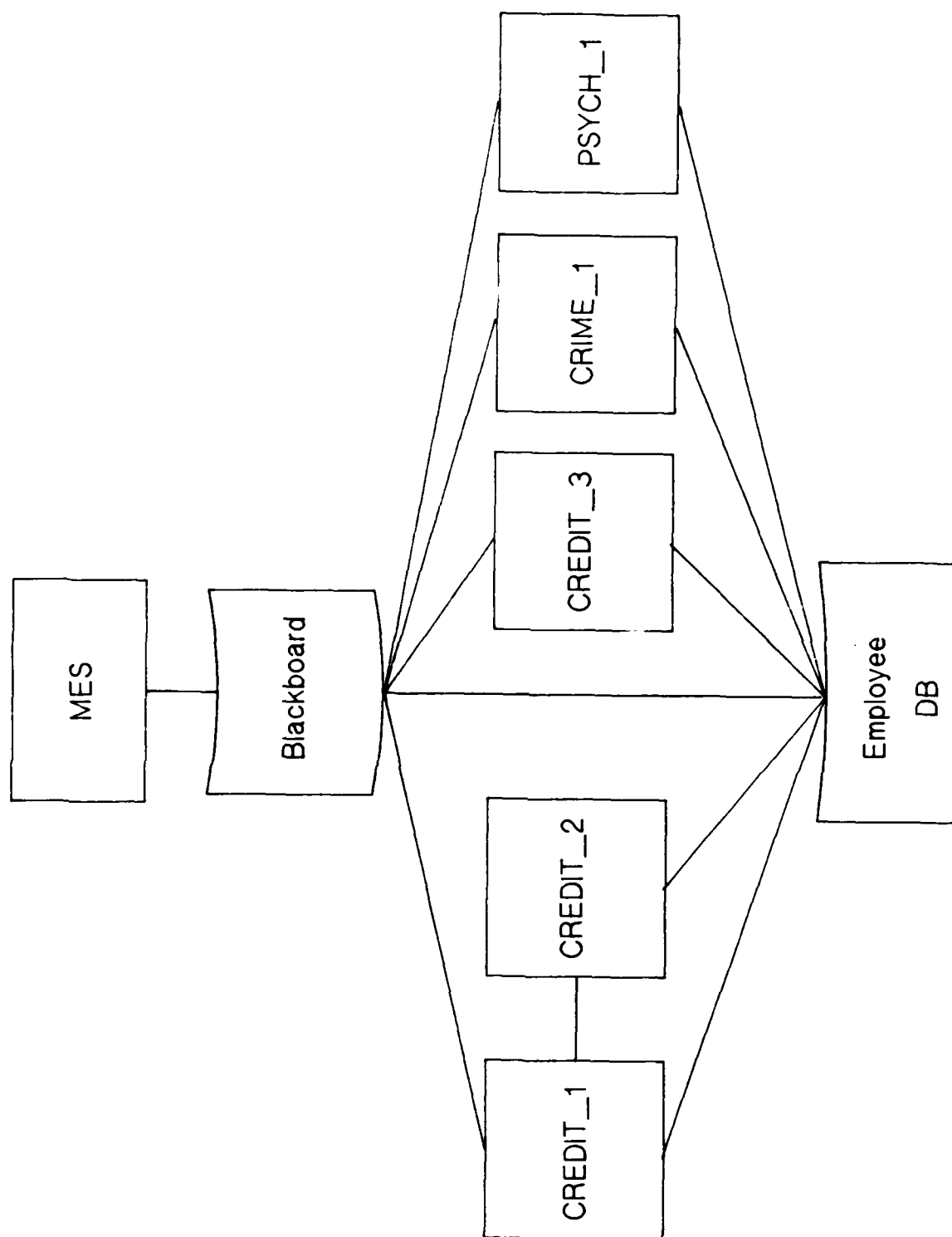
14

Figure 3.2   Prototype implemented on a LAN

15

process can be made much more efficient. META could make a call to the operating system telling it to send an interrupt signal to the desired processors. When the called processor receives the interrupt, it "wakes up" the polling process.

META, Figure 3.3, has a list of all problems that can be solved on the system in its database. Any problem input by a user is checked for validity. An incorrect problem submission will generate an error message to the user, and the system will cease execution and return to the initial state. Input of a valid problem begins execution, and META will determine which expert systems are to be consulted from the database. Each problem is associated with a list of expert systems required to solve it. The calling file, k_file.inp, is created, and the list of expert systems to be called is included. K_file.inp is then written to the virtual drive.

Immediately after the calling file is written, META goes into the polling process. It looks for a report file from each of the expert systems in the order in which they were called. Polling continues until a report file, for example crime_1.rep, appears. By use of the "directory" predicate META checks the time stamp on crime_1.rep and compares it to the time in its database. The base time is initialized to zero for the first run. If the times do not match, the file is determined to be current and is opened. The information provided by the remote expert system is read and asserted to the database. The polling process is again initiated, and the procedure is repeated until all expert systems have reported. The time stamp of the latest read of the report file is recorded.

META proceeds to evaluate the reports to determine if the candidate will satisfy the security screening requirements. Each expert system returns a score which is the measure of the candidates performance in each area. Point levels are awarded for discrepancies. For each area of evaluation there
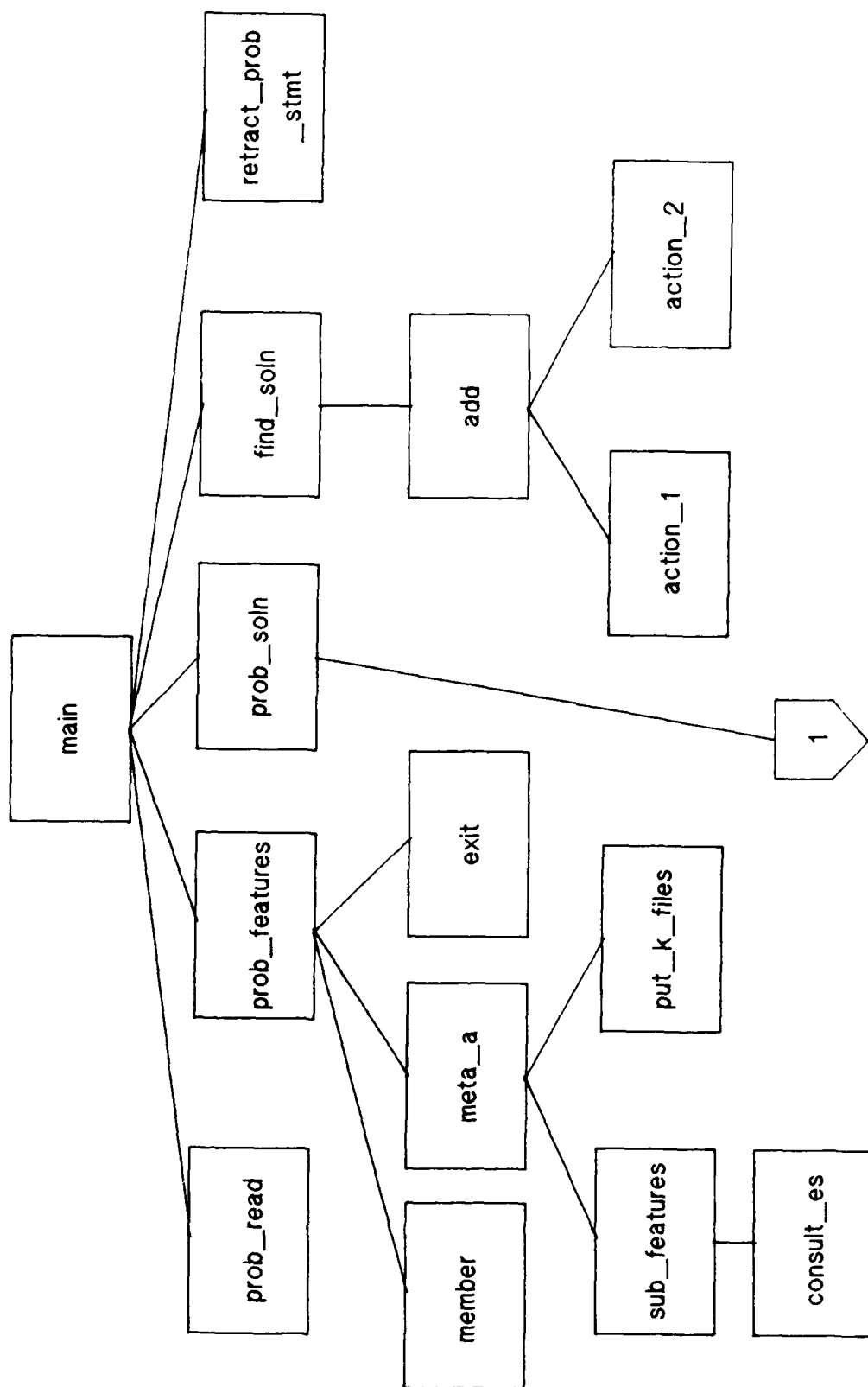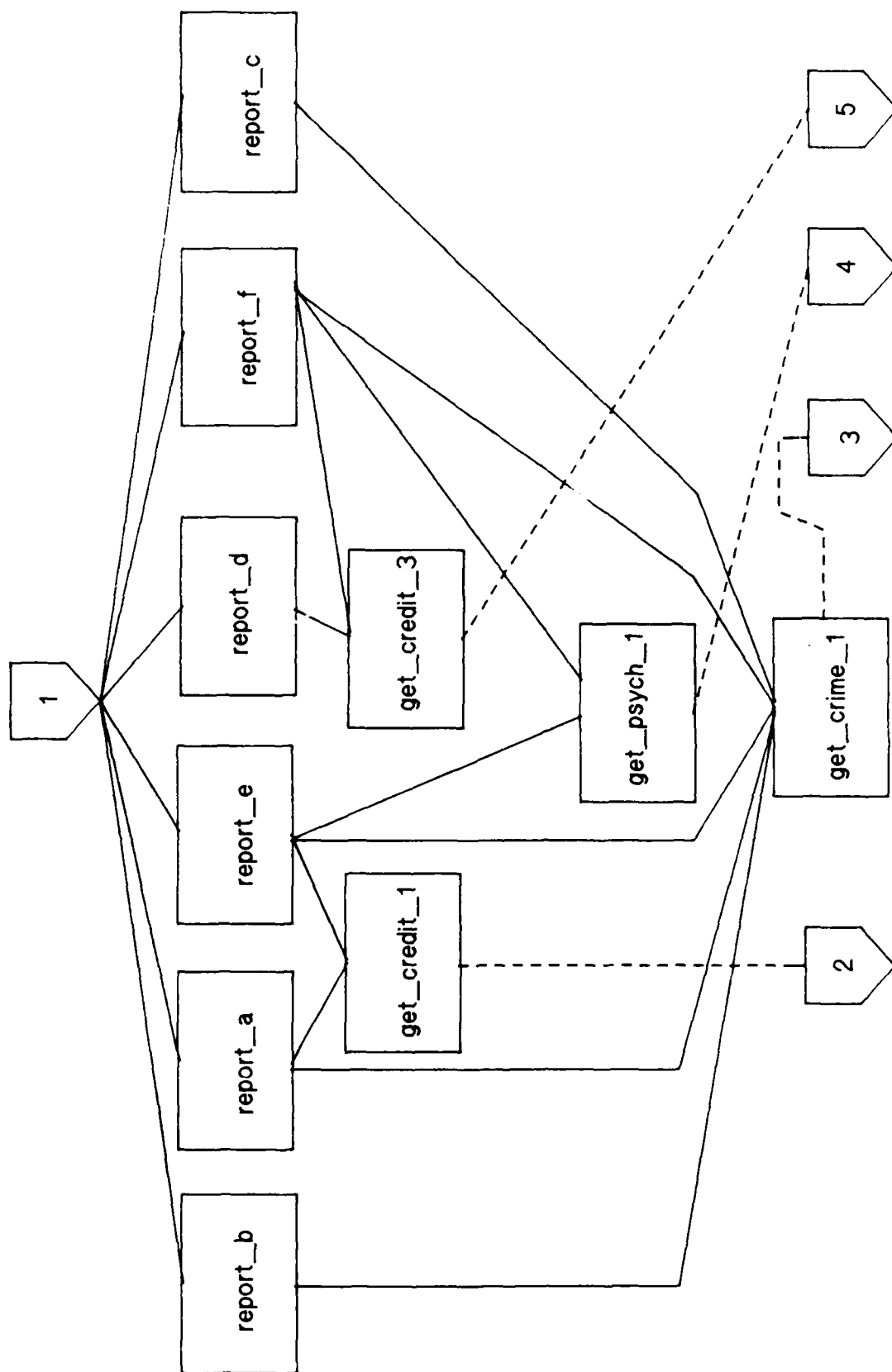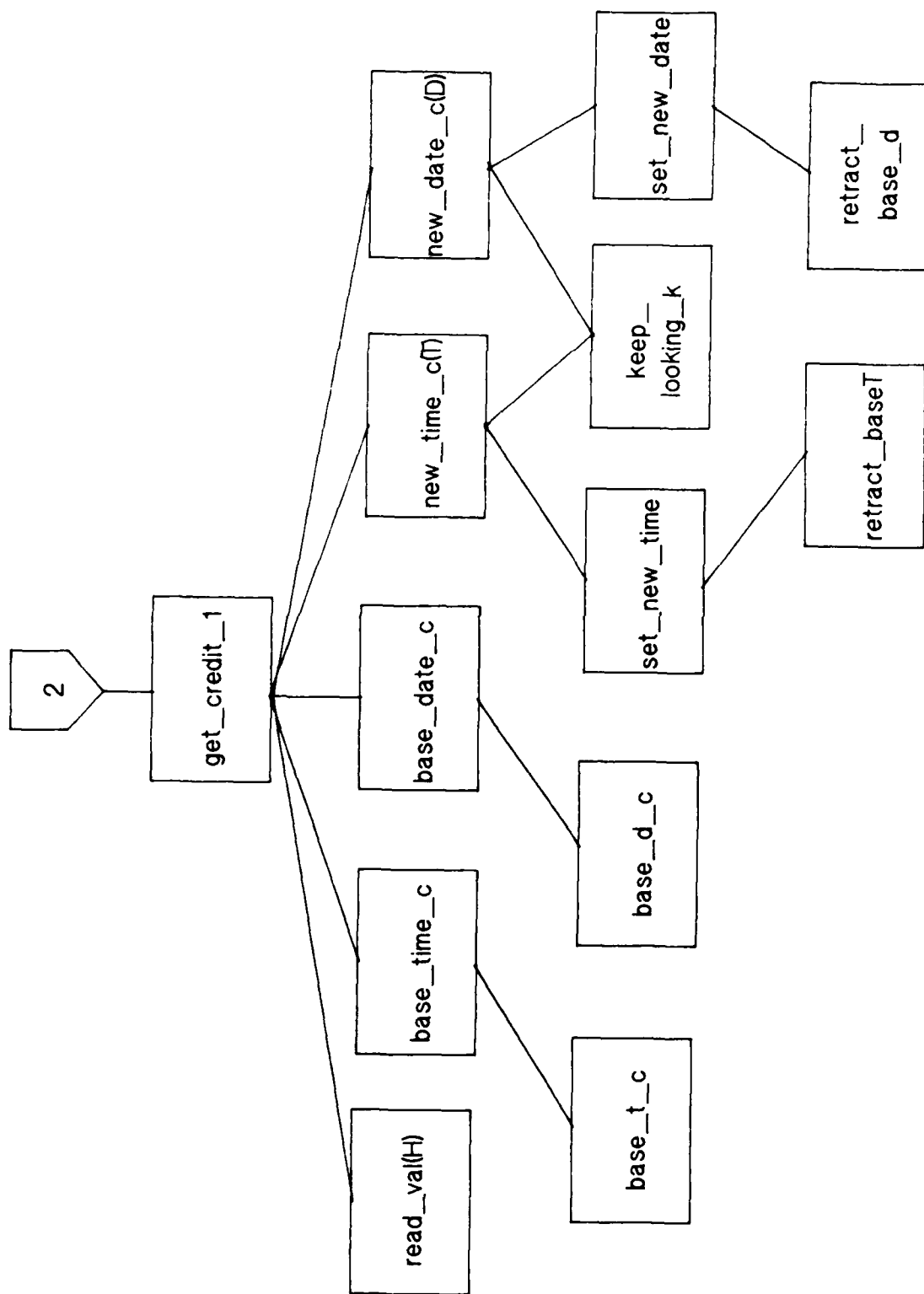
16

Figure 3.3   META

17

Figure 3.3 (cont.)
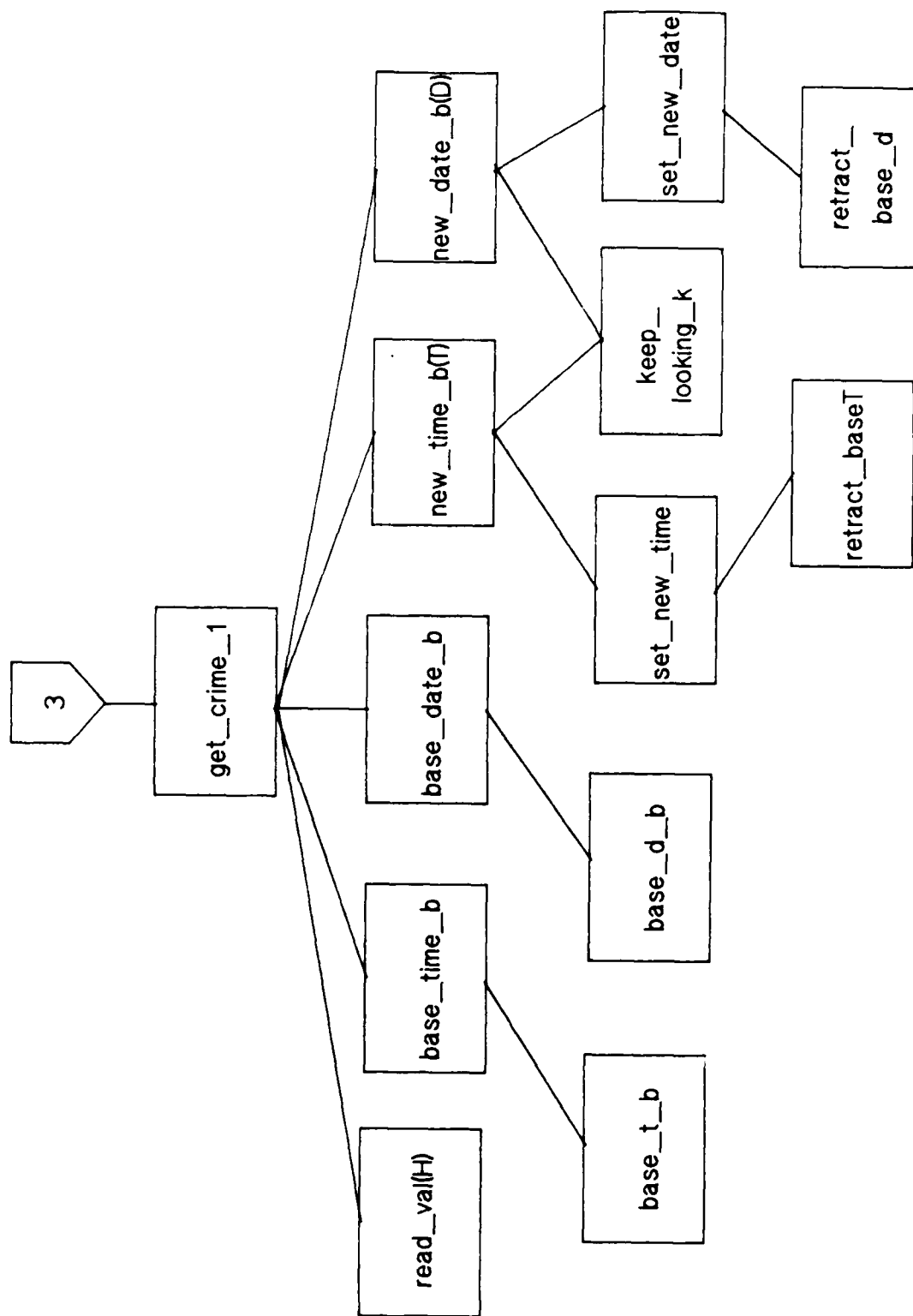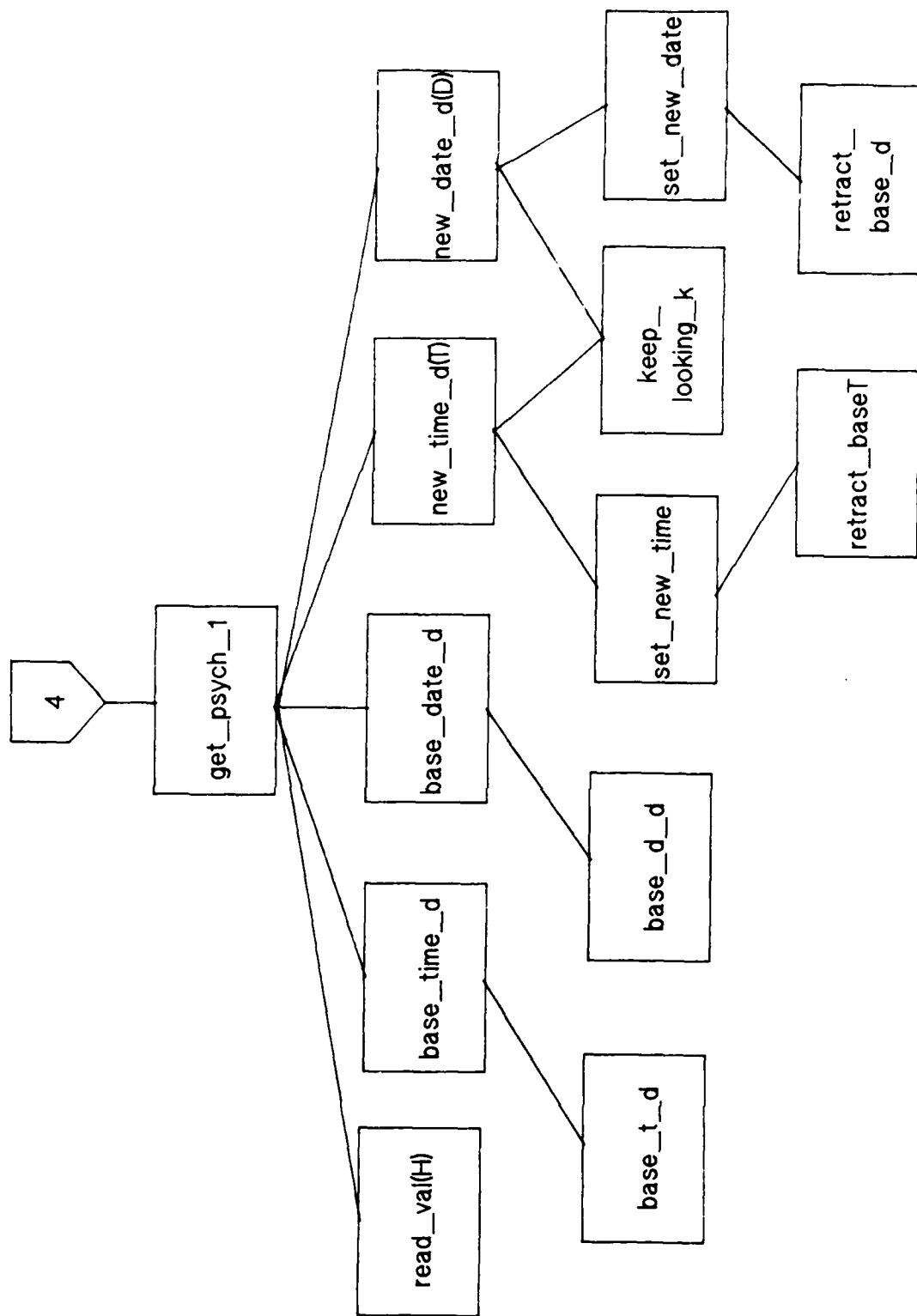
18

Figure 3.3(cont.)

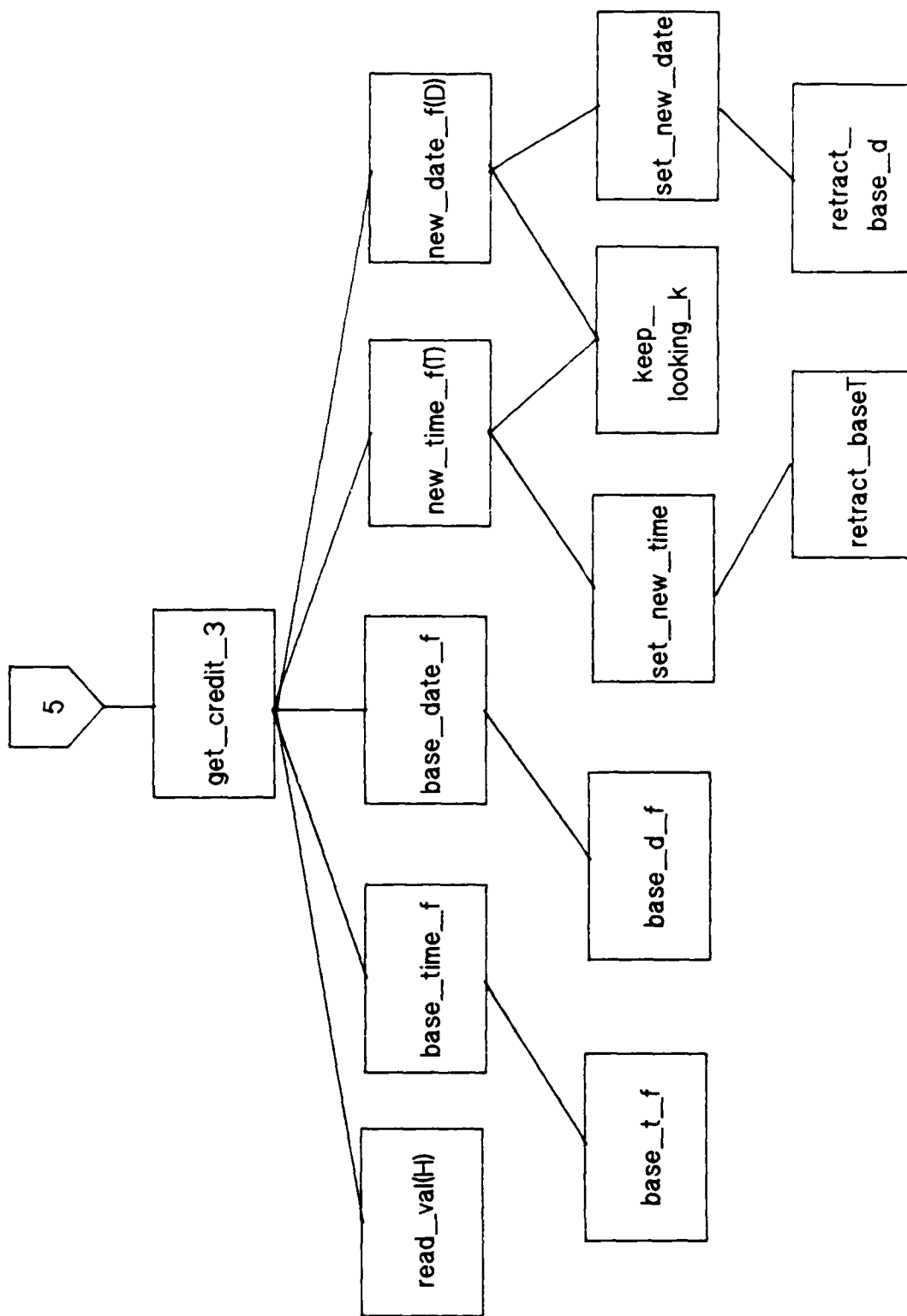19

Figure 3.3(cont.)

Figure 3.3(cont.)

21

Figure 3.3(cont.)

22

is a threshold level above which the candidate fails. The scores are then totaled. There is likewise a failing threshold for total score, accounting for the synergistic effect. It is possible that one might pass each area by a narrow margin but fail for having done significantly poorly in more than one area. META then displays the results to the user.

CRIME_1, PSYCH_1, and CREDIT_3, Figures 3.4, 3.5, and 3.6 respectively, are very similar in structure. They are called by and report directly to META. CRIME_1 will be used as an example. It uses the "directory" predicate to poll the virtual drive as does META. It also uses the same time stamp checking procedure and the same membership check. Upon determining the validity of the call, it executes a criminal record check. Upon determining a score, it creates the file crime_1.rep including the score and writes the file to the blackboard.

CREDIT_1, Figure 3.7, is called by and reports to META by the same process as do CRIME_1 and PSYCH_1. The unique feature of CREDIT_1 is peer-to-peer communication. It is this added feature which is the primary contributor to its higher consultation cost. The decision to consult may be delegated to a lower level or may exist at that lower level as an intrinsic part of the problem. In this case, the more extensive credit expert system decides whether or not to initiate further investigation based on its own first-cut findings. The decision to consult CREDIT_2, Figure 3.8, is based on a threshold score. If the candidate fails the initial check, a more detailed investigation can be prompted to see if a failing score is warranted.

The structure of the code used to call CREDIT_2 and to make and receive its report is the same as that of the higher level expert systems. Only the names of the calling and reporting files are changed. The call to an expert system at
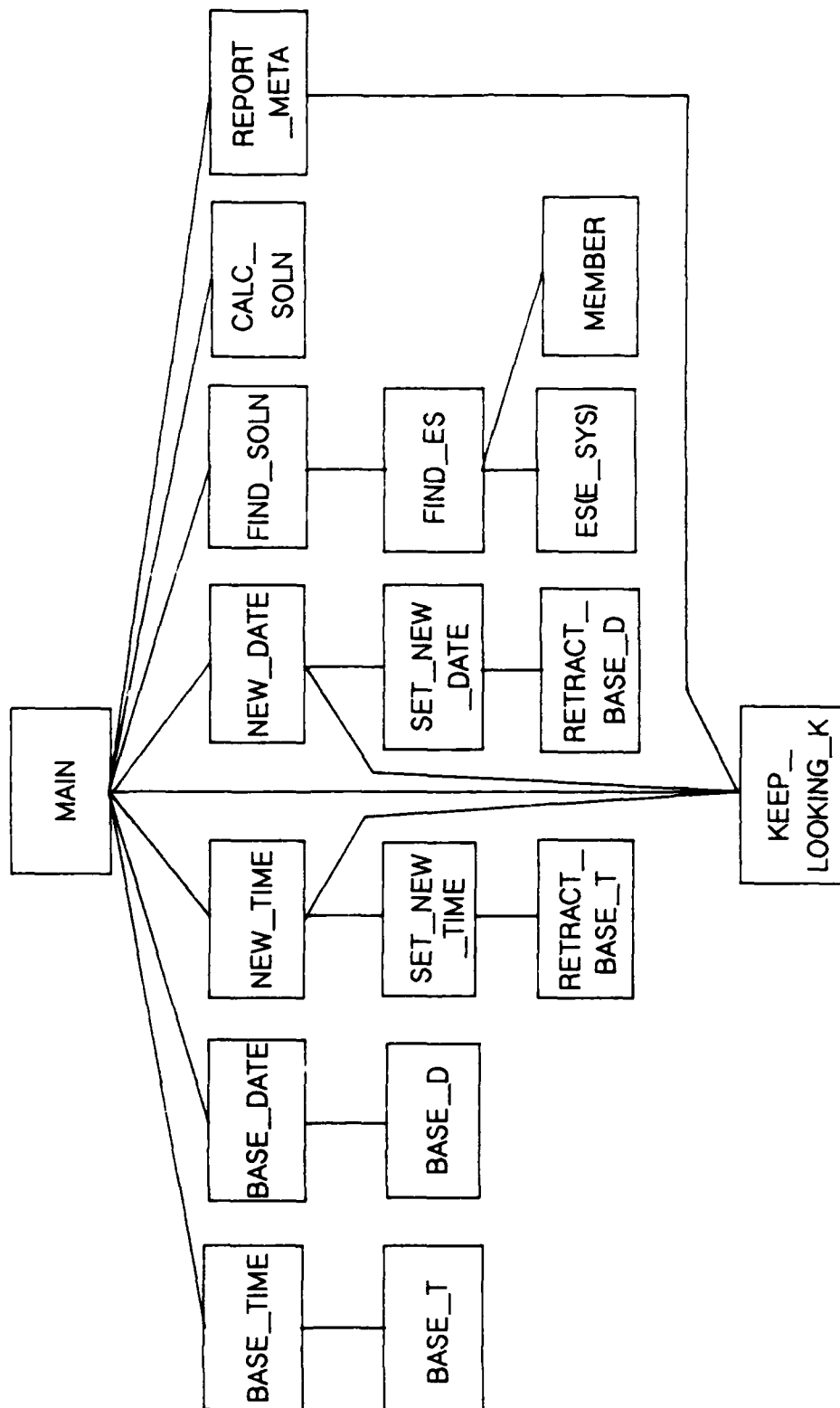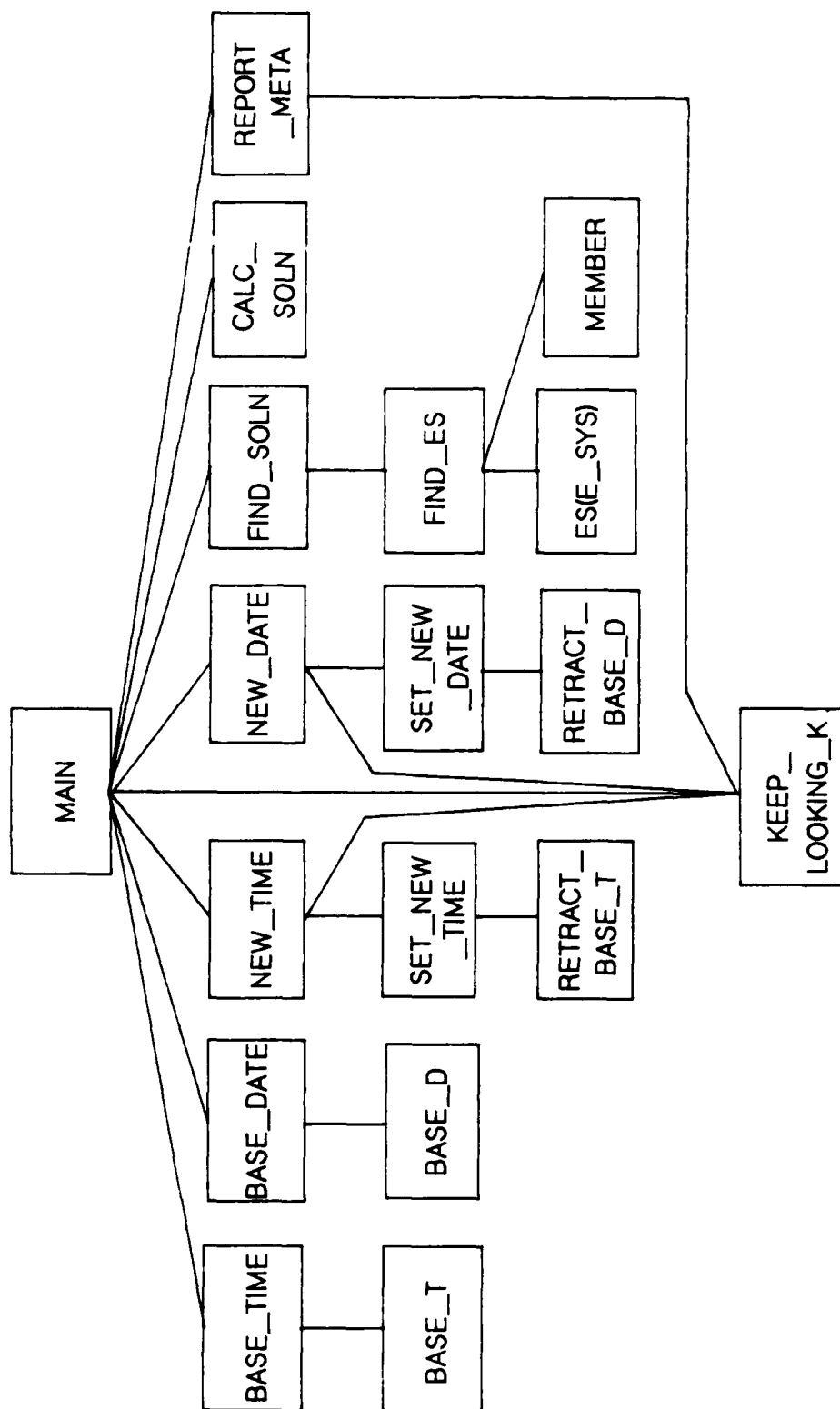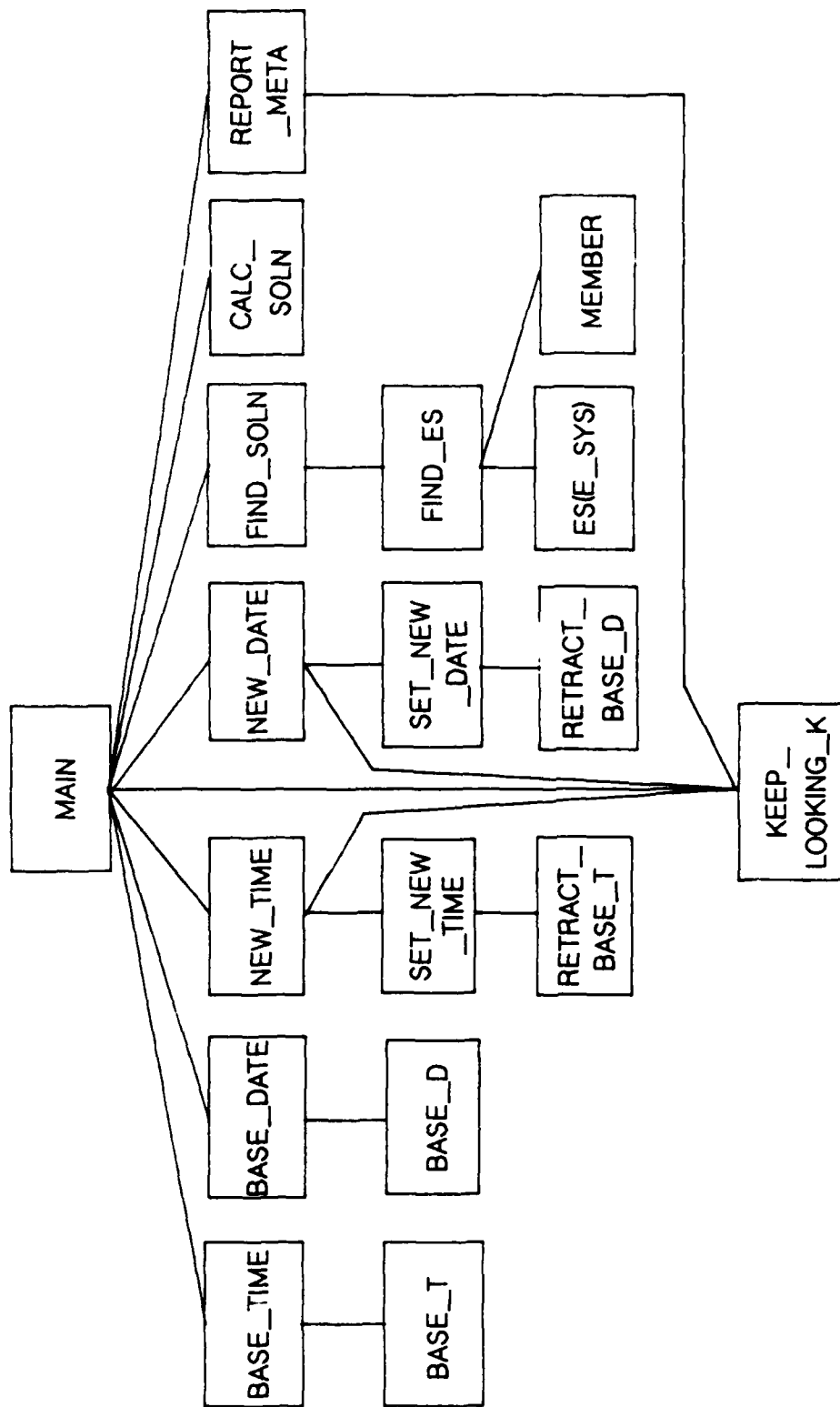
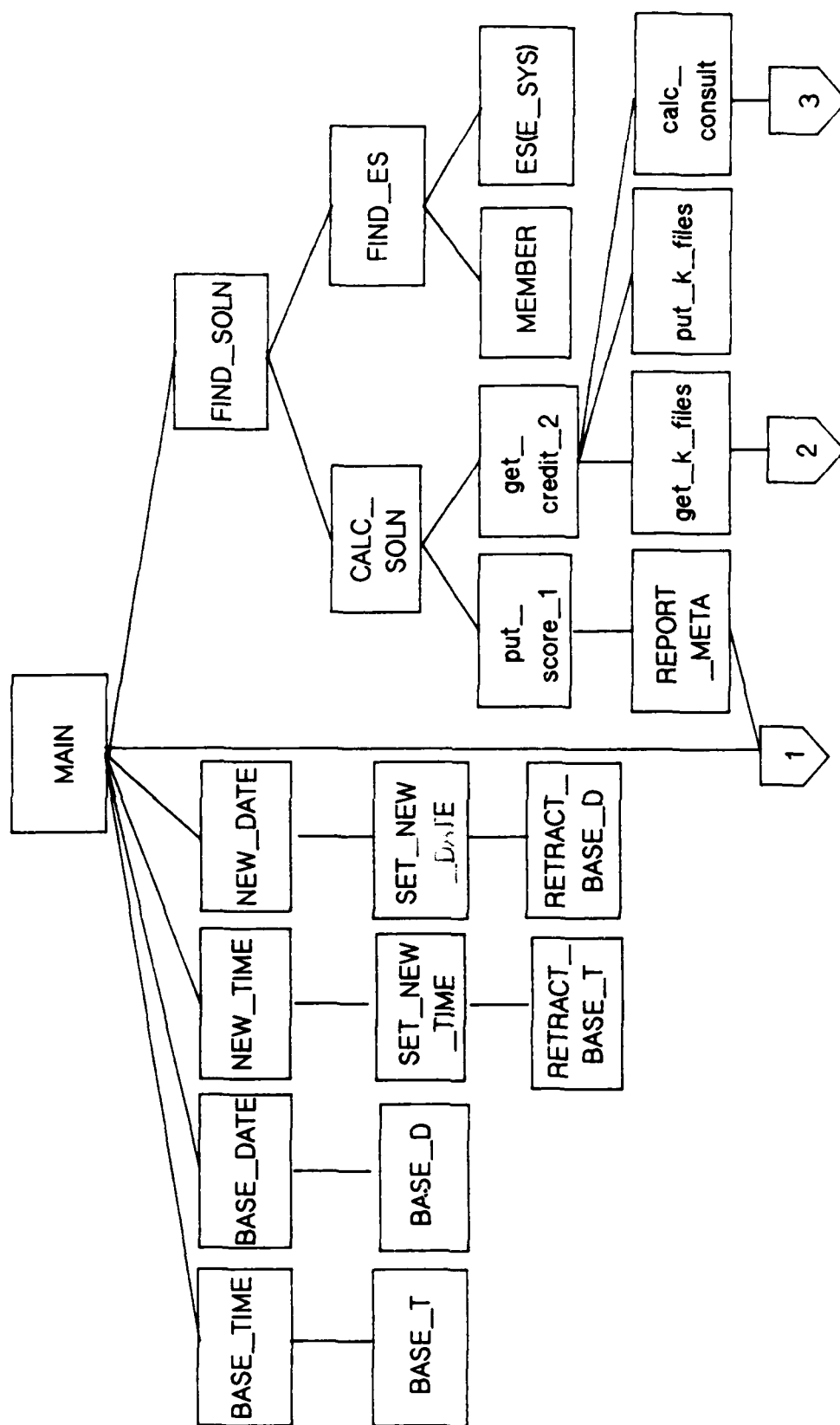Figure 3.4   CRIME_1

24

Figure 3.5   PSYCH_1

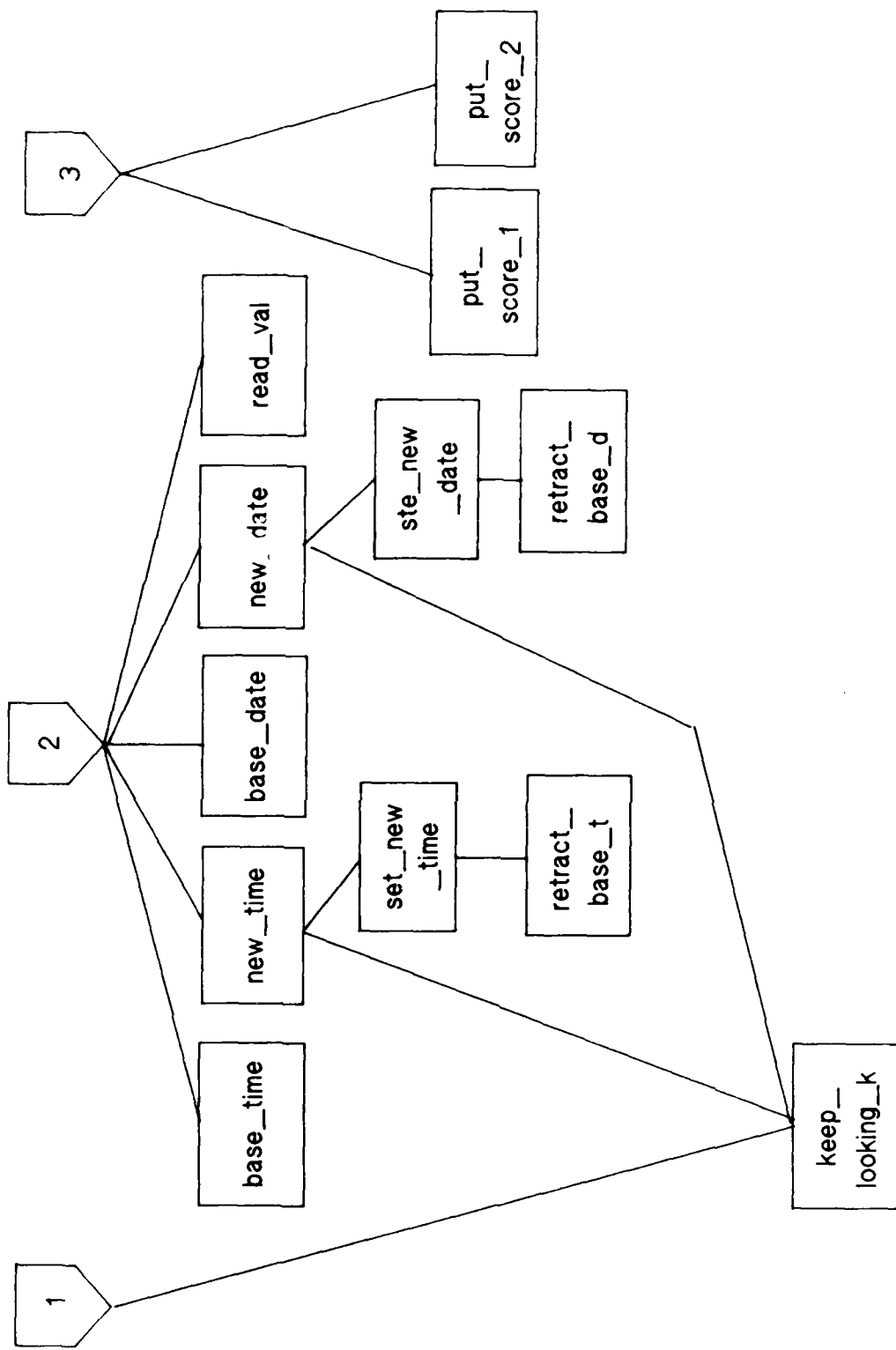Figure 3.6   CREDIT_3

26

Figure 3.7 CREDIT_1

27

Figure 3.7(cont.)

28

Figure 3.8   CREDIT_2

29
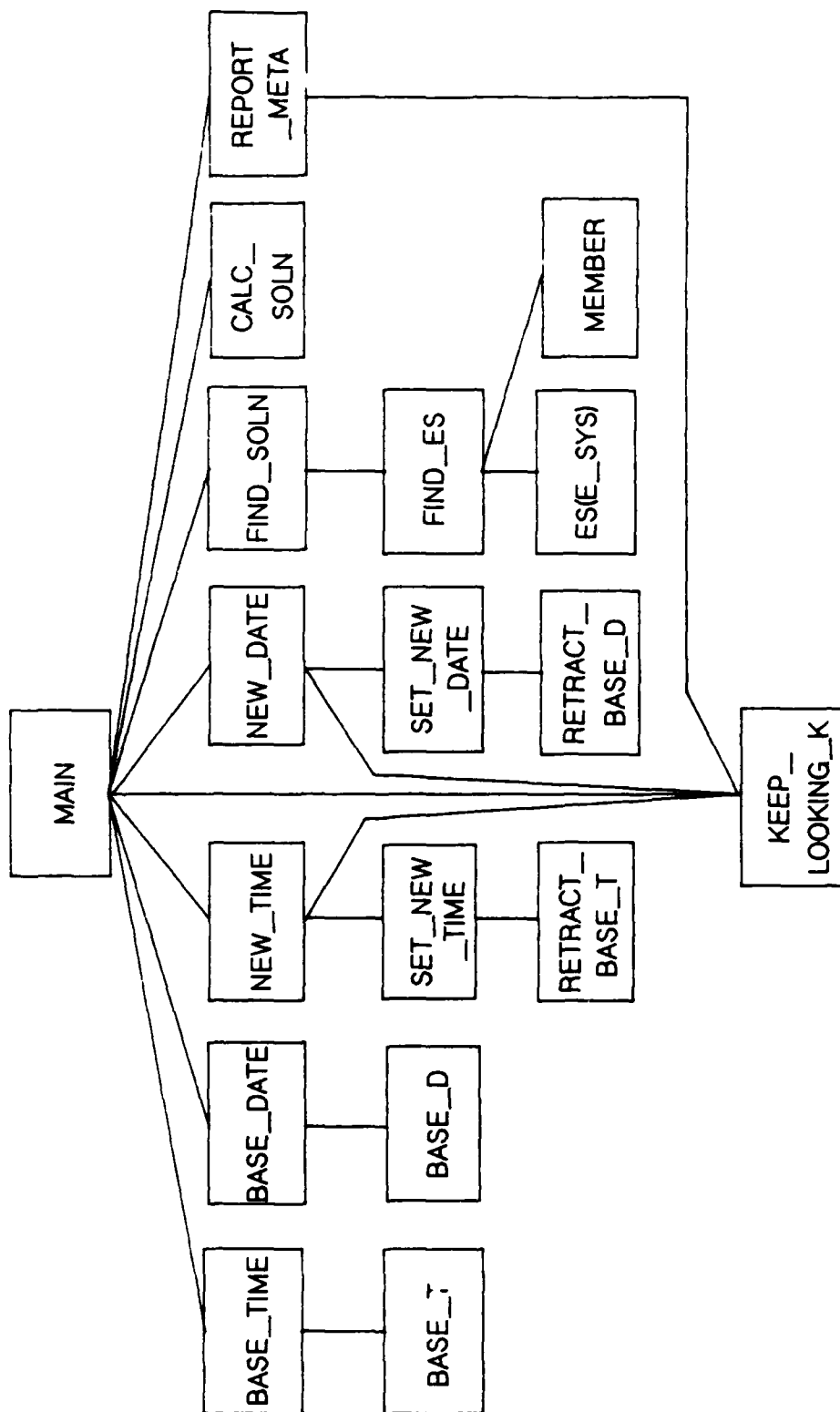
the peer level can be made in the same way as a vertical call.

The key, again, is modularization. The communication structure is generalized within the problem, such that it supports vertical and horizontal interfaces. The problem with implementation on non-distributed and non-multitasking microcomputer network is the necessity of writing to the blackboard. A problem-specific file, such as crime_1.rep, does not allow complete separation of problem and communication.

A better example of modularity is the consulting module in CREDIT_2, "calc_soln". For the purposes of the prototype, a resultant score is asserted rather than conducting a consultation of an actual database:

calc_soln:- asserta(score(160)).

This module is called from the main program module and is completely independent from all other code. By only asserting a score, simplicity of the model is maintained, and most importantly, the generic nature of this module is demonstrated. The string "calc_soln:-" can be followed by a call to any credit database. In fact, the predicate can be followed by any argument, however unrelated, as this module is not coupled to any other. Creating an open environment will maximize the ease with which even third party expert systems can be incorporated into the GES. (Silverman, 1986, p. 28)

The credit check may even be conducted by a separate expert system which is called by calc_soln. By doing so the solution of a single problem is further separated from the group communication structure.

GESP presents a basic architecture for supporting group expert systems. Although it was not possible to obtain the lowest degree of coupling and highest degree of problem independence within the constraints of the PC LAN, the advantages that can be derived from modular design have been

30

demonstrated. Through the use of structured modular design and functional independence at all three levels of modularity, it is possible to implement group expert systems on a PC network regardless of the nature of the expert systems.

IV. COMMUNICATION AND ALLOCATION CONSIDERATIONS

Once the decision has been made to employ a group expert system the question of where to locate it arises. Certainly there are external environmental factors affecting system location, such as organizational structure and politics. However, the focus here will be strictly on system issues for which there is an algorithmic solution. The question of optimum allocation strategy applies whether implementing a group expert system on an existing hardware suit or designing an entirely new system including hardware. The question to be answered here is, "What is the optimum allocation strategy on a network for a group expert system involving multiple expert systems"?

There is no good allocation strategy presently in use. Saj-nicole Joni, director of consulting services at Gold Hill Computers, Inc. in Cambridge, Massachusetts, has stated that there is no threshold number of rules or processing speed that can be used to determine where the expert system should reside (Williamson, 1987, p. 56). If Joni's statement is true, how then are systems to be allocated?

The key to answering the question of allocation is to think in terms of maximizing benefits and minimizing costs. Networked expert systems benefit from the modularity of distributed processing as do other distributed systems. Modular design and distribution achieve the greatest advantages over stand-alone systems. These benefits, as discussed in detail in previous chapters, include an increased problem scope and knowledge base, multiple knowledge domains, and the ability to physically distribute expertise.

The disadvantage of distributing expert systems is the same as that for any other type of system, and that is overhead cost. Overhead cost is incurred when an expert system executing on a processor makes a call to the operating

32

system, the database, or communicates with another expert system residing on a different processor.

Expert system overhead cost is analogous to interprocess communication, as discussed by Wesley W. Chu, et al, in that system performance can be maximized by minimizing system cost. Interprocessor communication in distributed systems is much like paging in memory systems. IPC can be increased to the point where thrashing takes place. The system becomes saturated by overhead, and performance is degraded. Chu suggests an integer programming approach to the problem of task allocation as a means of minimizing IPC and maximizing system performance. Chu's objective function (Chu and Holloway, 1980, p. 57) for minimizing IPC cost, in the general case, is as follows:

$$Cost(X) = \Sigma_k \Sigma_i [q_{ik} x_i k + \Sigma_i < k \Sigma_j < i (w v_i j d_k i x_i k x_j 1)]$$

Subject to:

$\Sigma s_i x_{ik} \leq R_k$, k=1,...n   memory constraint, and

$\Sigma u_i x_{ik} \leq T_k$, k=1,...n   real-time constraint.

Where:

q = processing cost

x = assignment of ES_i to node k, 0 or 1

w = normalization factor

v = volume of communication

d = distance between nodes.   (Chu and Holloway, 1980, p. 62)

As task allocation minimizes IPC in conventional distributed systems, the allocation of expert systems to nodes and the allocation of problems to expert systems minimizes IPC in the specific case of distributed expert systems. An integer programming technique similar to the one presented by Chu can be used to solve the expert system allocation problem. Processing costs and communication costs are calculated for every expert system at every node. Assignment of expert systems to nodes is based on minimizing

33

the objective function, and therefore, maximizing perfor ce. The system cost function for expert system s comprised of two cost components, processing cost and communication cost. One of the major tasks of using the integer programming model is the evaluation of the system processing costs. Studies of literature concerning task allocation in a distributed environment have suggested the following list of variables pertinent to system cost:

N = execution frequency of a module

AET = accumulative execution time

r_i = number of expert system rules

s_i = I/O and level of coupling among rules

MIL = machine language instructions   (Chu and Lan, 1984, p. 692)

$t_p(A)$ = processor turnaround time

$t(A)$ = task turnaround time   (Shen and Tsai, 1985, p. 198)

t_k = power of processor at the node

Other considerations:

Growth potential

Memory

The values of r_i and s_i can be found by using the A* algorithm, which employs a cost function and an evaluation function, as described later in this chapter. N is captured in s_i through the cost function, and therefore becomes statistically insignificant and can be dropped. AET, $t_p(A)$, and $t(A)$ are inversely proportional to t_k. AET is expressed in machine language instructions, MIL. Machine instructions vary from processor to processor. Measuring MIL across heterogeneous processors could require a significant normalization effort. Measuring execution cost with respect to the expert system rather than a specific processor's instruction set provides a uniform measurement, and MIL is r_i / t_k. Considerations such as growth and memory are determined by the power of the processor at the node, t_k. rocessing

cost is the execution time of an expert system on a processor. Execution time is a function of the number of rules fired to solve a given problem by an expert system, the level of coupling among the rules, the amount of I/O required, and the power of the CPU at the node. The following formula for processing cost was derived as part of this study:

$$q_{ik} = f(r_i, s_i, t_k)$$

Combining the cost function and the evaluation function with respect to the power of the processor yields the cost of execution, q_{ik}.

The cost of processing an expert system is directly proportional to the number of rules in that expert system. Any expert system will eventually test all its rules, even if the level of coupling among the rules is zero. Certain queries will find an immediate match, while others will be matched by the latter rules. On the average fifty per cent of the rules in a system will be fired. The cost of executing a system is driven up by the total number of rules, r_i.

$$q_{ik} \propto r_i$$

The level of coupling is a factor which weights the rules. Rules that are highly coupled, that is, one rule fires one or more successive rules, are more costly to execute. Rules which perform I/O functions utilize more system resources than those that do not. The number of rules is weighted by the level of coupling and amount of I/O, and therefore, r_i is multiplied by s_i. System cost is directly proportional to s_i.

$$q_{ik} \propto s_i \quad \text{and} \quad q_{ik} \propto r_i s_i$$

The relationship between execution cost and the processing power is straight forward. As the power of the CPU increases, ceteras paribus, the time of system execution decreases. System cost is inversely proportional to t_k.

$$q_{ik} \propto 1 / t_k$$

If expert system (i) is located on node (k), then the processing cost, q_a, of running problem (a) on expert system (i) located on processor (k) is q_{ik}.

$$q_{ik} = r_i \, s_i \, / \, t_k$$

r_i = number of rules in ES_i

s_i = I/O overhead and level of coupling among rules in ES_i

t_k = power of CPU at node k

For the general case, the total system processing cost for any expert system located on any node is captured in the formula:

$$\text{Processing cost} = \Sigma_k \Sigma_i (q_{ik} x_{ik})$$

where, q is the cost of processing expert system (i) on processor (k), for all i and k, and from the assignment matrix, x = 1 if expert system (i) is on processor (k) and 0 if it is not.

The number of rules fired and the level of coupling among the rules depends on the control structure of the expert system. Expert systems may use any of a number of control structures for search; depth-first, breadth-first, optimization, best first, branch-and-bound, or A* search. Depending on the type of problem and the control structure used, one can determine the state of search of a problem using tools such as evaluation functions or cost functions. When designing an expert system, these functions help to clarify which search structure is best for a given problem by identifying the number of states, levels of logic, which must be transversed in order to reach a goal state. An evaluation function can numerically represent the distance from the goal, at any state in the search.

The concept of measuring distance from the goal by number of states is also useful in determining processing cost. Each state transversed to reach the goal requires an additional number of rules fired. Different search structures will fire a different number of rules for a problem, but regardless of

36

the type of search, the distance from the goal can be measured. By using an evaluation function one can determine the number of states required to solve a problem by an expert system.

To find r_i and s_i, each state must be evaluated in terms of the number of rules fired, the level of coupling of the rules, and I/O overhead. Optimal path searches, of which there are two, are best suited to state evaluation in terms of processing costs. The branch-and-bound search employs an evaluation function, and, as do all evaluation functions, it measures cost in terms of distance to the goal. This "strategy may jump around among states ... but it has a nice property: the first path to the goal is guaranteed to be the lowest-cost path to the goal" (Rowe, Ch. 9 p. 9). Evaluation functions, however, account only for distance in terms of states. With respect to expert systems, they do not account for the number of rules fired in a given state or the cost of executing operators within that state. Operators which require I/O have a higher system cost associated with them due to the difference in speed of CPU processing versus database access and calls to the operating system. The I/O operators can be converted to standard work units for system design and tuning purposes, as will be discussed later.

The A* (A star) search employees both an evaluation function, to account for search distance, and a cost function to assign values to the states based on the cost of the operators within the states.

Combining the values of the evaluation and cost functions provides a method of measuring r_i and s_i. Both functions should use the same unit of measure. The processing cost function measures the execution time of equivalent instructions on a processor in terms of rules fired and I/O requirements. Therefore, if the evaluation gives the distance to the goal, the number of states to the goal, and the cost function assigns a value to each state, based on rules fired

and I/O overhead the combination of the two yields the lowest cost of processing a given problem on a node.

A specific example of how this is done can be demonstrated using the prototype distributed expert system. The processing cost for expert system CRIME_1 to solve problem (a) is calculated as follows:

$$q_{ik} = r_i \, s_i \, / \, t_k$$

Using A* search to trace r_i and s_i through each state within an expert system is a variation of the state-space search method, applied specifically to expert system allocation. The purpose of this application of state-space search is not to find the optimal weak homomorphism (Shen and Tsai, 1985, p. 200) but to measure the processing cost q_{ik} of a possible system allocation or that of an existing system. There are significant system design implications for the A* algorithm which will be discussed in the concluding chapter.

State-space search allows one to find the cost of the path to the goal for a single problem. Applying the integer programming method to the result of the A* algorithm sums the processing costs of all problems that can be executed by all expert systems at a given node to determine the system processing cost for that node. The system objective function can now optimize processing cost for all problems over all nodes. It is now possible to allocate systems to nodes based on minimizing processing cost, however, an optimum allocation strategy must also consider communication costs.

Communications costs for the expert are decomposed into volume and cost per unit. Volume is the amount of communication required between expert systems, $C_i j$, for expert systems (i and j). Cost per unit volume, C_{kl}, is the cost of communicating between two processors, k and l.

Volume of communication is not problem dependent when allocating expert systems to nodes. The primary consideration is the volume of communication necessary to create the

interface between expert systems regardless of the problem to be solved. Volume becomes problem dependent at the task allocation level. That is, how to decide which problem should be run on which expert system, based on cost verses solution payback. Volume becomes problem dependent at the system level only if the system solves a single problem or if task allocation is static. Both of these situations are uninteresting from an interactive GDSS point of view and will not be discussed further. A static allocation problem can be solved using the dynamic allocation model.

The formula for determining communication cost is developed by combining volume of communication with cost per unit volume with respect to the assignment matrix:

Communication cost = $\Sigma_k \Sigma_l (C_{ij} C_{kl} x_{ik} x_{jl})$

Other variables for communication cost suggested by previous studies:

V = average number of words communicated

M-F = number of updates

L = average number of words per update   (Chu and Lan, 1984, p. 692)

V and L are captured by C_{ij}, and M-F is represented in C_{kl}.

Total system cost is realized by combining processing and communication costs:

Total system cost = $\Sigma_k \Sigma_l [q_{ij} x_{ij} + \Sigma_k \Sigma_l (C_{ij} C_{kl} x_{ik} x_{jl})]$

Optimum allocation can be realized by minimizing total cost subject to:

real-time constraint:

$\Sigma (u_1 x_i k \leq T_k)$, k = 1,...,n

where, u = processing time required by ES (i)

T = time constraint for processing ES (i) on node (k) and,

memory constraint:

$\Sigma (s_i x_{ik} \leq R_k)$, k = 1,...,n

where, s = memory required by ES(i)

r = maximum memory in node k.

The expert system allocation formula is similar to the general format for distributed systems derived by Chu (1980):

$$\text{Cost}(X) = \Sigma_k \Sigma_i [q_{ik} + x_{ik} + \Sigma_{i<k} \Sigma_{j<l} (wv_{ij} d_{ik} x_{ik} x_{jl})].$$

For the model system implemented on the LAN, distance (d) is considered constant and insignificant to communication cost and will be ignored. As well, the normalization factor (w) is unnecessary, as units of measure are consistent throughout the model.

To look at a specific problem on the model, suppose the problem is problem "a". There is some probability associated with each expert system in the model which is the likelihood that the system will be called to solve problem a. There is likewise some probability associated with each expert system for every problem solvable by the distributed expert system model. The probability that a system is called to solve a problem can be applied to the processing costs and communication costs of executing that problem on the individual expert system. Summing the costs of every expert system called to solve the problem solution yields the cost of solving that individual problem on the distributed system. Summing the costs of all the individual problems on the system determines the total system cost.

$$\text{Cost} = \Sigma_k \Sigma_i [q_{ik} x_{ik} + \Sigma_k \Sigma_l (C_{ij} C_{kl} x_{ik} x_{jl})]$$

The distributed expert system model has four expert systems; META, CRIME_1, CREDIT_1, and CREDIT_2, to be referred to in the equation as ES1, ES2, ES3, and ES4 respectively. The probability that expert system 1 is called is P_{ES1}. the probability that problem a is executed is P_a. As discussed in the previous chapter, all nodes communicate via the blackboard, which will be represented as node 5. During each call, a node both writes to and reads from the blackboard. Therefore, communication costs between nodes 1 and 2 via node 5 is $2C_{15} + 2C_{25}$.

40

The problem of optimizing the allocation of problems on the system can be solved by the following equation:

$$\text{Cost} = PE_{S1}(P_a q_1 + P_b q_1 + P_c q_1) +$$
$$PES_2[Pa(q_2 + 2C_{15} + 2C_{25}) + P_b(q_2 + 2C_{15} + 2C_{25}) +$$
$$P_c(q_2 + 2C_{15} + 2C_{25})] +$$
$$PES_3\{P_a[q_3 + 2C_{15} + 2C_{35} + P_{ES4}(q_4 + 2C_{35} + 2C_{45})] +$$
$$P_b[q_3 + 2C_{15} + 2C_{35} + P_{ES4}(q_4 + 2C_{35} + 2C_{45})] +$$
$$P_c[q_3 + 2C_{15} + 2C_{35} + P_{ES4}(q_4 + 2C_{35} + 2C_{45})]\}$$

The problem of optimizing the allocation of expert systems to nodes is more complicated when considering the processing and communication costs for the entire set of problems to be solved by the expert systems. The above equation must now be solved for each expert system on each node, and the equation is expanded as follows:

$$\text{Cost} = P_{ES1}(P_a q_{11} x_1 + P_b q_{11} x_1 + P_c q_{11} x_1 + P_a q_{12} x_2 + P_b q_{12} x_2 +$$
$$P_c q_{12} x_2 + pa q_{13} x_3 + P_b q_{13} x_3 + P_c q_{13} x_3 + P_a q_{14} x_4 +$$
$$P_b q_{14} x_4 + P_c q_{14} x_4) +$$
$$P_{ES2}[P_a(q_{21} x_2 + q_{22} x_2 + q_{23} x_2 + q_{24} x_2 + 2C_{15} + 2C_{25}) +$$
$$P_b(q_{21} x_2 + q_{22} x2 + q_{23} x_2 + q_{24} x_2 + 2C_{15} + 2C_{25}) +$$
$$P_c(q_{21} x_2 + q_{22} x_2 + q_{23} x_2 + q_{24} x_2 + 2C_{15} + 2C_{25}) +$$
$$P_{ES3}\{P_a[q_{31} x3 + q_{32} x_3 + q_{33} x_3 + q_{34} x_3 + 2C_{15} + 2C_{35} +$$
$$P_{ES4}(q_{41} x_4 + q_{42} x_4 + q_{43} x_4 + q44 x_4 + 2C_{35}$$
$$2C_{45})] + P_b[q_{31} x_3 + q_{32} x_3 + q_{33} x_3 + q_{34} x_3 +$$
$$2C_{15} + 2C_{35} + P_{ES4}(q_{41} x_4 + q_{42} x_4 + q_{43} x_4 +$$
$$q_{44} x_4 + 2C_{35} + 2C_{45})] + P_c[q_{31} x_3 + q_{32} x_3 +$$
$$q_{33} x_3 + q_{34} x_3 + 2C^1{}_5 + 2C_{35} +$$
$$P_{ES4}(q_{41} x_4 + q_{42} x_4 + q_{43} x_4 + q_{44} x_4 + 2C_{35} +$$
$$2C_{45})]\}$$

Assume ES_1 is located on node 1, ES_2 on node 2, ES_3 on node 3, and ES_4 on node 4 with node 5 as the blackboard. Problem "a" will be solved, and all expert systems are called to reach a solution.

$$\text{Cost} = q_{11} + (q_{22} + 2C_{15} + 2C_{25}) + [q_{33} + 2C_{15} + 2C_{35} +$$
$$(q_{44} + 2C_{35} + 2C_{45})]$$

41

For problem (a) on q_{11} there are 83 rules. These rules
perform I/O functions or are coupled to other rules 51
times. The power of the 8086 processor on the IBM PC LAN is
4.77MHz.

$$q_{11} = r_i \, s_i \, / \, t_k$$
$$= 83 \times 51 \, / \, 4.77 = 4233 \, / \, 4.77 = 887.42$$
+

$$q_{22} = 40 \times 26 \, / \, 4.77 = 1040 \, / \, 4.77 = 218.03$$
+

$$2C_{15} = 2(C_{ij}C_{kl})$$
$$= 2(2 \times 2) \qquad = 8$$
+

$$2C_{25} = 2(1 \times 2) \qquad = 4$$
+

$$q_{33} = 80 \times 46 \, / \, 4.77 = 3680 \, / \, 4.77 = 771.49$$
+

$$2C_{35} = 2(1 \times 1) \qquad = 2$$
+

$$q_{44} = 40 \times 26 \, / \, 4.77 = 1040 \, / \, 4.77 = 218.03$$
+

$$2C_{45} = 2(1 \times 1) \qquad = 2$$

$$\text{Cost} = 887.42 + 218.03 + 8 + 4 + 771.49 + 2 + 218.03 + 2$$
$$= 2110.97$$

For this example the level of coupling was determined by
the number of times a rule fires additional rules or invokes
an I/O function. The module "get_credit_1" has a coupling
value of 6.

```
    get_credit_1:-
*         directory('c:\credit_1.rep',_,_,T,D,_),
#         base_time_c,
#         base_date_c,
```

42

```
#          new_time_c(T),
#          new_date_c(D),
           asserta(base_d_c(D)),
           asserta(base_t_c(T)),
*          shell('copy c: credit_1.rep'),
           open(H,'credit_1.rep,r),
           read_val(H),
           close(H).
```

There are two calls to the operating system (*) and four additional rules fired (#). It is assumed that vertical communications cost twice as much as peer-to-peer communications and that META is sending twice the message volume as the other systems.

The resultant value is a magnitude. It can applied across heterogeneous processors, with regard to the value of q, and to dissimilar networks, with regard to C. The value of q may be converted to equivalent instructions on a specific machine and the time of execution on that machine. C represents the number of messages sent and the value of the data. The end result can be measured in terms of the standard work unit (SWU). In terms of the interactive environment, the SWU is the single most important concept. Applications built within parameters of the standard work unit minimize critical resources used for execution and uniformly apply the discipline of controlling resource consumption across all applications within the system. (Inmon, 1983, p. 75)

A magnitude expressed in terms of performance constraints, such as the SWU, provides an optimal means of allocating expert systems. A threshold number of rules or processor speed is not necessary, as the number of rules, level of coupling among rules, and power of the processor are contained in the objective function.

Through the use of distributed expert systems, it is possible to solve problems previously thought to be too broad

for expert system application. A large knowledge domain may be partitioned into distinct areas of expertise which are incorporated into separate expert systems. Overall problem management is performed by the meta expert system. The system allocation function can be applied to any specific configuration to determine to determine where these expert systems should reside. If the problem can be solved by more than one expert system, or if it may be necessary to solve only part of a problem, the task allocation function can determine which expert system should be called. Integer programming allows one to determine the optimal use of the distributed system. Programming the selection criteria into the meta system automates the optimization process of task allocation.

V.   CONCLUSIONS AND RECOMMENDATIONS FOR FUTURE RESEARCH

A. CONCLUSIONS

This thesis has shown that implementation of distributed expert systems is possible on a Local Area Network. By distributing expertise within group expert systems, it is feasible to solve large problems using artificial intelligence applications on PC's.

If the problem domain is one that is factorable, then independent expert systems can solve portions of the problem. The domain must be factored into discrete areas of expertise which can be distributed across the nodes of a network. Solutions from these expert systems can be synthesized by a single system to solve the meta problem.

The general architecture for group expert systems consists of a communication structure, a meta expert system, and consultant expert systems. The communication mechanism should be independent of the specific function of the expert systems in the group. It is merely a vehicle to invoke consultant expert systems, pass problem information and solutions, and perform I/O functions. Vertical communications between the meta and consultant systems and horizontal peer-to-peer communications between consultant systems must be supported.

The meta expert system manages the problem solution. It identifies, validates, and decomposes the problem. The meta system determines problem-domain and domain-domain relationships and devolves the problem to the appropriate expert systems. When given a choice of consultant expert systems within the same domain, it selects a system based on a cost verses problem pay-back criteria. Inputs from consultant expert systems are then synthesized into a problem solution by the meta system.

The consultant expert systems read and validate the problem input. These expert systems may consult with each

other on a peer-to-peer basis during the formulation of a problem solution within its own domain. They then communicate vertically their individual solutions to the meta system. This general architecture supports the solution of any problem for which the knowledge domain is factorable. The architecture is not affected by the specific nature of the problem.

The optimal location for an expert system on the network can be determined by minimizing system cost. System cost has two significant components, processing cost and communication cost. Communication cost is straight-forward. It is a factor of the volume of communication, C_{ij}, multiplied by the cost per unit of volume, C_{kl}.

Processing cost, q_{ik}, for expert systems was found to be a function of the number of rules in an expert system, r_i, the level of coupling among the rules, s_i, and the power of the CPU at the node, t_k. The relationship is the formula:

$$q_{ik} = r_i \, s_i \, / \, t_k$$

Using integer programming, minimum total system cost can be obtained by minimizing the following objective function:

$$\text{Cost} = \Sigma_k \Sigma_l [q_{ij} x_{ij} + \Sigma_k \Sigma_l (C_{ij} C_{kl} x_{ik} x_{jl})]$$

The end result is a magnitude which can be applied to expert systems across heterogeneous processors. Optimum distribution is important to both system design and tuning.

2. FUTURE RESEARCH

The group expert system developed in this thesis is a pioneering model used to prove the possibility of implementing group expert systems on a PC LAN, to demonstrate a method of system allocation, and to propose an architecture for group expert systems. This unique prototype is limited by the purposes for which it was created and the environment in which it was implemented.

Expansion of the model itself offers several directions for future research. To make the GESP system practical it

should be implemented on a distributed operating system. The advent of multitasking microcomputer operating systems, such as OS/2, will allow calling procedures for consultant expert systems to be efficient. The processor need not be dedicated to a polling procedure to receive information, and the use of a blackboard can be eliminated.

In a distributed environment the communication architecture can be totally generic. There will be no need to communicate by sending problem-specific files. The goal is to create a communication ES which is completely problem-independent, achieving the first level of modularity. Adhering to a modular communication structure will allow multiple group expert systems to co-exist on the network and be controlled by a single communication expert system, CES. The CES will know of all other expert systems in the GES and of all problems the GES is capable of solving. The generic CES can be located at any or all nodes in the network. A user can then enter the GES from any node and solve any problem in the GES. To make this extension from the GESP system one must a CES separate from the meta ES. It is the meta ES that achieves the second level of modularity, the factoring of the problem domain, and synthesizes the problem solution.

The ultimate extension of this study would be the implementation of a group expert system on a Wide Area Network. Aside from the obvious differences in communication requirements, the implications for GES and individual expert system architecture will need to be studied. Wide Area Network GES offer tremendous potential support for both the strategic C^3 environment and managerial decision making in the private sector.

APPENDIX

SOURCE CODE FOR GESP

```
/*      META.ARI      */
/*   database  */
base_t_b(time(0,0,0,0)).
base_d_b(date(0,0,0)).
base_t_c(time(0,0,0,0)).
base_d_c(date(0,0,0)).
base_t_d(time(0,0,0,0)).
base_d_d(date(0,0,0)).
base_t_e(time(0,0,0,0)).
base_d_e(date(0,0,0)).
base_t_f(time(0,0,0,0)).
base_d_f(date(0,0,0)).


member(Probs,[a,b,c,d,e,f]).


consult_es(a,[crime_1,credit_1]).
consult_es(b,[crime_1]).
consult_es(c,[credit_1]).
consult_es(d,[psych_1,credit_3]).
consult_es(e,[crime_1,credit_1,psych_1]).
consult_es(f,[credit_3,crime_1,psych_1]).


/*    rules    */


:- public main/0.
main:-
    prob_read,
    prob_features,
    prob_soln,
    find_soln,
    retract(prob_stmt(Prob_Stmt)).  /* Do this last. */
```

```prolog
prob_read:-
    nl,
    write('input problem statement:'),
    read(Prob_Stmt),
    asserta(prob_stmt(Prob_Stmt)).


prob_features:-
    call(prob_stmt(Prob_Stmt)),
    ifthenelse(member(Prob_Stmt,Probs),meta_a,exit).


member(X,[X|_]).
member(X,[_|Y]):- member(X,Y).


meta_a:-
    subfeatures(E_Sys),
    put_k_files(E_Sys),
    write('kfile written').


exit:-
    nl,write('Could not interpret').


put_k_files(E_Sys):-
    create(H3,'k_file.inp'),
    write(H3,'es('),write(H3,E_Sys),write(H3,').'),nl(H3),
    close(H3),
    shell('copy k_file.inp c:').


subfeatures(E_Sys):-
    call(prob_stmt(Prob_Stmt)),
    consult_es(Prob_Stmt,E_Sys).


prob_soln:-
    call(prob_stmt(Prob_Stmt)),
    ifthen(Prob_Stmt=a,report_a);
    ifthen(Prob_Stmt=b,report_b);
```

```prolog
        ifthen(Prob_Stmt=c,report_c);
        ifthen(Prob_Stmt=d,report_d);
        ifthen(Prob_Stmt=e,report_e);
        ifthen(Prob_Stmt=f,report_f).


report_a:-
     get_crime_1,
     get_credit_1.


report_b:-
     get_crime_1.


report_c:-
     get_credit_1.


report_d:-
     get_psych_1,
     get_credit_3.


report_e:-
     get_crime_1,
     get_credit_1.


report_f:-
     get_crime_1,
     get_credit_3,
     get_psych_1.


get_crime_1:-
     directory('c:\crime_1.rep',_,_,T,D,_),
     base_time_b,
     base_date_b,
     new_time_b(T),
     new_date_b(D),
     asserta(base_d_b(D)),
```

```
        asserta(base_t_b(T)),
        shell('copy c: crime_1.rep'),
        open(H,'crime_1.rep',r),
        read_val(H),
        close(H).


get_credit_1:-
        directory('c:\credit_1.rep',_,_,T,D,_),
        base_time_c,
        base_date_c,
        new_time_c(T),
        new_date_c(D),
        asserta(base_d_c(D)),
        asserta(base_t_c(T)),
        shell('copy c: credit_1.rep'),
        open(H,'credit_1.rep',r),
        read_val(H),
        close(H).


get_psych_1:-
        directory('c:\psych_1.rep',_,_,T,D,_),
        base_time_d,
        base_date_d,
        new_time_d(T),
        new_date_d(D),
        asserta(base_d_d(D)),
        asserta(base_t_d(T)),
        shell('copy c: psych_1.rep'),
        open(H,'psych_1.rep',r),
        read_val(H),
        close(H).


get_credit_2:-
        directory('c:\credit_2.rep',_,_,T,D,_),
        base_time_e,
```

```
        base_date_e,
        new_time_e(T),
        new_date_e(D),
        asserta(base_d_e(D)),
        asserta(base_t_e(T)),
        shell('copy c: credit_2.rep'),
        open(H,'credit_2.rep',r),
        read_val(H),
        close(H).

get_credit_3:-
        directory('c:\credit_3.rep',_,_,T,D,_),
        base_time_f,
        base_date_f,
        new_time_f(T),
        new_date_f(D),
        asserta(base_d_f(D)),
        asserta(base_t_f(T)),
        shell('copy c: credit_3.rep'),
        open(H,'credit_3.rep',r),
        read_val(H),
        close(H).

base_time_b:- base_t_b(time(W,X,Y,Z)).

base_date_b:- base_d_b(date(X,Y,Z)).

base_time_c:- base_t_c(time(W,X,Y,Z)).

base_date_c:- base_d_c(date(X,Y,Z)).

base_time_d:- base_t_d(time(W,X,Y,Z)).

base_date_d:- base_d_d(date(X,Y,Z)).
```

```
base_time_e:- base_t_e(time(W,X,Y,Z)).


base_date_e:- base_d_e(date(X,Y,Z)).


base_time_f:- base_t_f(time(W,X,Y,Z)).


base_date_f:- base_d_f(date(X,Y,Z)).


new_time_b(T):-
    base_t_b(B_T),
    ifthenelse(T=B_T,keep_looking_k,set_new_time).


new_date_b(D):-
    base_d_b(B_D),
    ifthenelse(D=B_D,keep_looking_k,set_new_date).


new_time_c(T):-
    base_t_c(B_T),
    ifthenelse(T=B_T,keep_looking_k,set_new_time).


new_date_c(D):-
    base_d_c(B_D),
    ifthenelse(D=B_D,keep_looking_k,set_new_date).


new_time_d(T):-
    base_t_d(B_T),
    ifthenelse(T=B_T,keep_looking_k,set_new_time).


new_date_d(D):-
    base_d_d(B_D),
    ifthenelse(D=B_D,keep_looking_k,set_new_date).


new_time_e(T):-
    base_t_e(B_T),
    ifthenelse(T=B_T,keep_looking_k,set_new_time).
```

```
new_date_e(D):-
    base_d_e(B_D),
    ifthenelse(D=B_D,keep_looking_k,set_new_date).


new_time_f(T):-
    base_t_f(B_T),
    ifthenelse(T=B_T,keep_looking_k,set_new_time).


new_date_f(D):-
    base_d_f(B_D),
    ifthenelse(D=B_D,keep_looking_k,set_new_date).


set_new_time:- retract(base_t_b(time(W,X,Y,Z))).


set_new_date:- retract(base_d_b(date(X,Y,Z))).


set_new_time:- retract(base_t_c(time(W,X,Y,Z))).


set_new_date:- retract(base_d_c(date(X,Y,Z))).


set_new_time:- retract(base_t_d(time(W,X,Y,Z))).


set_new_date:- retract(base_d_d(date(X,Y,Z))).


set_new_time:- retract(base_t_e(time(W,X,Y,Z))).


set_new_date:- retract(base_d_e(date(X,Y,Z))).


set_new_time:- retract(base_t_f(time(W,X,Y,Z))).


set_new_date:- retract(base_d_f(time(X,Y,Z))).


read_val(H):-
    repeat,
```

```prolog
        read(H,T),
        asserta(T),
        recordz(val,T,_),
        T=end_of_file.

find_soln:-
    add.

add:-
    findall(X,score(X),L),
    L=[L1,L2],
    Total is L1+L2,
    write(Total),
    ifthen(Total>300,action_1);
    ifthen(L1>160,action_1);
    ifthen(L2>180,action_1);
    ifthen(Total=<300,action_2);
    ifthen(L1=<160,action_2);
    ifthen(L2=<180,action_2).

action_1:-
    write('Further investigation required.').

action_2:-
    write('Subject requires no further investigation.').

keep_looking_k:-
    prob_soln.

/*   end META.ARI   */
```

```
/*    CRIME_1.ARI    */
/*    database    */
    base_t(time(0,0,0,0)).
    base_d(date(0,0,0)).


/*    rules    */


:- public main/0.
main:-
    directory('c:\k_file.inp',_,_,T,D,_),
    base_time,
    base_date,
    new_time(T),
    new_date(D),
    asserta(base_d(D)),
    asserta(base_t(T)),
    shell('copy c:k_file.inp'),
    open(H,'k_file.inp',r),
    read_val(H),
    close(H),
  (( find_soln,
    calc_soln,
    report_meta ) ;
    keep_looking_k).


base_time:- base_t(time(W,X,Y,Z)).


base_date:- base_d(date(X,Y,Z)).


new_time(T):-
    base_t(B_T),
    ifthenelse(T=B_T,keep_looking_k,set_new_time).


new_date(D):-
    base_d(B_D),
```

```prolog
            ifthenelse(D=B_D,keep_looking_k,set_new_date).


    set_new_time:- retract(base_t(time(W,X,Y,Z))).


    set_new_date:- retract(base_d(date(X,Y,Z))).


    read_val(H):-
        repeat,
        read(H,T),
        asserta(T),
        recordz(val,T,_),
        T=end_of_file.


    find_soln:-
        find_es(E_Sys).


    find_es(E_Sys):-
        call(es(E_Sys)),
        member(crime_1,E_Sys).


    calc_soln:- asserta(score(180)).


    /*  A score of 180 is asserted for demo purposes.       */
    /*  The actual criminal database would be called here.  */


    member(X,[X|_]).
    member(X,[_|Y]):- member(X,Y).


    report_meta:-
        call(score(X)),
        create(H3,'crime_1.rep'),
        write(H3,'score('),write(H3,X),write(H3,').'),nl(H3),
        close(H3),
        shell('copy crime_1.rep c:'),
        keep_looking_k.
```

```
keep_looking_k:-
     main.

/*     End CRIME_1.ARI     */
```

```
/*      CREDIT_1.ARI     */
/*   database      */
    base_t(time(0,0,0,0)).
    base_d(date(0,0,0)).
    base_t_c(time(0,0,0,0)).
    base_d_c(date(0,0,0)).


/*    rules      */

:- public main/0.
main:-
     directory('c:\k_file.inp',_,_,T,D,_),
     base_time,
     base_date,
     new_time(T),
     new_date(D),
     asserta(base_d(D)),
     asserta(base_t(T)),
     shell('copy c:k_file.inp'),
     open(H,'k_file.inp',r),
     read_val(H),
     close(H),
  (( find_soln,
     report_meta ) ;
     keep_looking_k).

base_time:- base_t(time(W,X,Y,Z)).


base_date:- base_d(date(X,Y,Z)).


new_time(T):-
     base_t(B_T),
     ifthenelse(T=B_T,keep_looking_k,set_new_time).


new_date(D):-
```

59

```prolog
        base_d(B_D),
        ifthenelse(D=B_D,keep_looking_k,set_new_date).


set_new_time:- retract(base_t(time(W,X,Y,Z))).


set_new_date:- retract(base_d(date(X,Y,Z))).


read_val(H):-
        repeat,
        read(H,T),
        asserta(T),
        recordz(val,T,_),
        T=end_of_file.


find_soln:-
        find_es(E_Sys),
        calc_soln.


find_es(E_Sys):-
        call(es(E_Sys)),
        member(credit_1,E_Sys).


member(X,[X|_]).
member(X,[_|Y]):- member(X,Y).


calc_soln:-
 /* do credit calc & insert X for 161 */
        asserta(score_1(161)),
        call(score_1(161)),
        ifthenelse(score_1(161)>160,get_credit_2,put_score_1).


put_score_1:-
/*      call(score_1(161),
        S=(161),
        asserta(score(161)), */
```

```
          report_meta.

get_credit_2:-
     put_k_files,
     write('kfile written').
     get_k_files,
     calc_consult.

calc_consult:-
     call(score_1(X)),
     call(score_2(Y)),
     ifthenelse(score_2(Y)>score_1(X),put_score_-
2,put_score_1).

put_k_files:-
     create(H4,'credit_1.inp'),
     write(H4,'es('),write(H4,credit_2),write(H4,').'),nl(H-
4),
     close(H4),
     shell('copy credit_1.inp c:').

get_k_files:-
     directory('c:\credit_2.rep',_,_,T,D,_),
     base_time_c,
     base_date_c,
     new_time_c(T),
     new_date_c(D),
     asserta(base_d_c(D)),
     asserta(base_t_c(T)),
     shell('copy c: credit_2.rep'),
     open(H,'credit_2.rep',r),
     read_val_c(H),
     close(H).

base_time_c:- base_t_c(time(W,X,Y,Z)).
```

```
base_date_c:- base_d_c(date(X,Y,Z)).


new_time_c(T):-
    base_t_c(B_T),
    ifthenelse(T=B_T,keep_looking_k,set_new_time_c).


new_date_c(D):-
    base_d_c(B_D),
    ifthenelse(D=B_D,keep_looking_k,set_new_date_c).


set_new_time_c:- retract(base_t_c(time(W,X,Y,Z))).


set_new_date_c:- retract(base_d_c(date(X,Y,Z))).


read_val_c(H):-
    repeat,
    read(H,T),
    asserta(T),
    recordz(val,T,_),
    T=end_of_file.


report_meta:-
    call(score(X)),
    create(H3,'credit_1.rep'),
    write(H3,'score('),write(H3,X),write(H3,').'),nl(H3),
    close(H3),
    shell('copy credit_1.rep c:'),
    keep_looking_k.


keep_looking_k:-
    main.


/*  end CREDIT_1.ARI  */
```

```
/*      CREDIT_2.ARI      */
/*    database      */
    base_t(time(0,0,0,0)).
    base_d(date(0,0,0)).


/*    rules      */


:- public main/0.
main:-
    directory('c:\credit_1.inp',_,_,T,D,_),
    base_time,
    base_date,
    new_time(T),
    new_date(D),
    asserta(base_d(D)),
    asserta(base_t(T)),
    shell('copy c:credit_1.inp'),
    open(H,'credit_1.inp',r),
    read_val(H),
    close(H),
  (( find_soln,
    calc_soln,
    report_meta ) ;
    keep_looking_k).


base_time:- base_t(time(W,X,Y,Z)).


base_date:- base_d(date(X,Y,Z)).


new_time(T):-
    base_t(B_T),
    ifthenelse(T=B_T,keep_looking_k,set_new_time).


new_date(D):-
    base_d(B_D),
```

63

```prolog
        ifthenelse(D=B_D,keep_looking_k,set_new_date).


set_new_time:- retract(base_t(time(W,X,Y,Z))).


set_new_date:- retract(base_d(date(X,Y,Z))).


read_val(H):-
    repeat,
    read(H,T),
    asserta(T),
    recordz(val,T,_),
    T=end_of_file.


find_soln:-
    find_es(E_Sys).


find_es(E_Sys):-
    call(es(E_Sys)),
    member(credit_1,E_Sys).


calc_soln:-  asserta(score(160)).


/*  A score of 160 is asserted for demo purposes.    */
/*  The actual credit database would be called here  */


member(X,[X|_]).
member(X,[_|Y]):- member(X,Y).


report_meta:-
    call(score(X)),
    create(H3,'credit_2.rep'),
    write(H3,'score('),write(H3,X),write(H3,').'),nl(H3),
    close(H3),
    shell('copy credit_2.rep c:'),
    ke-_looking_k.
```

```
keep_looking_k:-
    main.

/*  end CREDIT_2.ARI  */
^z
```

```
/*      CREDIT_3.ARI     */
/*    database      */
    base_t(time(0,0,0,0)).
    base_d(date(0,0,0)).


/*    rules      */

:- public main/0.
main:-
     directory('c:\k_file.inp',_,_,T,D,_),
     base_time,
     base_date,
     new_time(T),
     new_date(D),
     asserta(base_d(D)),
     asserta(base_t(T)),
     shell('copy c:k_file.inp'),
     open(H,'k_file.inp',r),
     read_val(H),
     close(H),
   (( find_soln,
     calc_soln,
     report_meta ) ;
     keep_looking_k).

base_time:- base_t(time(W,X,Y,Z)).

base_date:- base_d(date(X,Y,Z)).

new_time(T):-
     base_t(B_T),
     ifthenelse(T=B_T,keep_looking_k,set_new_time).

new_date(D):-
     base_d(B_D),
```

66

```prolog
        ifthenelse(D=B_D,keep_looking_k,set_new_date).


    set_new_time:- retract(base_t(time(W,X,Y,Z))).


    set_new_date:- retract(base_d(date(X,Y,Z))).


    read_val(H):-
        repeat,
        read(H,T),
        asserta(T),
        recordz(val,T,_),
        T=end_of_file.


    find_soln:-
        find_es(E_Sys).


    find_es(E_Sys):-
        call(es(E_Sys)),
        member(credit_3,E_Sys).


    calc_soln:-  asserta(score(160)).


    /*  A score of 160 is asserted for demo purposes.      */
    /*  The actual credit database should be called here.  */


    member(X,[X|_]).
    member(X,[_|Y]):- member(X,Y).


    report_meta:-
        call(score(X)),
        create(H3,'crdeit_3.rep'),
        write(H3,'score('),write(H3,X),write(H3,').'),nl(H3),
        close(H3),
        shell('copy credit_3.rep c:'),
        keep_looking_k.
```

```
keep_looking_k:-
    main.

/*      End CREDIT_3.ARI      */
```

68

```
/*    PSYCH_1.ARI    */
/*   database    */
    base_t(time(0,0,0,0)).
    base_d(date(0,0,0)).


/*   rules    */


:- public main/0.
main:-
     directory('c:\k_file.inp',_,_,T,D,_),
     base_time,
     base_date,
     new_time(T),
     new_date(D),
     asserta(base_d(D)),
     asserta(base_t(T)),
     shell('copy c:k_file.inp'),
     open(H,'k_file.inp',r),
     read_val(H),
     close(H),
  (( find_soln,
     calc_soln,
     report_meta ) ;
     keep_looking_k).

base_time:- base_t(time(W,X,Y,Z)).


base_date:- base_d(date(X,Y,Z)).


new_time(T):-
     base_t(B_T),
     ifthenelse(T=B_T,keep_looking_k,set_new_time).

new_date(D):-
     base_d(B_D),
```

```prolog
            ifthenelse(D=B_D,keep_looking_k,set_new_date).


    set_new_time:- retract(base_t(time(W,X,Y,Z))).


    set_new_date:- retract(base_d(date(X,Y,Z))).


    read_val(H):-
         repeat,
         read(H,T),
         asserta(T),
         recordz(val,T,_),
         T=end_of_file.


    find_soln:-
         find_es(E_Sys).


    find_es(E_Sys):-
         call(es(E_Sys)),
         member(psych_1,E_Sys).


    calc_soln:-  asserta(score(170)).


    /*  A score of 170 is asserted for demo purposes.        */
    /*  The actual psychological database should be called here.
    */


    member(X,[X|_]).
    member(X,[_|Y]):- member(X,Y).


    report_meta:-
         call(score(X)),
         create(H3,'psych_1.rep'),
         write(H3,'score('),write(H3,X),write(H3,').'),nl(H3),
         close(H3),
         shell('copy psych_1.rep c:'),
```

70

```
        keep_looking_k.

keep_looking_k:-
     main.

/*    End PSYCH_1.ARI    */
```

REFERENCES

Biegl, C., Foxvog, D., and Kawamura, K., "Distributed Expert Systems in Prolog," The Eighteenth Southeastern Symposium on System Theory, IEEE Computer Society Press, 1986.

Chu, W.W., Holloway, L. J., Lan, M., and Efe, K., "Task Allocation in Distributed Data Processing," Computer, Vol. 13, No. 11, November 1980.

Chu, W.W., Lan, M., and Hellerstein, J., "Estimation of Intermodule Communication (IMC) and Its Applications in Distributed Data Processing Systems," IEEE Transactions on Computers, Vol. C-33, No. 8, August 1984.

Inmon, W. H., Management Control of Data Processing: Preventing Management by Crisis, Prentice-Hall, 1983.

Isett, J. B., Must an Effective Decision Support System be an Expert System?, paper presented at the Graduate School of Business, University of Texas at Austin, 22 April 1985.

Rowe, N. C., Class Notes for CS3310, Artificial Intelligence, unpublished manuscript, Naval Postgraduate School, Monterey, California, 1987.

Shen, C. and Tsai, W., "A Graph Matching Approach to Optimal Task Assignment in Distributed Computing Systems Using a Minimax Criterion," IEEE Transactions on Computers, Vol. C-34, No. 3, March 1985.

Silverman, B. G., "FACILITY ADVISOR: A Distributed Expert System Testbed for Spacecraft Ground Facilities," Expert Systems in Government Symposium, IEEE Computer Society Press, 1986.

Waterman, D. A., A Guide to Expert Systems, Addison-Wesley, 1986.

Williamson, M., "Will AI Fit Onto a PC?," Computerworld, 17 August 1987.

SELECTED BIBLIOGRAPHY

Bennett, J. L., _Building Decision Support Systems_, Addison-Wesley, 1983.

Benoit, J. R., et al., "ALLIES: An Experiment in Cooperating Expert Systems for Command and Control", _Expert Systems in Government Symposium_, IEEE Computer Society Press, 1986.

Burkholder, L., et al., "Prolog for the People", _AI Expert_, Vol. 2, No. 8, June 1987.

Clocksin, W. F., "A Prolog Primer", _BYTE_, Vol. 12, No. 9, August 1987.

Clocksin, W. F. and Mellish, C. S., _Programming in Prolog_, Springer-Verlag, 1987.

Davis, F. D., "Multiuser Programming", _BYTE_, Vol. 12, No. 8, July 1987.

Davis, W. S., _Systems Analysis and Design_, Addison-Wesley, 1983.

Delahaye, J. P., _Formal Methods in Artificial Intelligence_, John Wiley & Sons, Inc., 1987.

Entner, R. S. and Tosh, D. E., "Expert Systems Architecture for Battle Management", _Expert Systems in Government Symposium_, IEEE Computer Society Press, 1986.

Isett, J. B., _Decision Support System Design for Crisis Decision Making: An Experiment in Automated Support for Crisis Management_, Ph. D. Dissertation, University of Texas at Austin, Austin, Texas, May 1987.

Martin, J., _Design and Strategy for Distributed Data Processing_, Prentice-Hall, 1981.

Page-Jones, M., _The Practical Guide to Structured Systems Design_, Yourdon, Inc., 1980.

Pressman, R. S., _Software Engineering a Practioner's Approach_, McGraw-Hill, Inc., 1982.

Salzberg, S., "Knowledge Representation in the Real World", _AI Expert_, Vol. 2, No.8, 1987.

Stallings, W. Data and Computer Communications, Macmillan
Publishing Company, 1985.

Yourdon, E., Managing Structured Techniques, Yourdon, Inc.,
1986.

INITIAL DISTRIBUTION LIST

No. Copies

1. Defense Technical Information Center   2
   Cameron Station
   Alexandria, Virginia 22304-6145

2. Superintendent   2
   Attn: Library, Code 424
   Naval Postgraduate School
   Monterey, California 93943-5002

3. Chief of Naval Operations   1
   Director of Information Systems (OP-945)
   Navy Department
   Washington, D.C. 20350-2000

4. Dr. T. Sivasankaran   3
   Department of Administrative Sciences
   Code 54-SJ
   Naval Postgraduate School
   Monterey, California 93943

5. Dr. T. Bui   1
   Department of Administrative Sciences
   Code 54-BD
   Naval Postgraduate School
   Monterey, California 93943

6. Dr. Y. Mortagey   1
   Department of Administrative Sciences
   Code 54-MY
   Naval Postgraduate School
   Monterey, California 93943

7. LT Michael B. Rattigan   5
   17133 Creekside Circle
   Morgan Hill, California 95037

8. Dr. D. R. Whipple   1
   Chairman, Dept. of Administrative Sciences
   Naval Postgraduate School
   Monterey, California 93943

9. Mr. and Mrs. John P. Rattigan   2
   30 Holly Avenue, Apartment 204-H
   Shalimar, Florida 32579

10. Mr. and Mrs. Ben Sirmons Jr.   1
    510 Wicker Street
    Greensgoro, North Carolina 27403

11. Mr. and Mrs. William Ryan   1
    378 Bacon Street
    Waltham, Massachusetts 02154

12. Mr. and Mrs. Robert Hurstak   1
    230 Marlboro Road
    Sudbury, Massachusetts 01776

13. Mr. and Mrs. Peter W. Ryan   1
    249 Dale Street
    Waltham, Massachusetts 02154