

/DISTRIBUTED PROBLEM SOLVING FOR DECISION SUPPORT/

by

Mark C. Foehse
"

B. S., University of Missouri - Columbia, 1977

A MASTER'S THESIS

submitted in partial fulfillment of the

requirements for the degree

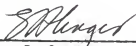
MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1986

Approved by:



Major Professor

LD
2668
.T4
1986
F63
c.2

A11202 965469

TABLE OF CONTENTS

TABLE OF CONTENTS	1
LIST OF FIGURES AND TABLES	111
ACKNOWLEDGEMENTS	iv
CHAPTER 1 - INTRODUCTION	1
CHAPTER 2 - LITERATURE REVIEW	5
2.1.1 Decision Support Systems - A Definition	5
2.1.2 Functional Components of Decision Support Systems	8
2.1.3 Decision Support Systems - Development Process	9
2.2.1 Expert Systems - A Definition	10
2.2.2 Expert Systems - Functional Components	10
2.2.3 Expert Systems - Development Process	13
2.3 A Comparison of Decision Support Systems and Expert Systems	13
2.4 Relevant Current Work	17
2.4.1 BEINGS	17
2.4.2 Scientific Community Metaphor	19
2.4.3 HEARSAY-II	21
2.5 Problem Statement and Proposed Solution	25
CHAPTER 3 - FUNDAMENTALS	28
3.1 Purpose	28
3.2 Environment	30
3.3 User	32
3.4 Need	33
CHAPTER 4 - COMPONENTS	35
4.0 Introduction	35

4.1	Blackboard	36
4.2	Objects	38
4.2.1	General Description	38
4.2.2	Activation	38
4.2.3	Execution	39
4.2.4	Structure	39
4.2.5	Types	40
4.2.5.1	Control Objects	40
4.2.5.2	Domain Objects	42
4.3	Messages	44
CHAPTER 5 - SIMULATED USE OF MODEL		51
5.1	Introduction	51
5.2	A Problem	51
5.3	A Solution	52
CHAPTER 6 - RESULTS AND CONCLUSIONS		62
6.1	Results	62
6.2	Challenges to the Model	63
6.3	Conclusions	64
6.4	Recommendations for Future Study	65
SELECTED BIBLIOGRAPHY		68

LIST OF FIGURES AND TABLES

Table 2-1	Decision Support System Definitions	6
Figure 2-1	User Interface Classifications	8
Figure 2-2	HEARSAY-II Hypothesis Links	23
Figure 3-1	Business Enterprise Hierarchy	31
Figure 4-1	Blackboard	37
Figure 4-2	Message	45
Figure 4-3	Status Attribute States	47
Figure 4-4	Longevity and Location	48
Figure 4-5	Representation of Plans	49
Figure 5-1	Simulated Decision Support System Hierarchy	53
Figure 5-2	Simulated Decision Support System Blackboard	54
Table 5-1	Simulated Decision Support System Messages	57
Figure 5-3	Manufacturing Firm Model	60

ACKNOWLEDGEMENTS

I wish to thank my advisor, Dr. Elizabeth Unger, for her support and encouragement during this work. It was a pleasure to work with her.

My thanks also go to the other members of my committee, Dr. Virgil Wallentine and Dr. Richard McBride, for their suggestions on this thesis.

I wish to thank Harvard Townsend and Robin Niederee for their help with the intricacies of mroff.

Last, I wish to thank my wife, Karen, for her unfailing support during the many years it took to achieve this goal.

Chapter 1

INTRODUCTION

Time has seen the assimilation of great numbers of automated systems into our everyday environment. In business, early computer applications were unit record data processing systems such as payroll processing. As the computer usage matured into another stage, the desire on management's part for more information led to the development of Management Information Systems (MIS). These systems were intended to provide more usable information in the form of charts, graphs, forms, et cetera, than their predecessors. The next evolutionary stage in business information systems involves the use of the Decision Support System (DSS). These systems are intended to aid managerial staff in the process of decision making by providing data analysis, models, and ease of communication between the tools, models, and decision makers.

Researchers in the area of computer science known as artificial intelligence (AI) set out in the late 1950s and early 1960s to build automated systems for natural language understanding and models of human thought processes and problem solving. Much of the success in artificial intelligence has been more a case of improved techniques than in finding real-world applications for artificial intelligence research. One notable exception to this has been the development of expert systems (ES). These programs attempt to match or exceed the problem solving ability of recognized experts within well-defined problem domains. Commercial applications of this technology are now

being developed.

Presently business managers and executives want greater problem solving support from their automated systems, and expert system developers have been able to create problem solving systems for specific problem domains. The time appears right for an integration of these two areas. It is quite possible that using expert system methodologies, decision support systems can be developed which go further than present day systems in meeting the needs of business decision makers.

At present there is no consistent theory of decision support systems. Decision support systems are broadly defined. Furthermore, it is not well understood how people make decisions, though several theories exist. What can be stated with certainty is that decision makers in business rely upon many sources of knowledge in trying to solve difficult problems. These problems may be characterized as having a large number of potential solutions, only a few of which are "good." Multiple sources of knowledge are used by the decision maker in an attempt to build a consensus with regard to the best problem solution available. This development of a consensus position is the role of the human support staff which a decision maker relies in part upon, i.e., selecting a few good alternatives from among the many that exist.

The development of a decision support system can be viewed as an attempt to automate the function of the decision maker's human support staff. In artificial intelligence, expert systems were developed to solve difficult problems, those having a few good solutions out of a

potentially large number of possibilities. Typically, these expert systems have dealt with narrowly defined problem domains. The business decision maker will need information from several different domains of knowledge, thus a single expert system, though it may solve difficult problems, will not have sufficient breadth to serve as a decision support system. An automated decision maker's support system will have to include knowledge from many problem domains. Research in artificial intelligence which has addressed the use of multiple knowledge sources in automated systems is known as distributed artificial intelligence, or distributed problem solving.

This thesis describes an abstract model architecture designed for decision support systems. This model is a combination of the results of research in distributed artificial intelligence and an organizational view of business enterprises. The goal of the model is to describe an architecture in which decision support systems can be developed to meet the needs of decision makers; that is, the incorporation of multiple diverse sources of knowledge for the purpose of solving problems.

The model has three primary components: multiple intelligent objects, messages, and a global blackboard. Each object represents some body of knowledge in a particular problem domain. The objects cooperate to solve problems by posting and reading messages on the blackboard. Messages all have the same structure; it is their value which makes each unique.

Following a review of the relevant literature, the role of the model in the business environment is discussed. A full description of the

model components is then given. An example of how the model may be used in a decision support system is provided, along with a discussion of the model's strengths and weaknesses. Recommendations for further study are given.

Chapter 2

LITERATURE REVIEW

2.1.1 DECISION SUPPORT SYSTEMS - A DEFINITION

There is no single, concise definition of a decision support system. Researchers in the field all seem to have a slightly different idea of what type of software system should be distinguished as a decision support system. The most common definition identifies decision support systems as systems designed to support unstructured managerial decision making.

That definition leaves a great deal of room for interpretation. The diversity of opinion on what is a decision support system is well shown by Ginzberg and Stohr [1982]. Their review of some of the decision support system literature to that time shows researchers defining decision support system in several ways (see Table 2-1).

Alter [1980] conducted a study of fifty-six different decision support systems. His assessment of these case studies was that the term decision support system did not refer to a "homogeneous category." Realizing the diversity of these systems, Alter devised a taxonomy of decision support systems based upon what he called the "degree of action implication of system outputs (i.e., the degree to which the system's outputs could determine the decision)."

<u>Source</u>	<u>DSS defined in terms of:</u>
Gorry and Morton [1971]	problem type, system function
Little [1970]	system function, interface characteristics
Alter [1980]	usage pattern, system objectives
Moore and Chang [1980]	usage pattern, system capabilities
Bonczek et al. [1980]	system components
Keen [1980]	development process
Sprague [1980]	system components, development process

Table 2-1
(adapted from Ginzberg and Stohr [1982])

These generic operations extend along a single dimension ranging from extremely data oriented to extremely model oriented:

- Retrieving a single item of information,
- Providing a mechanism for ad hoc data analysis,
- Providing prespecified aggregations of data in the form of reports,
- Estimating the consequences of proposed decisions,
- Proposing decisions, and
- Making decisions. (Alter [1980])

Using these operations as a criteria for judgement, Alter classified the fifty-six decision support systems in his study according to the following taxonomy:

- A. File drawer systems
- B. Data analysis systems

- C. Analysis information systems
- D. Accounting models
- E. Representational models
- F. Optimization models
- G. Suggestion models.

To be noted in Alter's taxonomy is the broad range of functions possible for a system labeled as a decision support system.

Bonczek et al. [1980] stressed the use of models in a system which was defined as a decision support system. They noted three major "interfaces" within a decision support system: the user-model interface, the user-data interface, and the model-data interface. Decision support systems were then classified according to the language(s) employed by the user in interacting with the system (see Figure 2-1).

Sprague [1980] proposed a decision support system "framework" based upon system components. He identified three different levels of technology: Specific Decision Support Systems, which are systems used in particular problem domains; Decision Support System Generators, which are packages of hardware and software used to quickly develop Specific Decision Support Systems; and Decision Support System Tools, the hardware and software functional components of Specific Decision Support Systems and Decision Support System Generators. Sprague also stated that the development process is an identifying characteristic of decision support systems, i.e., decision support systems should be developed iteratively, building adaptability into the system.

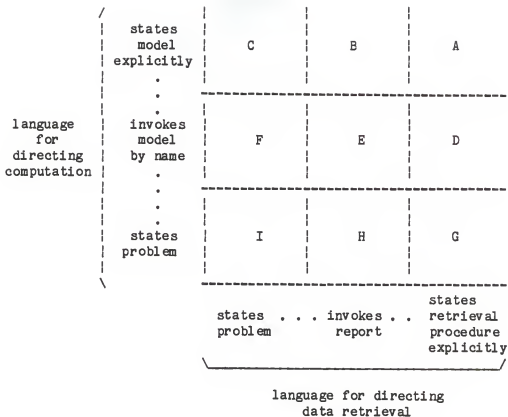


Figure 2-1
(Bonczek et al. [1980])

Ariav and Ginzberg recently made note of the variety of definitional criteria for decision support systems. They proposed yet another definition framework based upon systems theory.

The premise of the systemic view of DSS is that in order to understand these systems the following five aspects must be considered simultaneously: environment, role, components, arrangement of components, and the resources required to support the system. (Ariav and Ginzberg [1985])

2.1.2 FUNCTIONAL COMPONENTS OF DECISION SUPPORT SYSTEMS

Despite the differences of opinion on just how a decision support system should be defined, there does appear to be one area of agreement, and that is what are the components of decision support

systems. The literature identifies three major components of a decision support system: a user interface management system, a database management system, and a model base management system.

The user interface management system, or dialogue management system, acts as go-between for the user and the database and model base management systems. From the user's point of view this is the decision support system (Sprague and Carlson [1982]). As summarized by Bonczek et al. [1980], there is a great deal of variation in user interfaces. The range extends from systems where the user must have knowledge of FORTRAN or PL/1 to access data and/or models, to those which present the user with a menu of possible functions, such as the Portfolio Management System described by Keen and Scott Morton [1978].

The model management system provides access to models for use in simulation, process optimization, or providing suggested decisions. Again, there is wide variation in decision support system models. Simple algorithms called as subroutines, to linear programming algorithms, to high-level simulation languages such as GPS (General Problem Solver) have been identified as decision support system models.

The database management system handles data oriented tasks. Systems which do no more than retrieve raw data to those which produce data summaries in the form of pie charts and bar graphs on graphics terminals fit into this category.

2.1.3 DECISION SUPPORT SYSTEMS - DEVELOPMENT PROCESS

The process by which a decision support system is developed has been

discussed by many authors: Keen and Scott Morton [1978], Bonczek et al. [1980], Sprague and Carlson [1982]. The consensus is that decision support systems should be developed incrementally, that is, working closely with the user to identify what the system should accomplish. This may involve prototyping, that is, giving the user hands-on feedback with regard to design decisions.

To develop a system in an iterative fashion, it must be readily adaptable to change. System functions (hence components) must be easily added, deleted, or modified. If a system is developed in an iterative manner, incorporating adaptability, the result should not only please the user at the present time, but it will be able to evolve to satisfy the user at some future time.

2.2.1 EXPERT SYSTEMS - A DEFINITION

An expert system is a program designed to solve a problem or problems within a particular domain of knowledge. Such systems are built with the aid of someone who is considered to be an expert in the field in question. These systems attempt to codify the expert's problem solving methodology.

Expert systems differ from the broad class of AI tasks in several respects. First, they perform difficult tasks at expert levels of performance. Second, they emphasize domain-specific problem-solving strategies over the more general 'weak methods' of AI (Newell [1969]). Third, they employ self-knowledge to reason about their own inference processes and provide explanations or justifications for conclusions reached.

(Hayes-Roth et al. [1983])

2.2.2 EXPERT SYSTEMS - FUNCTIONAL COMPONENTS

In their simplest form expert systems have four functional components: rule base, working memory, an inference engine, and a user interface.

Expert systems rely on large databases of knowledge. A large number of expert systems use production rules to represent this knowledge. A production rule can be represented in the following form:

IF <pattern> THEN <action>.

A list of production rules is assembled into what is called the rule base, or knowledge base, of the expert system. A working memory is used to hold the current state of a problem under investigation. The driving program, or interpreter, of an expert system is the inference engine. The inference engine attempts to match the rule base against a subset of the working memory, driving towards a solution. In attempting to arrive at a solution the inference engine must employ some sort of control strategy.

There are two requirements for a control strategy (Rich [1983]). First, a good control strategy must cause motion, that is, progress must be made towards achieving some goal. Second, a good control strategy must be systematic. A systematic strategy is necessary to reduce the amount of time and effort expended in solving the problem at hand, as opposed to random or exhaustive search.

There are a number of different methods which can be employed as a control strategy by an inference engine. A pure recognize-act cycle searches the rule base for a rule whose pattern "matches" a subset of working memory. The first matching rule which is found is "fired", in other words, the actions specified by the rule are carried out. This

control strategy implies an ordering of the rules in the rule base.

Another control technique is conflict resolution. Here the entirety of the rule base is searched. All rules which match the working memory are selected. Two methods can then be used to determine which of the selected rules to fire; only one rule will be fired.

A decision procedure may be employed to rank the rules, selecting the "best" one to fire. Since this choice may not lead to a problem solution, backtracking may be necessary. Alternatively, viewpoints can be used to determine which selected rule to fire. In this method, which rule to fire is indeterminate. Thus a copy of the working memory is made and any one rule is fired using the copy. The firing of rules can continue in this manner until a problem solution is reached. If a solution cannot be reached, it is possible to return to previous states of the working memory and select a different rule to fire.

The user interface provides the means for interaction between the user and the expert system. This is extremely important in that these systems are designed to solve problems. If the human user is to accept the system's proposed problem solution, the user must be both comfortable with and have confidence in the expert system's reasoning process. This can be accomplished through explanation facilities in the user interface.

There are two important classes of situations in which expert systems should be able to explain their behavior and results. For the user of the system who needs clarification or reassurance about the system's output, the explanation can contribute to the transparency, and thus the acceptance, of the system. The second major need for explanation is in

the debugging process ..., where a human expert, in order to locate some error in the knowledge base, makes use of the system's explanations of why it has done what it has done. (Barr and Feigenbaum [1982])

2.2.3 EXPERT SYSTEMS - DEVELOPMENT PROCESS

The mention of debugging in the above section raises the issue of expert system development. In order to develop an expert system, a human expert must be found who is willing to invest the necessary time and effort. The process of interviewing the expert to determine how they go about their job, and then translating this knowledge into an expert system, is referred to as knowledge engineering. The development of the system becomes an iterative process of extracting knowledge from the expert, codifying it, and then testing.

Production systems are especially adapted to this iterative design process as knowledge can be easily added to the rule base via new rules. Knowledge which is correct and already encoded as rules is not disturbed.

2.3 A COMPARISON OF DECISION SUPPORT SYSTEMS AND EXPERT SYSTEMS

It should be immediately recognized that both decision support systems and expert systems are automated systems that deal with problems and problem solutions. Some researchers would have you believe that any similarities end there. Those who would support this statement are decidedly decision support system oriented, in other words, decision support system researchers and developers.

A return to some decision support system definitions and characteristics will serve to illustrate.

Decision support implies the use of computers to:

1. Assist managers in their decision process in semi-structured tasks.
2. Support, rather than replace, managerial judgement.
3. Improve the effectiveness of decision making rather than its efficiency. (Keen and Scott Morton [1978])

...DSS, which became characterized as interactive computer based systems, which help decision makers utilize data and models to solve unstructured problems. (Sprague [1980])

...they tend to be aimed at the less well structured underspecified problems that upper level managers typically face; they attempt to combine the use of models or analytic techniques with traditional data access and retrieval functions; they specifically focus on features which make them easy to use by noncomputer people in an interactive mode; and they emphasize flexibility and adaptability to accommodate changes in the environment and the decision making approach of the user. (Sprague [1980])

We propose that, for now at least, a definition of DSS quite close to the early definitions of Gorry and Scott Morton and Little be adopted. That is, a DSS is a computer-based information system used to support decision making activities in a situation where it is not possible or not desirable to have an automated system perform the entire decision process. (Ginzberg and Stohr [1982])

We stress supporting rather than replacing managerial judgements. We focus on improving the effectiveness of decision making rather than merely improving its efficiency. (Bennett [1983])

These authors define decision support systems in terms which imply that the type of problem solving activities decision support system users are involved in defy translation into computer programs. It appears further that not only should decision support system developers refrain from attempts to embody the problem solving methods of their users, but also that it is plainly impossible to do so.

A point to be made at this time is that there is no single, consistent theory of decision support systems. The above authors, in forming their definitions, are attempting to integrate specific decision support system implementations with what they feel should be a decision support system. I feel these definitions are weighted heavily towards what a decision support system is now in an implementation sense, as opposed to what a decision support system should be at some time in the future.

If we look towards the future of decision support systems the definitional views of decision support systems change somewhat. Bennett [1983] stated:

In an ideal DSS the computer would take an active role in leading the DM [decision maker] to a problem solution. This would require the computer to have an "understanding" of what the DM is seeking to do. In such a "knowledge-based" system, the DM and the computer would share responsibility for arriving at a "mutually satisfying" solution. In this idealized case the DSS would take an active role in directing the DM towards optimized decisions.

This statement contains the major difference that now exists between decision support systems and expert systems. Both decision support systems and expert systems use data (facts) and models (rules) to arrive at a problem solution. However, an expert system uses the codified inference methods of an expert to reach its solution. The decision support system solution is only as good as the current user's problem solving skills.

The need to include knowledge in decision support systems is being recognized, as knowledge about specific problem domains can lead to more efficient and efficacious decisions. Reitman [1982] observed

that the decision support provided by a human staff is still superior to that of a decision support system, if only in the fact that the human staff provides the decision maker with reasonable alternatives. The decision maker does not have to search through all possible scenarios. Knowledge about the problem domain, coupled with effective problem solving skills, can lead to better decisions.

If the knowledge and problem solving strategies of the human decision support staff, or an expert, could be codified in a DSS, a substantial impact could be made in unstructured managerial problems. (Gorry and Krumland [1983])

The coding of knowledge and problem solving methods is exactly what artificial intelligence researchers have done in expert systems, in an attempt to produce superior problem solutions. This work has not been lost on some decision support system researchers.

The systemic view makes it clear the ES bear many similarities to DSS. Their environments and roles are quite similar, and it is mainly a change in the arrangement and resources that differentiates them. (Ariav and Ginzberg [1985])

The following definition of an expert system could be applied equally to a decision support system which incorporates both knowledge of the problem domain and problem solving methods.

An expert system is one that has expert rules and avoids blind search, performs well, reasons by manipulating symbols, grasps fundamental domain principles, and has complete weaker reasoning methods to fall back on when expert rules fail and to use in producing explanations. It deals with difficult problems in a complex domain, can take a problem description in lay terms and convert it to an internal representation appropriate for processing with its expert rules, and it can reason about its own knowledge (or lack thereof), especially to reconstruct inference paths rationally for explanation and self-justification. (Brachman et al. [1983])

2.4 RELEVANT CURRENT WORK

A search of the current decision support system literature indicates little published work dealing expressly with the application of artificial intelligence techniques, specifically expert systems, to decision support systems. Reitman [1982] concluded that it may be possible to build a system of multiple experts for cooperative problem solving, as he did for the game of Go, but that a system of this type for decision support was not in the foreseeable future.

Artificial intelligence literature is the most applicable in this case. Several researchers have proposed models of problem solving by cooperating experts, sometimes referred to as knowledge sources. This area of research is referred to as distributed artificial intelligence or distributed problem solving. Though none of the following models or implemented systems deals directly with decision support systems, the techniques used may be applicable to that domain.

2.4.1 BEINGS

Lenat [1975a] modeled knowledge as interacting experts called Beings. Each Being is a "specialist" in some domain of knowledge. They cooperate to solve problems through questioning and answering each other. This implies that each Being recognizes when its own expertise is relevant.

The particular problem domains with which Lenat's systems dealt are automatic programming and the discovery of mathematical theories. Programs in both domains were successfully implemented using the Being concept.

Beings are uniform in that they all have the same mental "parts." The values of its parts make each Being unique. It is this difference in knowledge (values) that allows Beings to solve a task, while the uniformity of structure provides ease of communication.

Since the paradigm of the meeting [problem solving session] is questioning and answering, the names of the parts should cover all the types of questions one expert wants to ask another. (Lenat [1975a])

The number of parts in a Being is thus important. A large number of parts makes the addition of Beings difficult because of both the effort required and the knowledge necessary to assign values to all the parts. Lenat states that the optimum number of parts appeared to be in the range of 10 to 100. The systems he implemented had Beings consisting of ca. 30 parts.

Beings are not recursively defined. The Being parts constitute the primitive level of the system; there are no Beings defined as parts of Beings, nor are there any aggregations of Beings.

In a community of Beings, only one Being at a time has control. Each Being can recognize when it is relevant. Should more than one Being at a time want control, a special Being, CHOOSER, takes control. CHOOSER ranks the Beings wanting control, seeing which needs control most immediately. If there is still conflict, the simplest Being is given control. If the issue of control is still not resolved, a Being is chosen at random and given control.

Lenat concluded that, though a community of Beings did effectively solve problems within its defined domain, there were flaws in the

model. The community was implemented in PUP6 (Lenat [1975a]). One problem was that the addition of new Beings was difficult. Lenat noted that for the purposes of his experiments, only 30% of the parts of the Beings in the community were filled in. Another difficulty with the implementation was the awkward user interface. Dialogue was difficult due to the minimal completion of the system noted above.

2.4.2 THE SCIENTIFIC COMMUNITY METAPHOR

The Scientific Community Metaphor was developed by Kornfeld and Hewitt [1981] to model the parallelism in problem solving which occurs within scientific communities. Scientists do not work individually on problems, but rather concurrently with other scientists. At any time, there will be many theories extant in the community. Some scientists will be working in support of a particular theory while others believe the theory false and work to disprove it. In both cases, their research is supported by a financial sponsor. The language Ether was developed by Kornfeld and Hewitt to capture this.

All computation in Ether is carried out by sprites. Communication between sprites is accomplished by disseminating messages of two types: assertions and goals. An assertion is the result of some computation, while a goal indicates a computation which needs to be done.

Each sprite has a set of potential interests called the InterestSet. If a message is disseminated which is a member of a sprite's InterestSet, then that sprite will receive the message. The receiving sprite can then create new sprites and disseminate new messages.

Kornfeld and Hewitt [1981] listed four important properties of dissemination in Ether: monotonicity, commutativity, parallelism, and pluralism.

Monotonicity means that once a message is disseminated, it will remain available forever. This is akin to a scientist's published works; they will forever be a part of the literature.

The principle of commutativity states that a sprite will receive a message in its InterestSet whether that message was disseminated before or after activation of the sprite. This is equivalent to a scientist finding all previously published works of interest. And, as soon as new works of potential interest are published, the scientist will find these also.

Parallelism states that if message m_1 is received by both sprites s_1 and s_2 , then s_1 and s_2 will process m_1 concurrently. Also, should sprite s_1 receive messages m_1 and m_2 , it will process both messages concurrently. Analogously, scientists can work simultaneously without negative effects.

Pluralism allows Ether to work concurrently on multiple, possibly incompatible hypotheses. In the scientific community, there is no one source of truth.

There are four general types of activities carried out by sprites. First, there are proposers. These are processes whose job it is to propose new theses or goals upon which the community can act. Second, proponents are processes which seek to prove correct, or assert, the proposed goals. Third, skeptic processes attempt to disprove, or

refute, proposed goals. Fourth, sponsors are processes which determine how the community resources should be allocated. No processing work is done unless it is supported by a sponsor. Sponsors give support in the form of processing power, measured in units of cycles per second. Sponsors prevent the wasting of resources by processes attempting to prove results already known, i.e. goals which have been asserted.

Within Ether there are also mechanisms for adherence, viewpoints, inheritance, and translation. Adherence means that, though a theory may be believed at present, it may not be true in the future. Thus messages can become context-sensitive through labeling with their author, date, time of creation, etc. Messages can be "relativized" via viewpoints. Using viewpoints, assertions do not have to be global, believed by everyone. Thus different theories may exist with regards to the same subject. Inheritance allows information to be shared between viewpoints. One piece of information can be treated in multiple ways, depending upon one's viewpoint. In some instances information cannot be shared directly among viewpoints, rather some translation may be necessary. As an example, there may be two groups of scientists working on theories of light. Both groups need access to some particular body of knowledge, but one group views light as particles, while the other views light as waves. These different viewpoints require a translation of information between them.

2.4.3 HEARSAY-II

The HEARSAY-II speech understanding system (Erman and Lesser [1975]) utilized a hypothesize-and-test paradigm as "... the basis for

cooperation among many diverse and independent knowledge sources (KS's)." An interesting note is that the individual KS's were assumed to be "errorful and incomplete."

The type of AI problems addressed in the HEARSAY-II architecture are those having a very large problem (search) space and requiring large amounts of knowledge for solution. The different kinds of knowledge necessary for problem solution are represented by knowledge sources. KS's cooperate by writing hypotheses on a "blackboard."

The blackboard was a shared data structure to which all the KS's had access. When a KS was activated ... it examined the current contents of the blackboard and applied its knowledge either to create a new hypothesis and write it on the blackboard, or to modify an existing one. (Rich [1983])

The blackboard of HEARSAY-II is subdivided into seven levels.* These levels are heterogeneous and represented different levels of abstraction in the problem domain. Each level is an abstraction of the one below it. The levels, considered hierarchically, constitute a plan for solving the problem.

The hypotheses written upon the blackboard have a uniform structure, regardless of the level at which they are written. Hypotheses have associated attributes of several types: name, rating, attention, problem-specific, KS-specific, processing state, and structural relationships. Each hypothesis must have a unique name, including the name of its level. The KS-assigned ratings are used by a scheduler to guide the search for a solution. The attention attribute indicates

*The exact number of levels varied from three to eight, depending upon the configuration. The most cited number is seven.

three things: the amount of processing resources which have already been used on the hypothesis, the amount of additional processing that may be required, and what type of additional processing is needed. The latter can represent system goals. Problem-specific attributes allow the addition of supplementary information to the hypothesis which is germane to some particular problem. KS-specific attributes provide KS's with the state information necessary to process the hypothesis. These attributes also allow implicit (not blackboard) KS communication. The processing state attributes are summaries of the other hypothesis attributes. These summary attributes are efficient ways to trigger KS's. The structural relationship attributes represent relationships between hypotheses, using "links."

Links describe several kinds of relationships between hypotheses. There may be OR-, AND-, and SEQUENCE-links. In Figure 2-2, if links l1 and l2 were both OR-links, then hypothesis h1 could be either an h2 or an h3. If the links were AND-links then both h2 and h3 are necessary to support h1. SEQUENCE-links can be thought of as ordered AND-links.



Figure 2-2

Hypotheses may be linked both upward and downward. Link-attributes on downward links can be used to indicate supporting or contradicting

hypotheses, while upward links show where a hypothesis might be used.

Hypotheses may have multiple uses. Duplication of hypotheses is prevented through the use of an additional hypothesis attribute - a connection matrix. Since one hypothesis may have multiple uses the value of the connection matrix "specifies which of the alternative supports of the hypothesis are applicable ('connected to') which of its uses" (Erman and Lesser [1975]).

A knowledge source is specified in three parts: a) the conditions under which it is to be activated (in terms of the conditions in the blackboard in which it is interested), b) the kinds of changes it makes to the blackboard, and c) a procedural statement (program) of the algorithm which accomplishes those changes. A knowledge source is thus defined as possessing some processing capability which is able to solve some subproblem, given appropriate circumstances for its activation. (Erman and Lesser [1975])

When conditions on the blackboard match a KS's preconditions, an activation record is created indicating which KS should be activated and which event caused this activation (Rich [1983]). This specifying of the event which triggered the KS allows each KS to carry out its actions within a particular context. Instantiated KS's (those with an activation record) are selected for activation by a scheduler KS. The scheduler KS employs an "opportunistic search strategy."

Each instantiated KS has a hypothesis upon which it wants to operate. The scheduler examines the ratings of all instantiated KS's and will select for activation the KS with the highest rated hypothesis. This is best-first search. Should all hypotheses have an equal rating, the associated KS's would be activated together. This increases the breadth of the search. Choices among competing hypotheses are thus

delayed pending more complete information.

The hypothesize-and-test paradigm allows independent activation of KS's. Therefore, KS's need have no knowledge of one another. Also, because KS's are activated based upon certain blackboard conditions, the processing of hypotheses becomes data-directed.

Barr and Feigenbaum [1981] summarized the design ideas of the HEARSAY-II system as follows:

- Separate, independent, anonymous knowledge sources;
- Self-activating, asynchronous, parallel processes;
- Globally accessed database; and
- Data-directed knowledge invocation.

I believe we can add to this summary the use of multiple levels of abstraction in the problem solving process.

2.5 PROBLEM STATEMENT AND PROPOSED SOLUTION

To this author, it appears that the next step in the development of decision support systems is the addition of knowledge and problem solving methods. In other words, the next step in the development of decision support systems should be to incorporate the artificial intelligence techniques of expert systems. A huge monolithic general purpose decision support system is not what is envisioned. A different approach is suggested.

Currently, most decision support systems are developed for use in a tightly defined problem domain. The same is true of expert systems. Continuing in this manner, what is proposed is a system of domain

specific experts. Each of these experts would have knowledge of its problem domain and methods for solving problems in that domain. Each expert must also include knowledge of itself, so as to know when it is out of its area of expertise.

These domain specific experts can be linked in a network structure, where each expert is a node. There could conceivably be other types of nodes in the network. These additional nodes could serve as information processors, their task being the collection and processing of data for use by the domain specific experts. These information processing nodes would also require knowledge about themselves and the network, e.g., the expert should know what is its task and where its information is needed. Let us then refer collectively to the system nodes as intelligent objects.

Access to this network of intelligent objects by a human user would be through a user interface management system. To paraphrase Sprague and Carlson, from the user's point of view this would be the system.

The use of intelligent objects in this model is simply an extension of the recognized role of abstraction in programming languages and systems. Abstraction reduces the level of complexity in large systems, thereby raising both the level of understanding and comfort of human users.

Programming language developments which contributed to the object model can be traced back to SIMJLA67, developed by Dahl (Jones [1979]). SIMJLA67 provided for the aggregation of data and allowed operations on said data via the class construct. Instantiations of

classes were referred to as objects. This is known as data abstraction (Ghezzi and Jazayeri [1982]).

SIMULA67 influenced the development of later languages such as ALPHARD (Shaw and Wulf [1977]) and CLU (Liskov, et al. [1977]). These later languages included not only data abstraction, but procedural and control abstraction as well. More important was the idea that user-defined abstractions should not only combine data and the allowable operations on said data, but also protect the data from manipulation by any means other than the user-defined operations. It was this later point which was not effected in SIMULA67.

A simple object could be defined as an encapsulation of data and the allowable operations on that data. This type of abstraction mechanism is not sufficient, though, for complex distributed systems involving concurrent operations (Unger [1978]). The best means of representing the functional system components in this environment would be autonomous objects, that is, objects containing not just data and the operations on that data, but also mechanisms for independent action by the object. Unger [1985] defines this type of object as an intelligent data object (IDO). In the model defined by this research, data may not necessarily reside within an object, therefore the designation intelligent object.

Chapter 3

FUNDAMENTALS

3.1 PURPOSE

The purpose of this model is to integrate multiple diverse intelligent objects in an automated environment for the purpose of decision support, just as a human decision maker now uses the diverse talents of a human staff. This idea agrees in principle with the structure of the HEARSAY-II system (Erman et al. [1980]). In the HEARSAY-II system there was one uniform method of communication among many different knowledge sources. This differs from BEINGS (Lenat [1975a]) in which the structure of the objects was uniform.

The primary facets of this model are:

- a homogeneous environment for inter-object communication, i.e., a blackboard and messages,
- a blackboard for the posting of messages,
- heterogeneous intelligent objects,
- no restriction on the number of objects within the system, and
- no restriction on the locality of the objects.

The goal of combining a homogeneous communication mechanism with heterogeneous intelligent objects is system flexibility. This allows communication between objects but does not restrict how an object is

implemented. An analogy would be the system of roads and highways in this country and the vehicles which operate on them. As we know, there are many different types of cars and trucks, each designed for a particular purpose. All cars and trucks share the same roads and highways. If a uniform representation of objects was required in this model, that would be akin to designing one combination car/truck for use by all persons for all tasks for all time. Likewise, if a uniform communication mechanism were not enforced, the analogy would be a divided road system, one set of roads for cars, another for small trucks, another for large trucks, et cetera. It would be difficult to move goods from cars to small trucks, small trucks to large trucks, or vice versa due to this segregation; this is not a desirable characteristic.

The use of a blackboard for the posting of messages coincides with the expectation that this model system will be quite large, larger than what could be easily maintained by one individual. If one relaxed the requirements that messages be the only inter-object communication mechanism and that all messages be posted on the blackboard, in other words one allowed direct calls between objects, then a great deal of the system flexibility is lost. It would then be necessary to know all the possible calling sequences in which an object might be involved. It would become extremely difficult to remove existing system objects; that would be definitely undesirable. Software systems of the future may be so large that no one person will be able to know all possible inter-object effects. Hence, it will be necessary to have systems (objects) operate in an autonomous manner.

3.2 ENVIRONMENT

This model is intended for use in any environment. An environment means any person who needs to make decisions based on a large body of knowledge. It is expected that this body of knowledge will be diverse in nature. This model attempts to mimic the support a human staff gives to a decision maker; office personnel or technical support staffs usually represent many areas of expertise. This staff, to support the decision maker, must cooperate. The objects in this model must do likewise.

This model is loosely based upon the plex structure in which most business enterprises are established (see Figure 3-1). This enterprise structure is a hold-over from the age of the Roman Legions and beyond. It may be viewed as a hierarchy of units (corporate divisions, departments, personnel; archdioceses, dioceses, parishes; federal, state, and local governments, etc.), though it is not a strict hierarchy. Information flows both upward and downward through the levels, there is also lateral information flow. Ordinarily, the goals of the enterprise are communicated in a top-down manner, while the data is processed in a bottom-up manner (becoming information in the process). This model preserves the basic enterprise structure as described.

The idea of applicability in any environment is not unreasonable given the characteristics of the model. Though the communication mechanism is homogeneous (standardized), the heterogeneous intelligent objects will be for the most part, environment (application) specific. For example, a business executive's decision support system based on this

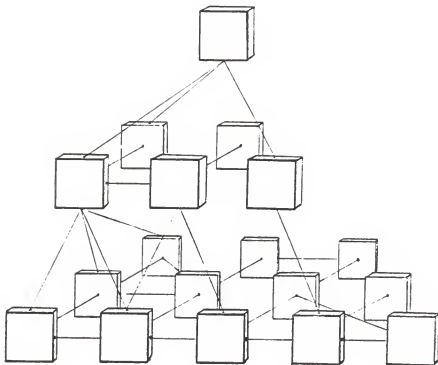


Figure 3-1

model might have the following objects as components:

- User-Interface object,
- Financial Advisor expert system object,
- Stock market data input object,
- Product market analysis expert system object,
- Statistical package object,
- Crisis Management expert system object,
- Graphics package object,
- Company Product database object, and
- Competitor's Product database object,

while an emergency room physician's system might include:

- User-Interface object,
- Myocardial Infarction expert system object,

Cerebral Vascular Accident (Stroke) expert system object,
Poison treatment expert system object,
Pharmacy database object,
Pharmacology expert system object,
Burn treatment expert system object,
Organ donor/recipient database object,
Medical dictionary database object,
Infectious disease expert system object,
On-call physician database object, and
Patient Monitor object.

3.3 USER

The model is designed for a single-user system. It is limited in this way for the sake of simplicity. This limitation removes any question of security. It is also assumed for the present that each object, when executing, does so to completion. This removes any problems associated with concurrency.

The user will interact with one intelligent object. This object's area of expertise would be human/system communication. The object could be tailored to the individual user, just as shell files in UNIX allow the tailoring of the user's environment. The User-Interface object could lead the inexperienced user by the hand via menus or it could provide a terse action oriented communication style for the more experienced user.

The user should be able to query the system through this User-Interface object as well as receive all system outputs through this

object. One exception to this might be a Graphics object, whose specialty is visual representation of data. Since the user interacts with only one object, he or she would be theoretically unaware of which other system object provided an answer to a query. The answer may have come from a local object, residing on the same processor as the User-Interface object, or the answer may come from a non-local object, since this model could be implemented on a distributed system. The location and identity of the responding object should be transparent to the user unless specifically requested.

All active objects in the system should be able to explain themselves as well as query the user if necessary. The user may wish to know how or why some output was arrived at by an object. In this case, the user should be able to ask the object to explain itself and the object could respond with a textual description of its function and how it carries out that function. An object may require more information before it can proceed with processing in some instance. If no other object can answer this object's query (posted as a message on the blackboard), the User-Interface object should output this query to the user. Under these circumstances, a dialog may ensue between the user and the object requesting information. Questions such as: "What do you need", "Why do you need that", "How do you know that", and "Where did that come from", would require answers on the part of the requesting object.

3.4 NEED

This model is particularly appropriate with respect to decision support systems for several reasons. The decision support system

literature indicates that existing systems tend to be one-of-a-kind. Those systems which are installed at multiple locations operate within a narrow problem domain. The current situation with implemented decision support systems is one where often there is no existing system applicable in and adaptable to a problem area. In this case the decision maker has to use conventional methods. Also, decision support system descriptions would indicate that these systems are large, essentially monolithic pieces of software. Such large systems are not easily modified. Neither are such systems easy to link together, given an business enterprise which might use multiple decision support systems.

This model alleviates such problems. It provides a framework independent of environment. A system designed under this model will be less like a traditional software system and more like a hardware system, in the sense that the user will begin with a base system and add components as needed. Some installations will have several components in common, while other components will be unique to that particular installation. The point is that these software components should be easily interchangeable (added/deleted).

If this easy interchange of software can be realized, then a potential user could start with a small, less expensive system and add other features at a later date. This is the sort of system evolution which decision support system researchers have identified as necessary.

Chapter 4

COMPONENTS

4.0 INTRODUCTION

There are three principle components of the model: the blackboard, intelligent objects, and messages. Knowledge within this model is embodied in the intelligent objects. This includes both procedural ("how-to") and declarative ("what") knowledge. Objects interact to solve problems by posting messages on the blackboard. The only inter-object communication mechanism is the message. All messages are posted on the blackboard; all messages have the same general structure. Together, messages and the blackboard form a uniform environment in which the objects operate.

The user posts messages on the blackboard via the User-Interface object. These messages are questions which the user wants answered. These questions may be thought of as system goals.

There are a number of different types of objects in the model. They may be viewed as being hierarchically arranged based upon where each posts its messages on the blackboard. Some low-level objects will be working constantly, collecting data for use by other objects, while other high-level objects are activated only in response to user queries.

For example, suppose one low-level object does nothing but poll a factory production line, counting the number of units produced.

Another low-level object watches the warehouse, counting the stock on hand of some particular item. It adds the number of units produced and subtracts the number of units shipped. These two objects would post their messages (outputs) on the blackboard. A third object could monitor customer orders. It might post messages at one minute intervals (if the product was electricity from a power plant), one hour intervals, or weekly (if the product was ready-to-eat cereal). An intermediate-level tracker object might read the messages posted by the customer order watcher and condense them to assess trends in product demand. It too would post its reports on the blackboard. All these messages may be read by a high-level inventory control object which generates reports or serves as a link in an automated production control system.

4.1 BLACKBOARD

The blackboard is an object which provides a global data structure accessible in one way or another to all objects. Subdivisions of the blackboard, called knowledge realms, represent particular problem domains. Within each knowledge realm the blackboard is further divided hierarchically into levels of abstraction (see Figure 4-1).

The blackboard is global in the sense that all objects use it. Communication between objects will tend to become localized, that is, all messages are not appropriate for all objects. Objects will communicate most frequently with those others who solve similar and/or related problems. This communication pattern is the motivation for subdividing the blackboard. The subdivisions correspond to realms of knowledge shared by several objects. An analogy, based on the

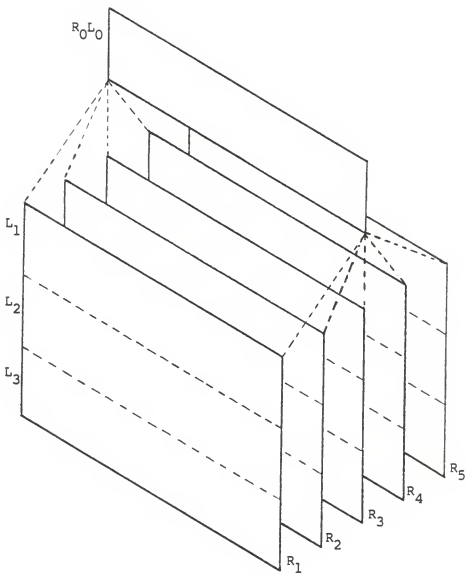


Figure 4-1

Scientific Community Metaphor, would be that the subdivisions are logically equivalent to the specialized discourse among experts in some particular field of science. To preserve the generality of the model, it must be noted that these specialized conversations are logical only; the structure of the messages cannot change from one blackboard subdivision to another.

Within each knowledge realm the blackboard can be further subdivided into several layers. These layers will correspond to logical subdivisions of knowledge (or tasks) within a given problem domain or, in other words, levels of abstraction. The determination of levels is akin to the top-down design process practiced in structured programming. The programmer begins with the most general and abstract problem components, refining each in turn into a lower level, more detailed description. For example, in the HEARSAY-II system (Lesser and Erman [1977]) the problem of recognizing spoken phrases was decomposed into the following six levels: phrase, word-sequence, word, syllable, segment, and parameter. Each level is a more detailed aspect of its predecessor. In this model, level R0L0 (knowledge realm 0, level 0) is the highest level of the blackboard. It is common to all knowledge realms. Distinct knowledge realms (R1..Ri) are differentiated beginning at level one downward (L1..Lj).

4.2 OBJECTS

4.2.1 GENERAL DESCRIPTION

Each object in the model may be viewed as a large-grained production rule of the form:

<condition> --> <action>.

The <condition> identifies those messages or blackboard states in which the object is interested, i.e., those messages upon which the object will act.

4.2.2 ACTIVATION

When a message is placed upon the blackboard which matches (triggers) an object's condition, the object is activated. At activation, a copy of the condition-matching message and its associated context, a summary of the action to be taken by the object and its effects upon the blackboard, and the objects rating (necessity of execution) are placed together in an activation record. This activation record is posted on a section of the blackboard reserved for control information (a control knowledge realm). There may be several object activation records posted on the control blackboard at any one time. Those objects whose activation records are posted are said to be pending execution. Since objects are activated based upon the blackboard state, the processing of messages becomes data-driven.

4.2.3 EXECUTION

A specialized object known as the Scheduler evaluates all pending activation records, selects one, and schedules the corresponding object for execution. The object selected for execution by the scheduler is thus in a ready state. When any preceding objects have run to completion the next scheduled object is executed, i.e., its <action> is carried out.

Each object, when executed, runs to completion. When implemented on a single-processor system, this criteria simplifies the model and negates problems associated with concurrency and shared data objects (the blackboard).

4.2.4 STRUCTURE

All objects in this model operate in the uniform environment provided

by the blackboard and messages. Recall that all messages have the same structure. Note however that objects are heterogeneous in structure. The objects within the model must be constructed with regard to a particular application; they are domain specific. As such, the system developer who uses this model must be granted the freedom to implement each object in an optimum manner. This lack of restraint on the internal representation of objects allows objects to be recursively defined.

4.2.5 TYPES

There are nine different types of objects segregated into two categories: control and domain. Substantial research on blackboard architectures has indicated the desirability of separating knowledge about the problem domain from knowledge about problem solving, hence domain and control objects, respectively (Balzer, et al. [1980], Aiello [1983], Hayes-Roth [1983], Hayes-Roth [1984], Hayes-Roth and Hewett [1985]). This distinction allows the modification of the problem solving method without disturbing domain knowledge, and vice versa. The result is a greater amount of generality and flexibility within the model.

4.2.5.1 CONTROL OBJECTS

The first category of objects, control objects, are divided into four types: blackboard handler, scheduler, policy, and user-interface.

The blackboard handler is a conceptual designation for the kernel routines used to add, remove, display, debug, and analyze information (messages and activation records) on the blackboard. Since all

objects use messages of a uniform structure, a standard set of blackboard functions can be created, simplifying system development.

The scheduler operates upon the activation records posted on the control knowledge realm of the blackboard. Its function is to rank the activation records of all triggered objects, selecting for execution that object whose activation record has the highest priority. For example, given two objects activated by the same message, if both objects perform a similar operation on the message but one uses twice the processing cycles to do so, the scheduler should select for execution the faster object. The factors which are used to calculate this priority will be problem-domain specific.

Policy objects are used to control the problem solving process in a more dynamic manner than the scheduler. This is accomplished by setting system goals which will trigger desired events. This does not violate the independence of objects as the policy objects are requesting (through goals) "what", not "who."

Policy objects are triggered by certain blackboard states. Events which result in a policy object triggering would be "quiescence", a condition in which objects above a designated level fail to activate, and "stagnation", which occurs when progress toward a goal state slows (Hayes-Roth and Lesser [1977]).

The user-interface object handles all interaction between the human user and the model system. The user inputs all queries to the system through the user-interface object. These inputs may be thought of as system goals or directives. These goals will be posted in the form of

messages by the user-interface object on section R0L0 (realm 0, level 0) of the blackboard. Recall that R0L0 spans all knowledge realms within the model. High-level domain objects and expert objects (both described below) respond to messages posted at that level.

4.2.5.2 DOMAIN OBJECTS

The second category of objects includes five types: expert, high-, intermediate-, low-level, and database. These objects capture the information germane to a particular problem domain (or knowledge realm).

Within any given field of knowledge, some processes will be well understood while others will not. In decision support system terminology, some problems are structured, others unstructured. Expert objects are intended to capture structured knowledge, that is, problems for which known solution methods exist. This covers a broad range of possibilities from the "problem" of finding a number's square to determining if a hospital patient has septicemia, e.g., MYCIN and other expert systems. Throughout this range, these problems all exhibit some degree of bounds, at least sufficient to allow the creation of a solution. This "well-defined" solution can be coded, compiled, and used in compiled form. Thus expert objects in this model can be viewed as compiled solution methods to particular problems.

A sophisticated expert object, a genuine expert system for instance, would respond directly to messages posted by the user-interface object on R0L0 of the blackboard. A simpler expert object could be used at

lower blackboard levels by other objects (recall that objects can be recursively defined). Contrast the class of problems defined by these expert objects with the class of problems for which the HEARSAY-II architecture was developed.

The basic premise of the HEARSAY-II system was that both the hypotheses and the knowledge source used by the system were "errorful and incomplete." In other words, the problem domain was not well structured; the methods by which problem solutions are arrived at are not well understood. Allowance for problems and solutions of this nature has been made through the inclusion of high-, intermediate-, and low-level objects in this model.

The designations high-, intermediate-, and low-level are arbitrary and refer to the blackboard levels at which these objects operate. Recall that blackboard levels correspond to levels of abstraction within a particular problem domain. This assumes that even unstructured problems can be decomposed into a hierarchy of partial solutions. High-, intermediate-, and low-level objects will cooperate while working at different blackboard levels to arrive at a problem solution. The number of levels into which a problem should be decomposed (hence the number of blackboard levels) should be left to the system designer.

Decision support researchers have indicated that decision support system development should be evolutionary. This idea is embodied in the high-, intermediate-, and low-level objects in this model. Artificial intelligence researchers using blackboard models have noted that, once a problem solution method is well understood, the method

should be re-implemented in some model other than a blackboard, thus allowing compilation and more efficient execution. That idea is captured in the expert objects of this model.

Note that some low-level objects will serve as data inputs for the model. These objects act as "sources" in Petri Net terminology (Peterson [1977]), bringing information into the model to be acted upon by other objects.

The last type of domain object is the database object. These objects, as their name implies, are simply data repositories. They provide, in response to a message, data for use by the human user or other objects.

In general, the number of objects within the model will be dependent upon the problem domain.

4.3 MESSAGES

Messages are the only communication mechanism between objects. All messages have the same structure; what distinguishes one message from another are the values of its parts. Not all message parts need be defined (have a value). Messages may be passed from higher blackboard levels to lower levels ("I need an answer to ...") and vice versa ("I have a response to ...").

A message consists of attributes as listed in Figure 4-2.

The message identifier is composed of two parts: name and context. The message name is generated by the blackboard handler at message creation (posting). All names must be unique throughout the

```

identifier
  name
  context
    knowledge realm (R0..Ri)
    level (L0..Lj)
rating
attention
  processing-cycles-used
  processing-cycles-needed
  operation-needed
  status
temporal
  longevity
  location
  replications
  authorization
processing-state
structural relationships
  upper hypotheses links
    connects-with
    type
    implication
    implication-strength
  lower hypotheses links
    connects-with
    type
    implication
    implication-strength
value

```

Figure 4-2

blackboard. The message context is of two parts - a label indicating the knowledge realm (R0..Ri) of the blackboard on which the message was posted as well as a label indicating the blackboard level (L0..Lj) within the knowledge realm of posting.

The rating attribute is a numeric index of the message's validity. A message's validity may range from positive fact (an absolute truth in support of something) through suppositions (weak support or contradiction) to negative facts (an absolute truth which contradicts something). This variability must be accommodated. The manner by which message ratings are indicated must be left to the system

developer e.g., +100.-100, +1.0.-1.0, etc.

The attention attribute has four components: processing-cycles-used, processing-cycles-needed, operation-needed, and status.

The processing-cycles-used is a scalar representation of the computing resources already expended by objects in processing a message. Processing-cycles-needed is an indication of the computing resources necessary to complete operations on a message. Processing-cycles-used and processing-cycles-needed are supplied by the objects which operate upon a message. A message which states a fact would have a low processing-cycles-used index (the cost of retrieval from a database object) and a zero processing-cycles-needed index. A problem hypothesis represented as a message would, in contrast, have a far greater range of values possible for these indices. The processing-cycles-used/processing-cycles-needed attributes could be used by control objects for resource allocation within the model.

Operation-needed indicates what processing needs to be done on the message by other objects. Facts would need no further processing, whereas hypotheses may need support evidence developed.

The status component of the attention attribute can be of five types: user-interface-query, in-process, done, user-interface-reply, and object-query. The allowable status attribute state transitions are shown in Figure 4-3.

The user-interface-query status would be assigned to a message when it was first posted on the blackboard by the user-interface object. This message would be a human user query. If the user-interface-query

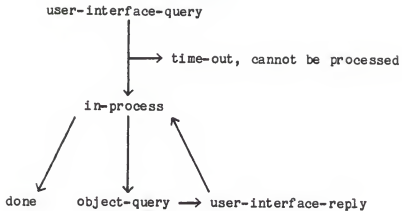


Figure 4-3

status remained unaltered for some specified number of object executions, the user-interface object could return an output message to the user indicating the query message could not be processed, i.e., it triggered no objects. The user could then reformulate the query.

Should a message marked user-interface-query trigger an object, causing the creation of an activation record, the message's status would be changed to in-process. An in-process status indicates processing is being performed on the message.

When processing upon a message is complete the status would be set to done. A done status serves as a trigger to the user-interface object that a message should be output to the user.

Situations may occur in which a message is in-process, but processing of the message halts. The message does not trigger an object to perform the operation-needed. The last object to process the message can then change the message status from in-process to object-query. An object-query status indicates to the user-interface object that no further processing can be done on the associated message without

additional information. It is the user-interface object's responsibility to obtain additional information that would allow message processing to proceed.

This additional information, in response to an object-query, will be posted on the blackboard with a status of user-interface-reply. The object which posted the object-query message would then trigger on the user-interface-reply message and attempt to proceed with message processing. This query/reply cycle could require several iterations before message processing can proceed.

The temporal attribute has four components: longevity, location, replications, and authorization.

Longevity represents the lifespan of a message within its context. Closely coupled with longevity is location, which represents the messages beginning horizontal coordinate on the blackboard level where it is posted (see Figure 4-4).

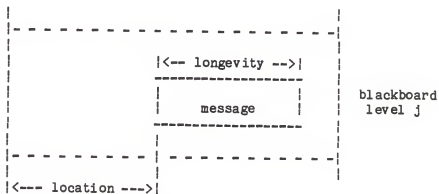


Figure 4-4

In some problem domains the longevity and location of a message will

be of importance. If, for instance, four messages were linked together representing the four stages of a plan, different plans could be represented through differing longevity and location values (see Figure 4-5).

Plan 1

longevity:	25	25	25	25

	phase 1	phase 2	phase 3	phase 4

location:	0	25	50	75

Plan 2

longevity:	19	27	33	21

	phase 1	phase 2	phase 3	phase 4

location:	0	19	46	79

Figure 4-5

The replications attribute is a positive integer indicating the number of copies of a message extant within the model. Multiple objects may be triggered by one message. If this model were implemented as a concurrent system all triggered objects could process individual copies of a message in parallel, thus the necessity of the replications attribute.

The value of the authorization component determines whether a message

may be replicated. As such, it can be represented as a boolean value.

The processing-state attribute constitutes a change record associated with each message. The processing-state entries comprise a chronological list of all processing performed on a message. Each entry in this list would include the identifier of the processing object, the operation performed on the message, the time at which the processing was performed, and a description of why the processing was done. This information would be of import in debugging a system based on this model. Also, to be successful a decision support system must have the user's confidence. The information contained in the processing-state attribute would be of use in an explanation facility which could explain to the user how problem solutions were developed.

The structural-relationships attributes link messages together to form an information net. Based on the HEARSAY-II architecture, these structural-relationships would be upward and downward links (to other messages) composed of four parts: connects-with (which other messages), type (logical AND; OR; and SEQUENCE, an ordered AND), implication (support or contradict), and implication-strength (numeric index).

The value attribute of a message would be the information to be conveyed from one object to another. In the event the message represented a datum, the structure of the value attribute could be a simple, aggregate, or enumerated data type. In general, the structure of the value attribute will be problem domain dependent. In fact, the model has the designed capability to be a representation of a large dynamic problem solution.

Chapter 5

SIMULATED USE OF MODEL

5.1 INTRODUCTION

The use of the model as described in this thesis will be illustrated using a problem as proposed below. The problem is quite simple. This is not to imply that this model can be used only on simple problems, but rather to keep the illustration tractable and to allow the reader to easily follow the sequence of events.

5.2 A PROBLEM

To simulate the use of the model, consider the following problem. There exists a business, Terri's Typing, which types theses and dissertations for graduate students.* Terri's Typing is a sole proprietorship, and Terri, the owner, wants to estimate her profits for the upcoming third quarter.

Terri's Typing is a unique business in several aspects. Terri operates out of her home. She does this to avoid any concerns relating to rent, utilities, insurance, etc. with regard to the business. Terri buys her own paper, her only expense. She was given a typewriter which she expects will last forever, hence no depreciation. The demand for Terri's typing is unlimited, thus the quantity of product produced (number of pages typed) is a function of

*The idea for a typing service was drawn from Forgiunne [1986].

the rate of typing (pages per hour) and the number of hours worked (forty per week, exactly). These factors combine to form a simple model of Terri's business.

This model may be represented mathematically by the following set of equations.

$$\begin{aligned} P &= p * Q \\ p &= i - e \\ Q &= r * h \\ r &= c / t \end{aligned}$$

where:

P = total profit
p = per page profit
Q = quantity produced (number of pages typed)
i = gross income per page
e = expense per page
r = rate of typing in pages per hour
h = hours worked per week
c = total number of pages typed
t = total elapsed work time in hours

5.3 A SOLUTION

A decision support system is to be implemented based upon the business model for Terri's Typing described above. This decision support system will be designed to answer Terri's questions about profits; specifically, what are the estimated third quarter profits?

Recall that the primary components of this thesis' model are the blackboard, messages, and objects. In the decision support system being developed for Terri's business the blackboard will consist of only one knowledge realm associated with profits, thus no distinction will be made between knowledge realms and the blackboard.

The blackboard will be subdivided into four levels based upon the

hierarchical structure of Terri's business model (see Figure 5-1). There will be multiple objects associated with each level; several of these objects correspond to the high-, intermediate-, and low-level objects of the thesis model.

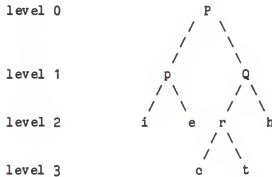


Figure 5-1

The User-Interface object will post Terri's queries on the top level (level 0) of the blackboard. The Profit object will be activated by messages dealing with profits. The second level of the blackboard (level 1) will be associated with the Page-Profit and Quantity objects; each object will be activated by messages about per-page profits and quantities produced, respectively. The third level of the blackboard (level 2) has four associated objects: Income, Expense, Rate, and Hours. The Rate object responds to messages about number of pages produced per hour. The other three objects are all database objects. In this simple business model each contains only 1 fact. Income contains the per-page fee charged by Terri for typing, currently \$0.80. Hours holds the number of hours worked per week by Terri; she works exactly forty hours per week. The Expense object contains the per-page cost of typing paper, which is \$0.005. The

lowest level of the blackboard (level 3) has two associated objects. The Counter object maintains a count of the total number of pages typed by Terri. The Counter object increments itself every time Terri finishes and removes another page from the typewriter. The Time object logs the total number of hours which Terri has typed.

The ten objects in Terri's decision support system may be pictured as shown in Figure 5-2. Note the four blackboard levels with multiple objects associated with each level. The objects are represented by directed arcs indicating the levels at which each object reads and posts messages on the blackboard.

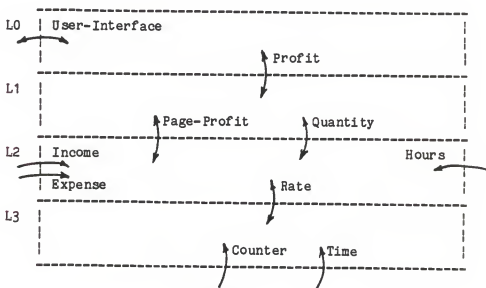


Figure 5-2

Given the decision support system described above, examine the trace of the sequence of events which follows when Terri enters the query "What will be the estimated third quarter profits?" The messages which are generated by the decision support system objects in answering this query have their attributes listed in an abbreviated

manner in Table 5-1.

When Terri sits down at the computer terminal and begins to type, she is interacting with the decision support system's User-Interface object. After Terri types her query "What will be the estimated third quarter profits?", the User-Interface object forms the query into a message M1, posting M1 on blackboard level 0 (L0). Since the value attribute of M1 deals with profits, Profit is activated and the status attribute of M1 is changed from user-interface-query to in-process. As Profit is the only object activated by M1, it is immediately executed by the Scheduler object.

The body of Profit is procedural in nature, that is, to complete the processing of M1, Profit must calculate $P=p*Q$. Profit creates two new messages, M2, which is "What is the per-page profit?", and M3, which is "What quantity will be produced in the third quarter?", because it does not possess values for p and Q. Both messages are posted by Profit at level L1.

Message M2 activates the Page-Profit object, while M3 activates the Quantity object. In this particular example, all objects must be activated and executed before a reply to Terri's original query can be produced. Therefore, there is no selection to be made among activated objects and further discussion of the Selection object is moot.

Assume the Page-Profit object executes before the Quantity object. Page-Profit is procedural, knowing it needs to calculate $p=i-e$ in response to M2. As Page-Profit does not have values for i and e, it posts two messages on L2: M4, which is "What is the value of i?", and

M5, which is "What is the value of e?"

Message M4 activates the Income object; it executes, posting M4 with attributes value = 0.80, operation-needed = none, and status = done. Recall that Income is a database object whose sole purpose is to return a value in response to a query. The Expense object is likewise a database object. It is activated by M5, executes, and posts M5 with value = 0.005, operation-needed = none, and status = done.

The change in attributes on M4 and M5 will activate/execute the Page-Profit object, calculating $p=i-e$. This value (0.795) is posted in M2 with operation-needed = none and status = done.

The Quantity object was activated by M3; assume it now executes attempting to calculate $Q=r*h$. Lacking a value for r, Quantity will post M6, "What is the rate of production?", at level L2.

Message M6 will activate/execute Rate. To obtain its needed values, Rate will post two messages at L3. The first, M7, will be "What is the total number of pages typed?", while the second, M8, will be "What is the total elapsed time?"

Terri has been in business just over four years, working 8640 hours and typing in this time 77760 pages. The Counter object will activate/execute on M7, posting M7 with value = 77760, operation-needed = none, and status = done. M8 will activate/execute the Time object, causing it to post M8 with value = 8640, operation-needed = none, and status = done.

These changes to the attributes of M7 and M8 cause Rate to again

Name	Context	Operation-Needed	Status	Processing-State	Value
M1	ROL0	define value	user-interface-query	1) user-interface goal	estimated third quarter profits?
M1	ROL0	define value	in-process	2) received by Profit	estimated third quarter profits?
M2	ROL1	define value	in-process	1) Profit subgoal	per-page profit?
M3	ROL1	define value	in-process	2) received by Page-Profit	quantity produced in third quarter?
M4	ROL2	define value	in-process	1) Page-Profit subgoal	value of <i>i</i> ?
M5	ROL2	define value	in-process	1) Page-Profit subgoal	value of <i>e</i> ?
M4	ROL2	define value	in-process	2) received by Income	value of <i>i</i> ?
M5	ROL2	define value	in-process	2) received by Expense	value of <i>e</i> ?
M4	ROL2	none	done	3) value defined by Income	\$0.80
M5	ROL2	none	done	3) value defined by Expense	\$0.005
M2	ROL1	none	done	3) calculate $p-i-e$	\$0.795
M3	ROL1	define value	in-process	2) received by Quantity	quantity produced in third quarter?
M6	ROL2	define value	in-process	1) Quantity subgoal	rate of production?
M6	ROL2	define value	in-process	2) received by Rate	rate of production?
M7	ROL3	define value	in-process	1) Rate subgoal	total number of pages typed?
M7	ROL3	define value	in-process	2) received by Counter	total number of pages typed?
M7	ROL3	none	done	3) value defined by Counter	77760 pages
M8	ROL3	define value	in-process	1) Rate subgoal	total number of hours worked?
M8	ROL3	define value	in-process	2) received by Time	total number of hours worked?
M8	ROL3	none	done	3) value defined by Time	8640 hours
M6	ROL2	none	done	3) calculate $r-c/1$	9 pages/hour
M9	ROL3	define value	in-process	1) Quantity subgoal	number of hours in a work-week?
M9	ROL3	define value	in-process	2) received by Hours	number of hours in a work-week?
M9	ROL3	none	done	3) value defined by Hours	40 hours/week
M3	ROL1	none	done	3) calculate $Q-r^i \cdot v^13$	4680 pages/quarter
M1	ROL0	none	done	3) calculate $P-p^qQ$	\$3720.60

Table 5-1

activate/execute, calculating $r=c/t$ and posting M6 with value = 9, operation-needed = none, and status = done.

With the changes to M6, Quantity moves closer to calculating a value for $Q=r*h$. Before Quantity can do so, it needs a value for h. M3, which originally activated/executed Quantity, specified a time period of one quarter. M6's value was expressed in hours; Quantity must therefore change a quarter into hours. To do so, Quantity will combine internal data with a response to message M9, "How many hours are currently in a work-week?", which it posts on L3. M9 activates/executes Hours, a database object, which replies by posting M9 with attribute value = 40, operation-needed = none, and status = done. With the value of M9, Quantity now uses an internal unit conversion table to calculate $40 \text{ hours/week} * 13 \text{ weeks/quarter} = 520 = h$. With h in the proper terms, $r*h$ is computed and posted as message M3 with value = 4680, operation-needed = none, and status = done.

Messages M2 and M3 now have a status of done, Profit again activates/executes and carries out its calculation, $P=p*Q$, and posts M1 with value = \$3720.60, operation-needed = none, and status = done. The User-Interface object detects the change in M1's status and outputs the result, M1's value of \$3720.60, to Terri.

This simple example illustrates how the thesis model may be used to structure a decision support system. The sample problem was decomposed into hierarchical levels of abstraction, yielding four blackboard levels. Multiple objects were described, each with its own area of expertise. The cooperation of objects through the posting and reading of messages was traced, showing how message attributes change as a

result of successive processing operations performed by the objects.

To repeat, a simple business enterprise model was selected to keep the example tractable and to allow the reader to easily follow the sequence of events. What may not be appreciated by the reader is the rapid growth in complexity of the problem solution process if the business enterprise model is expanded. Consider the following business model based on a Fortune 500 manufacturing firm (see Figure 5-3). Even as shown, this model is simpler than the actual corporation on which it is based. Consider what may transpire in attempting to answer the same query used above, i.e., what will be the estimated third quarter profits?

Potentially, almost every department of every division possesses information which impacts upon the answer to the query. The Industrial Engineering Department may have noticed a recent downward trend in production line productivity. Warehouse Operations may have scheduled construction for loading dock modifications early in the quarter, severely limiting shipping. The Market Trend Analysis Group may have noted a steady upward trend in demand over the previous year, though this may be tempered by a traditionally lower third quarter demand due to seasonal fluctuation. The Legal Division may report that the Federal Trade Commission has begun antitrust proceedings in Federal Court, indicating a lengthy court battle and large expenditures for legal services.

A sophisticated decision support system would have to take all these pieces of information into account in generating an answer to the query. In the simple example of Terri's Typing, all objects had to

Manufacturing Division
Industrial Engineering
Inventory Control and Scheduling
Maintenance and Mechanical
Quality Control
Shipping and Distribution
 Raw Materials Storage and Delivery
 Finished Product Storage and Delivery
 Warehouse Operations
Production
 Processing
 Packaging
Operations

Sales and Marketing Division
Advertising
Customer and Consumer Services
Product Liability and Insurance
Market Planning
 Profit Factor Analysis
 Merchandising Methods Analysis
 Business Cycle Analysis
 Cost Analysis and Projection
 Customer Analysis
 Market Trend Analysis
 Price/Demand Analysis
Product Planning
Sales Administration

Accounting and Finance Division
Auditing and Accounting
Budgeting
Credits and Collections
Financial Planning and Analysis
Taxes
Treasury

Administrative Services Division
Personnel
Benefits
Compensation
Public Affairs
Purchasing

Legal Services Division

Technical Services Division (Science and Engineering)

Figure 5-3

execute before arriving at a solution, and some degree of parallelism was possible. In this later, more "real-world" example, not all of the objects would have to execute thus introducing a greater degree of non-determinism. Unlike Terri's Typing, messages no longer represent facts but beliefs, requiring the use of message ratings. The need for control objects arises, as the selection from among competing messages becomes necessary. Further, tremendous parallelism is possible in a decision support system which captures this larger business enterprise model. In the face of such complexity, the uniform operating environment of this thesis model may provide a solution.

Chapter 6

RESULTS AND CONCLUSIONS

6.1 RESULTS

This model describes how a decision support system may be structured as heterogeneous intelligent objects cooperating in a homogeneous environment. The use of heterogeneous objects organized in a plex structure parallels the organizational scheme of most business enterprises.

The model views the components of a decision support system at a more abstract level than other current descriptions, through the use of knowledge realms. Thus the model is more general than other decision support system descriptive models. At the same time, this model is more finely grained through the use of hierarchical decomposition of problem domains (levels within knowledge realms) and the use of heterogeneous intelligent objects.

The use of a global blackboard subdivided into knowledge realms and levels, and messages of uniform structure for inter-object communication provides a homogeneous environment in which objects may be easily added to or deleted from the model. This increases the adaptability of the model to particular problem domains as well as its flexibility within problem domains. This also provides for the evolutionary development of decision support systems, cited as necessary by many researchers. Additionally, a homogeneous environment of operation would allow for the use of "stock" software

modules (objects), an interchangeable library of programs as envisioned by computer scientists over three decades ago.

The separation of control knowledge from domain knowledge permits experimentation with one form of knowledge without disturbing the other form. Independent optimization of problem solving methods and problem domain expertise can thus be realized.

6.2 CHALLENGES TO THE MODEL

There are a number of difficulties inherent in the model which would slow its becoming widely implemented in actual decision support systems. This is not to say that these difficulties are insurmountable, as several systems have been implemented using a blackboard architecture.

The uniform structure of messages is both a model strength and weakness. A uniform structure may be difficult to realize in a large system incorporating many knowledge realms due to the diversity of the information to be represented. As such, system developers may be prone to relax the uniform message structure requirement. A relaxation of this requirement, or any deliberate attempts to limit object independence, subverts the idea of a homogeneous model environment. This subversion would be the downfall of the model.

Multiple knowledge realms and levels within realms of the blackboard present problems for the system developer. First, this assumes a problem domain has a natural decomposition, that is, the domain has several recognizable levels of abstraction. Second, there are no rules or guidelines to help the system developer determine these

levels, either how many or what they should represent. The HEARSAY-II speech understanding system went from three, to six, to seven blackboard levels in its successive incarnations. The path to an optimum number of levels is experimentation. Business personnel, both system users and management, may have little tolerance for a software system that seems never to be done.

The User-Interface object will be a difficult piece of software to develop. It must be comfortable for the user while also dealing with many forms of data (message attributes). The requirements of simple and powerful are difficult to weld together.

The use of multiple intelligent objects introduces the complexities normally associated with distributed systems. These systems are harder to develop, code, debug, and maintain than more traditional software.

Technological support for implementing this model is still lacking. Many problems previously viewed as difficult become easy in light of new technology: moving heavy objects versus the wheel, communication across vast distances versus the telegraph, distant travel versus powered flight, etc. Though advances are made almost daily in VLSI circuit technology and software systems, the tools to readily implement this model are found, at best, in research laboratories and are certainly not found in wide commercial distribution.

6.3 CONCLUSIONS

The approach to structuring decision support systems taken by this model uses three primary components: a global blackboard, intelligent

objects, and messages. The blackboard is subdivided into knowledge realms associated with particular problem domains. Intelligent objects germane to each domain cooperate to solve problems via the posting of messages on the blackboard. Messages are the only inter-object communication mechanism. All messages are of uniform structure; messages are distinguished one from another by the values of their attributes.

6.4 RECOMMENDATIONS FOR FUTURE STUDY

Given that this model was based upon the hierarchical structure of most business enterprises, it would be appropriate to model a real-world business using this model's architecture. This would be of aid in determining the validity of this model. Such a real-world business model would also provide baseline data for determining appropriate blackboard knowledge realms and levels in addition to the specific number and function of objects necessary.

The intelligent objects in this model may be thought of as abstract data types. As such, the development of complete operational and/or denotational specifications for each object type would be of benefit in clarifying each objects function, hence its area of applicability.

Partial implementation of this model would be of aid in determining the optimum structure of messages. Though a truly optimum structure may never be found, no good or adequate structures will be ascertained without experimentation.

Implementation would also allow experimentation on different control strategies within one problem domain (Aiello [1983]). Studies could

also be carried out to determine the most suitable technology to use in systems based on the model.

A description of the means by which learning could be incorporated into this model would be significant. A model system which learned how humans solved less well-structured problems could automatically code the solution method as one or more objects. The solution method would thus be captured by the system and easily re-used by the user.

Developing the model as a multi-user system versus a single-user system would be a worthwhile research effort. It is highly likely that in a business environment, the expertise of a decision support system based on this model would be shared by several users. Implementation of such a distributed system, where intelligent objects could reside on any processor (assuming each user had his/her own PC, for instance) would pose problems of security and concurrency.

It is interesting to note that in a distributed implementation of the HEARSAY system (Lesser and Erman [1977]), a four- to six-fold increase in parallelism was realized. This increase was lower than expected and resulted from superfluous knowledge source synchronization. It was discovered that large areas of the blackboard were being locked in order to maintain data consistency, resulting in Knowledge Source interference. With system synchronization turned off, a fourteen-fold increase in parallelism was realized. Rigid synchronization was found to be unnecessary due to the self-correcting nature of the HEARSAY architecture, i.e., its data-driven computation coupled with a hypothesize-and-test paradigm. These results prove interesting in that the model in this thesis is in part based upon the HEARSAY

architecture.

SELECTED BIBLIOGRAPHY

- Ahlsen, M., A. Bjornerstedt, S. Britts, C. Hulten, and L. Soderlund. 1984. An architecture for object management in OIS. ACM Transactions on Office Information Systems 2(3):173-196.
- Aiello, N. 1983. A comparative study of control strategies for expert systems: AGE implementation of three variations of PUFF. Report No. HPP-83-33, Heuristic Programming Project, Stanford University, Stanford, California.
- Alter, S. 1980. Decision Support Systems: Current Practice and Continuing Challenges. Addison-Wesley, Reading, Mass.
- Ariav, G. and M. J. Ginzberg. 1985. DSS design: A systemic view of decision support. Comm ACM 28(10):1045-52.
- Balzer, R., L. Erman, P. London, and C. Williams. 1980. HEARSAY-III: A domain-independent framework for expert systems. Proc. 1st Ann. Natl. Conf. on Artif. Intell., AAAI pp. 108-110.
- Barr, A. and E. A. Feigenbaum (eds.). 1981. The Handbook of Artificial Intelligence. Volume 1. HeurisTech Press, Stanford, California.
- Barr, A. and E. A. Feigenbaum (eds.). 1982. The Handbook of Artificial Intelligence. Volume 2. HeurisTech Press, Stanford, California.
- Bennett, J. L., Ed. 1983. Building Decision Support Systems. Addison-Wesley, Reading, Mass.
- Bobrow, D. G. and B. Wegbreit. 1973. A model for control structures for artificial intelligence programming languages. Advance Papers 3rd Intl. Joint Conf. Artif. Intell. pp. 246-251.
- Bonczek, R. H., C. W. Holsapple, and A. B. Whinston. 1980. The evolving roles of models in decision support systems. Decision Sciences 11(2):339-56.
- Bonczek, R. H., C. W. Holsapple, and A. B. Whinston. 1981. Foundations of Decision Support Systems. Academic Press.
- Brachman, R. J., S. Amarel, C. Engelman, R. S. Engelmores, E. A. Feigenbaum, and D. E. Wilkins. 1983. What are expert systems? In Building Expert Systems. Frederick Hayes-Roth, Donald A. Waterman, and Douglas B. Lenat, Eds. Addison-Wesley, Reading, Mass. pp. 31-57.
- Cohen, P. R. and E. A. Feigenbaum (eds.) 1982. The Handbook of Artificial Intelligence. Volume 3. HeurisTech Press, Stanford, California.
- Corkill, D. D. 1979. Hierarchical planning in a distributed

- environment. Proc. 6th Intl. Joint Conf. Artif. Intell. pp. 168-175.
- Erman, L. D., R. D. Fennell, V. R. Lesser, and D. R. Reddy. 1976. System organizations for speech understanding: implications of network and multiprocessor computer architectures for AI. IEEE Transactions on Computers, Vol. C-25(4):414-421.
- Erman, L. D., F. Hayes-Roth, V. R. Lesser, and D. R. Reddy. 1980. The HEARSAY-II speech-understanding system: integrating knowledge to resolve uncertainty. Computing Surveys 12(2):213-253.
- Erman, L. D. and V. R. Lesser. 1975. A multi-level organization for problem solving using many, diverse, cooperating sources of knowledge. Advance Papers 5th Intl. Joint Conf. Artif. Intell.
- Filman, R. E. and D. P. Friedman. 1984. Coordinated Computing. McGraw-Hill, New York.
- Ford, N. F. 1985. Decision support systems and expert systems: A comparison. Info & Mgmt 8:21-6.
- Forgionne, G. A. 1986. Quantitative Decision Making. Wadsworth Publishing Co., Belmont, CA.
- Ghezzi, C. and M. Jazayeri. 1982. Programming Language Concepts. John Wiley and Sons, Inc., New York.
- Ginzberg, M. J. and E. A. Stohr. 1982. Decision support systems: Issues and perspectives. In Decision Support Systems. M. J. Ginzberg, W. R. Reitman, and E. A. Stohr, Eds. North-Holland, Amsterdam. pp. 9-32.
- Gorry, G. A. and R. B. Krumland. 1983. Artificial intelligence research and decision support systems. In Building Decision Support Systems. J. L. Bennett, Ed. Addison-Wesley, Reading, Mass. pp. 205-19.
- Gorry, G. A. and M. S. Scott Morton. 1971. A framework for management information systems. Sloan Management Review 13(1):55-70.
- Hayes-Roth, B. 1983. A blackboard model of control. Report No. HPP-83-38, Heuristic Programming Project, Stanford University, Stanford, California.
- Hayes-Roth, B. 1984. BB1: An architecture for blackboard systems that control, explain, and learn about their own behavior. Report No. HPP-84-16, Heuristic Programming Project, Stanford University, Stanford, California.
- Hayes-Roth, B. and F. Hayes-Roth. 1979. A cognitive model of planning. Cognitive Science 3:275-310.
- Hayes-Roth, B., F. Hayes-Roth, S. Rosenchein, and S. Cammarata. 1979.

- Modeling planning as an incremental, opportunistic process. Proc. 6th Intl. Joint Conf. Artif. Intell. pp. 375-383.
- Hayes-Roth, B. and M. Hewett. 1985. Learning control heuristics in BB1. Report No. HPP-85-2, Heuristic Programming Project, Stanford University, Stanford, California.
- Hayes-Roth, F. and V. R. Lesser. 1977. Focus of attention in the HEARSAY-II speech understanding system. Proc. 5th Intl. Joint Conf. Artif. Intell. pp. 27-35.
- Hayes-Roth, F., D. A. Waterman, and D. B. Lenat. 1983. An overview of expert systems. In Building Expert Systems. F. Hayes-Roth, D. A. Waterman, and D. B. Lenat, Eds. Addison-Wesley Publishing Company, Inc., Reading, MA.
- Hewitt, C. 1977. Viewing control structures as patterns of passing messages. Artificial Intelligence 8:323-364.
- Jones, A. K. 1979. The object model: A conceptual tool for structuring software. In Operating Systems: An Advanced Course. R. Bayer, R. M. Graham, and G. Seegmuller, Eds. Springer-Verlag, Berlin.
- Keen, P. G. W. 1980. Adaptive design for decision support systems. Data Base 12(1,2).
- Keen, P. G. W. and M. S. Scott Morton. 1978. Decision support systems: an organizational perspective. Addison-Wesley, Reading, Mass.
- Kornfeld, W. A. and C. E. Hewitt. 1981. The scientific community metaphor. IEEE Trans. on Systems, Man, and Cybernetics, Vol. SMC-11:24-33.
- Lenat, D. B. 1975a. Beings: knowledge as interacting experts. Proc 4th Intl. Joint Conf. Artif. Intell. pp. 126-133.
- Lenat, D. B. 1975b. Duplication of human actions by an interacting community of knowledge modules. In: Modern trends in cybernetics and systems. Proc. 3rd Intl. Cong. of Cybernetics and Systems, Bucharest, Romania, pp. 853-867.
- Lesser, V. R. and D. D. Corkill. 1979. The application of artificial intelligence techniques to cooperative distributed processing. Proc. 6th Intl. Joint Conf. Artif. Intell. pp. 537-540.
- Lesser, V. R. and L. D. Erman. 1977. A retrospective view of the HEARSAY-II architecture. Proc. 5th Intl. Joint Conf. Artif. Intell. pp. 790-800.
- Liskov, B., A. Snyder, R. Atkinson, and C. Schaffert. 1977. Abstraction mechanisms in CLU. Comm ACM 20(8):564-76.
- Little, J. D. C. 1970. Models and managers: the concept of a decision

- calculus. Management Science 16(8):B466-85.
- Moore, J. H. and M. G. Chang. 1980. Design of decision support systems. Data Base 12(1,2):8-14.
- Newell, A. 1969. Heuristic programming: ill-structured problems. In Progress in Operations Research. A. Aronofsky, Ed., Vol. 3, John Wiley and Sons, New York, pp. 360-414.
- Peterson, J. L. 1977. Petri nets. Computing Surveys 9(3):223-252.
- Reitman, W. 1982. Applying artificial intelligence to decision support: where do good alternatives come from? In Decision Support Systems. M. J. Ginzberg, W. R. Reitman, and E. A. Stohr, Eds. North-Holland, Amsterdam. pp. 155-74.
- Rich, E. 1983. Artificial Intelligence. McGraw-Hill Book Company.
- Rosenchein, J. S. and M. R. Genesereth. 1984. Communications and cooperation. Report No. HPP-84-5, Heuristic Programming Project, Stanford University, Stanford, California.
- Shaw, M and W. A. Wulf. 1977. Abstraction and verification in ALPHARD: defining and specifying iteration and generators. Comm ACM 20(8):553-64.
- Sprague, R. H. 1980. A framework for the development of decision support systems. MIS Quarterly, December.
- Sprague, R. H. and E. D. Carlson. 1982. Building Effective Decision Support Systems. Prentice-Hall, Englewood Cliffs, New Jersey.
- Unger, E. A. 1978. A Natural Model for Concurrent Computation. The University of Kansas, Ph.D. Dissertation.
- Unger, E. A. 1985. Intelligent data objects: a concept useful in networks. forthcoming.
- Winston, P. H. 1984. Artificial Intelligence. Addison-Wesley Publishing Company.

DISTRIBUTED PROBLEM SOLVING FOR DECISION SUPPORT

by

Mark C. Foehse

B. S., University of Missouri - Columbia, 1977

AN ABSTRACT OF A MASTER'S THESIS

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1986

ABSTRACT

Much has been written in recent years about decision support systems (DSS): applications, design methodology, and morphology. Over a similar time period the literature has also been full of articles describing expert systems (ES) and their design, applications, et cetera.

It is the perspective of the author that decision support systems and expert systems have more similarities than dissimilarities when viewed at a conceptual level. A hybrid system is therefore proposed based upon a blackboard model drawn from distributed artificial intelligence research. This system will be treated as a network of intelligent objects, each with its own inference mechanism. An abstract model is provided which describes how these intelligent objects could be linked in a three dimensional network for the purpose of decision support.

ABSTRACT

Much has been written in recent years about decision support systems (DSS): applications, design methodology, and morphology. Over a similar time period the literature has also been full of articles describing expert systems (ES) and their design, applications, et cetera.

It is the perspective of the author that decision support systems and expert systems have more similarities than dissimilarities when viewed at a conceptual level. A hybrid system is therefore proposed based upon a blackboard model drawn from distributed artificial intelligence research. This system will be treated as a network of intelligent objects, each with its own inference mechanism. An abstract model is provided which describes how these intelligent objects could be linked in a three dimensional network for the purpose of decision support.