CMU-CS-79-155

ORGANIZATION STRUCTURING: Designing Large Complex Software

Mark S. Fox

Department of Computer Science[1]
Carnegie-Mellon University
Pittsburgh, Pennsylvania

18 December 1979

[1]This research was sponsored by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 3597, monitored by the Air Force Avionics Laboratory Under Contract F33615-78-C-1551, and by Grant MCS77-03273 to The RAND Corporation from the National Science Foundation. In addition, the author was partially supported by a National Research Council of Canada Postgraduate Scholarship.

Abstract

This report investigates the problem of how to design and structure large, complex software. Software complexity has expanded beyond the point where current design methodologies and programming languages can effectively reduce complexity. Consequently, this report surveys a number of fields to ferret out useful design (structuring) concepts. Control structure research in Artificial Intelligence is briefly surveyed. Organization Theory, a field in management science, provides interesting approaches based on analyses of complexity, uncertainty and behavior. Economic Team Decision Theory provides an analytical approach to measuring alternative organizations of (rule-based) programs. Finally, a language is defined incorporating many of the concepts elicited from the surveyed fields.

Preface

This report describes the state of ongoing research (begun in 1976) in the area of program design. My interest in this area was sparked by my experience with Hearsay-II. In particular, I wanted to understand why the Hearsay-II architecture was an interesting and successful control structure. Much thought on this problem led to an investigation of how problem characteristics affect program organization. Initially, effort focused on uncertainty and its effects. During this exploration, Herb Simon introduced me to organization and economic decision theory. A survey of these areas from the nucleus of this report.

The primary goal of this report is to present a variety of views on how to organize large (distributed) programs and systems. I hope that these ideas, whose origins lie in organization theory, economic theory and aritificial intelligence provide food for thought. In the same vein, the organization structuring language presented in the final chapter is an attempt to understand how these concepts could be incorporated in a programming language. Though it lacks completeness of detail, the main concepts are displayed.

Many people have read and criticized drafts of this report. Herb Simon has continually pointed me in new directions and provided fruitful discussions. Karsten Schwans, Victor Lesser, Lee Erman, Anita Jones, John Kender, and Dave Notkin have also provided criticism on numerous drafts. Dan Corkhill, Scott Reid, Reid Smith, John Gashnig, Nico Habermann, Bob Sproull, Jeff Barnett, and John Ousterhout have also read various portions. They are in no way responsible for the faults remaining in this report.

Table of Contents

1. Introduction 1
2. Computer Science Approach To Design 3
   - 2.1 Programming Languages Approach to Design 3
   - 2.2 Artificial Intelligence Approach To Design 5
   - 2.3 Organization Structuring 9
3. Organization Theory 14
   - 3.1 Bounded Rationality 14
   - 3.2 Transaction Analysis 15
   - 3.3 Organization Structures 17
   - 3.4 Complexity 22
     - 3.4.1 Information Complexity 22
     - 3.4.2 Task Complexity 24
     - 3.4.3 Coordination Complexity 25
   - 3.5 Uncertainty 31
     - 3.5.1 Information Uncertainty 32
       - 3.5.1.1 Uncertainty of Intention 33
       - 3.5.1.2 Veracity Uncertainty 41
       - 3.5.1.3 Semantic Uncertainty 42
     - 3.5.2 Program (Algorithm) Uncertainty 43
       - 3.5.2.1 Decision Theory 44
       - 3.5.2.2 Adaptation 45
       - 3.5.2.3 Information Gathering 48
       - 3.5.2.4 Program Construction 48
       - 3.5.2.5 System Evaluation and Repair 50
     - 3.5.3 Environment Uncertainty 50
       - 3.5.3.1 Uncertainty and Hierarchies 51
       - 3.5.3.2 Uncertainty and Markets 53
     - 3.5.4 Behavior Uncertainty 53
       - 3.5.4.1 Motivation 54
       - 3.5.4.2 Conflict 55
       - 3.5.4.3 Opportunism 56
   - 3.6 Structural Continuum 56
     - 3.6.1 Complexity 56
     - 3.6.2 Uncertainty 58
   - 3.7 Observations 60
   - 3.8 Conclusion 61
4. Economic Team Decision Theory 62
   - 4.1 Economic Theory 62
     - 4.1.1 Single Person Decision Theory 63
     - 4.1.2 Team Decision Theory 63
     - 4.1.3 Network Team Decision Models 65
   - 4.2 Mapping Economic Theory onto Organization Structures 66
     - 4.2.1 Problems 67
     - 4.2.2 Mapping the Network Model 67

4.2.2.1 Single Node Network 67
4.2.2.2 Multi-Node Networks 68
4.2.2.3 An Example 68
4.2.2.4 Formalizing the Network 70
4.2.3 Interpretation of Numeric Functions 71
4.2.4 Adding Costs to Network Analysis 73
4.3 Interpreting Utility 75
4.4 Interpreting Optimality Conditions 76
4.5 Network Models for Linear Certainty Functions 77
4.6 Observations and Conclusions 80
5. ODL: Organization Design Language 83
5.1 Language Overview 83
5.2 Language Constructs 84
5.2.1 Information 84
5.2.2 Abstraction 85
5.2.3 Encapsulation 87
5.2.3.1 Goals 89
5.2.3.2 View 90
5.2.3.3 Ports 91
5.2.4 Data-Modules 91
5.3 Structuring 94
5.3.1 Loose Ends 97
5.4 Rationalizing the Language Constructs 97
5.4.1 Organization Structure Continuum 98
5.4.2 Complexity Reduction 98
5.4.3 Uncertainty Reduction 99
5.4.4 Behavior Uncertainty 99
5.4.5 Information Uncertainty 100
5.4.6 Algorithm Uncertainty 101
5.4.7 Team Decision Theory 101
5.5 An Example: Hearsay-II 101
5.6 Summary 108
ferences 109

1. Introduction

Current projections of software costs place them at about 90% of computer expenditures (hardware and software). Part of this differential can be explained by the decrease in hardware cost, and by the increase in software starts. But it is not just the quantity of software that explains the sizable cost. It appears that the tasks that are being attempted are growing in complexity. Though programming expertise has increased, complex systems are difficult to construct.

Why are these systems difficult to design? Because of system complexity. Human designers are limited in the amount of information they can absorb and manipulate. Hence they require a way of expressing the task that reduces information. Methods that abstract function and information in program design are required.

This paper is concerned with the problem of designing large complex programs and systems. It is concerned not only with the program primitives that facilitate the design process, but with understanding how the task affects the design solution. Current design concepts are syntactic in nature. "Step-wise refinement" and "complexity analysis" are defined without mention of peculiar task characteristics. They ignore the semantics of the task. A second problem with current design methods is their low level of description. They do not provide an adequate language for describing high level abstractions of large program designs. I view the current design languages as existing at the *control structure* level. I believe that in order to facilitate the design process higher level languages should be constructed. I call design at this level of abstraction *Organization Structuring*. The differences between control and organization structures are developed more fully in chapter 2. The goal of this study is to investigate the organization of large complex systems: to understand the merits of various organizational forms, and the effects of the task on the organization.

In chapter 2 two areas of computer science are reviewed briefly. The first area looked at is the design approach taken by Programming Systems. By definition, programming systems is concerned with the design problem. The second area is Artificial Intelligence (AI). Artificial Intelligence is a field whose varied goals include the understanding, via the computer paradigm, of tasks that require intelligence. Attempts at such tasks have resulted in programs whose size and complexity are staggering. Examples are Speech Understanding, Image Understanding, and Medical Diagnosis. The act of organizing such systems is quite arduous. As a result, AI has evolved a set of design primitives and methodologies to ease its task. An analysis of these two approaches results in the creation of the organization structure level of design and its primary primitives.

The next two chapters go outside of computer science to see how other disciplines deal with complexity.

Chapter 3 surveys the field of Organization Theory. Organization theory is a field of management science that attempts to analyse large complex organizations to ascertain problems and structural solutions. the problems of business organizations (i.e., uncertainty, complexity, and behavior) are remarkably similar to problems in computer programs. These problems are symptomatic of certain task attributes. The organization theory analysis is extended by an analysis of uncertainty in program data and algorithm.

Chapter 4 constructs an analytical model of program organizations. By mapping the network model of Economic Team Decision Theory on program organizations, a model for measuring the efficacy of alternative program organizations and the knowledge contained therein is produced. The model is applied to measuring the organization and knowledge of rule-based systems. The analysis raises interesting questions with respect to feedback, utility of results and data-certainty. In addition, the model allows the interpretation of decision theory theorems in a program organization context, and suggests changes to decision theory models.

Finally, chapter 5 can be viewed as a distillation of the preceding chapters. It presents informally a high-level, i.e., organization level, program design language that includes many of the ideas of the previous chapters. The language is used to represent a portion of the Hearsay-II speech understanding system.

2. Computer Science Approach To Design

In this chapter two approaches to program design are presented. The first is the programming system approach. The second is the artificial intelligence approach. Following these two brief samplings some basic program design primitives suggested by the two approaches are described.

2.1 Programming Languages Approach to Design

The study of the programming process has been a primary interest to to software scientists for some time. They are interested in the most effective way of creating programs -- effective in terms of efficiency in algorithm and reduction of programming effort.

Knuth (1974) describes programming as an "Art". This seems to be empirically true. Introductory courses in computer science teach programming by example, the Guild method. None the less, some practitioners of the art of programming have attempted to explicate their view of the programming process. They have succeeded in providing valuable programming heuristics.

Parnas (1972a; 1972b) has contributed the idea of *modules* and their *connections*. A module embodies a major concept in the program, while its connections specify how it interacts with other modules. An important characteristic of a module is that it *conceals* all but what is necessary for another module to use it. In other words, the knowledge one module has of another is minimal. The fewer assumptions that modules make of other modules, the less *strongly-connected* the system is. Weakly-connected systems are important because they allow partitioning of the programming task and facilitate system modifications. A module does not use any knowledge of how the internals of another module are constructed; internal changes are transparent.

Wirth's (1971) idea of "program development by stepwise refinement" is another valuable heuristic. He believes programs should be designed by first specifying the workings of the program at a high level of abstraction. This abstraction is *partitioned* into steps and each step is *refined* into a more specific statement of the step's task. Partitioning and refining is repeated until the specification is at the programming language level. This is a *top down* approach to the development of a program: specify the idea and continually refine it, all the while delaying the specification of detail until the context is small and well-defined. The specification of one step is (usually) independent of another.

The work of Wulf et al. (1977), Liskov et al. (1977), and Dahl & Hoare (1972) is centered around facilitating program construction through the use of *abstraction* and/or *hierarchies*. A module, class, form, or cluster contains both data and procedure. Building on the work of

Parnas, only portions of the module can be seen and used by other modules. New modules are defined using previously-defined modules as primitives. A hierarchy of modules is built so that the top-most modules can be combined to solve the problem.

Continued work in the area of modularization has led to a more explicit definition of how modules interact. MIL (DeReemer & Kron, 1976) is a module interconnection language which describes subsystems and the resources transferred among them. Mesa (Mitchell et al, 1978) develops the idea of module interfacing by the selective importation and exportation of information. In addition they develop a module interconnection language. Tichy (1979) has continued research in interconnection languages to include both information sharing descriptions and version tracking for proper system compilation. His system dynamically builds and maintains a model of module attributes and connections.

An orthogonal but complementary approach to program development has been taken by Newell et al. (1977) in the L* system and Teitelman (1975) in the Interlisp system. Part of the philosophy (i.e., interaction, and design strategy) behind this system is that the programmer should be provided with a complete environment (e.g., interactive, proper software tools) in which to design his program. This environment must provide the means to build, execute, debug, and modify programs interactively. The use of the complete environment should significantly decrease program development time by easing each step of the process..

Teitelman's (1977) approach to system development is to "human-engineer" the complete environment. Through the use of sophisticated graphics (and a modified monitor), the user can easily manipulate many aspects of the complete programming environment including multiple process suspension and continuation.

An analytical approach to program design has been taken by Chanon (1973) and McLure (1978). Each attempts to measure complexity of module interaction. Depending on the definition of interaction, a different measure can be derived. By reducing interaction, it is hoped that complexity is reduced. This is an example of *near-decomposability* which was first described by Simon (1962) and applied by Alexander (1965) to architectural design. Defining good measures seems to be the main problem hindering this work.

How can these varied contributions to the "art of programming" be summarized? The techniques of Parnas, Wirth, and Wulf et al. can be viewed as heuristic methods applicable to the design process. Newell and Teitelman's ideas are concerned with both the implementation process and the iterations between design and implementation. DeRemer and Kron, Mitchel et al., and Tichy are concerned with the explication of module interaction. Yet all of these ideas have one key feature: they reduce the *complexity* of the programming process. By

information hiding, problem reduction (partitioning), interconnection definition, or easing the build, execute, debug, modify cycle, the complexity of building programs has been significantly reduced, allowing the attack of bigger and more ambitious problems.

2.2 Artificial Intelligence Approach To Design

Section 2.1 dealt with the "programming system" approach to program development. Typical applications of these ideas are usually small in size and complexity. They are small enough that the vagaries of large programs do not appear. It is valuable to review some of the recent work in artificial intelligence to understand the complexity of the tasks and the corresponding complexity and magnitude of the programs designed to solve them.

Hearsay-II (Erman, 1975; 1977) is a system designed to understand connected speech[2]. Utterances, without artificially introduced pauses between words, are spoken to the system. Hearsay-II must interpret, understand, and reply to the utterance. The current version of Hearsay-II retrieves and answers questions about abstracts stored in its data base (Hayes-Roth et al, 1977c).

The process of understanding utterances requires the application of many sources of knowledge: acoustic, syllabic, lexical, prosodic, syntactic, semantic, pragmatic, etc. Each source of knowledge can be used to interpret the utterance at its own particular level of representation. Each source of knowledge only partially represents the knowledge a human brings to bear when parsing speech. These sources represent the state of the art of our knowledge of the speech understanding process. Because of the incompleteness of the knowledge, the understanding process is saturated with error. Thus speech understanding is a search in a large space of possible interpretations for the utterance that best fits the input data, i.e., the speech wave form.

The design of a speech understanding system must allow the integration of sources of knowledge in such a way that they may gracefully interact. The errorfulness of the processing requires that the program have the ability to redirect its attention whenever the current best interpretation of the utterance proves implausable.

The approach taken in Hearsay-II is as follows: The knowledge in the system is represented in *Knowledge Sources* (KSs). Each KS contains a separate portion of knowledge such as Syntax and Semantics: SASS (Hayes-Roth Mostow & Fox, 1977); Lexical: POMOW (Smith, 1976); Semantics: SEMANT (Fox & Mostow, 1977). The knowledge is integrated by

[2]See (Reddy, 1976) for a good introduction to the problem.

allowing the knowledge sources to communicate via a *Blackboard* (BB). The BB is a common, dynamic data structure. Each KS can be viewed as an expert in its particular field and contributes to the "discussion" among the experts by reading and writing information on the BB. The mode of BB interaction is *Hypothesize and Test* (Newell, 1969). Each KS can either place an *Hypothesis*, describing its interpretation of BB data (i.e., other hypotheses), on the BB, or test (i.e., accept or reject) BB hypotheses produced by other KSs. As mentioned above, the knowledge in the different KSs can be used to interpret the utterance at different levels of representation. Specifically, the *Levels of Representation* (knowledge) form a hierarchy. Each level is built upon a lower level. The lexical level is built upon the syllabic, and the syntactic upon the lexical. The job of a KS is to construct an interpretation (hypothesis) at its level of expertise by postulating (or testing) hypotheses constructed from hypotheses at a lower level or by elaborating hypotheses from a higher level.

The processing of the system is *Data-Directed.* It is directed by the current state of the BB data[3]. Each KS can view BB hypotheses at its level(s) of expertise. Whenever a change is made to an hypothesis or the BB by a KS, other KSs react through further hypothesization and testing. At any time there are many possible KSs capable of executing. The choice of which KS to execute is controlled by *policy modules* and the *scheduler.* Together they provide a *focus of control* mechanism (Hayes-Roth & Lesser, 1977) capable of directing the system's attention to the currently best hypotheses, or re-directing the system when the current hypothesis proves unfruitful.

Figure 2.1 shows the organization of the Hearsay-II system. Figure 2.2 shows the blackboard hypotheses for interpreting the utterance "Tell me about beef".

PUP6 (Lenat, 1975) is a theory of knowledge organization and representation applied to automatic programming. All the knowledge necessary to produce a program, (specifically, a concept formation program) is stored in BEINGS. A BEING can be viewed as an expert in a particular knowledge area (similar to a knowledge source). A program is built by the BEINGS (experts) carrying out a group dialogue between themselves and the user. The dialogue is composed of BEINGS asking questions about the task that is to be programmed. BEINGS, expert in the area of the question, reply by asking more detailed questions or by actually writing code. All questions fall into one of many predefined categories. A BEING replies if it has information associated with the question's category. Through this process of highly structured interaction via communicating modules (BEINGS), the problem is reduced from an imprecise statement to a working program by the inference, specification, and deferral of

[3]Each hypotheses is rated. It is a function of the ratings of the the hypotheses it is constructed from and the knowledge used in the construction.

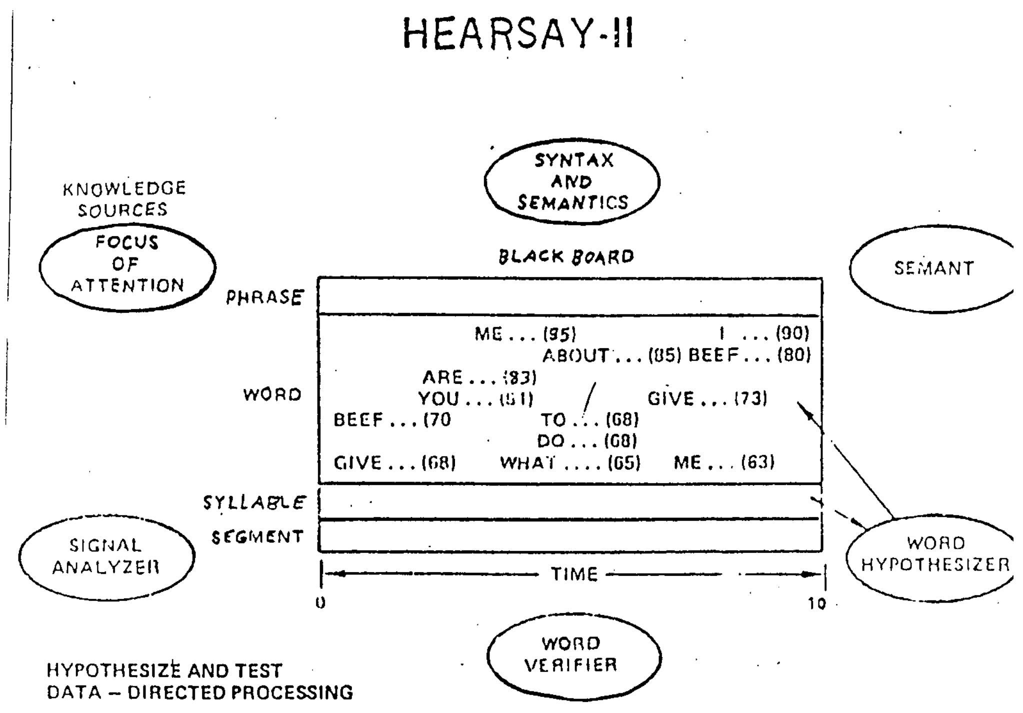


FIGURE 2.1

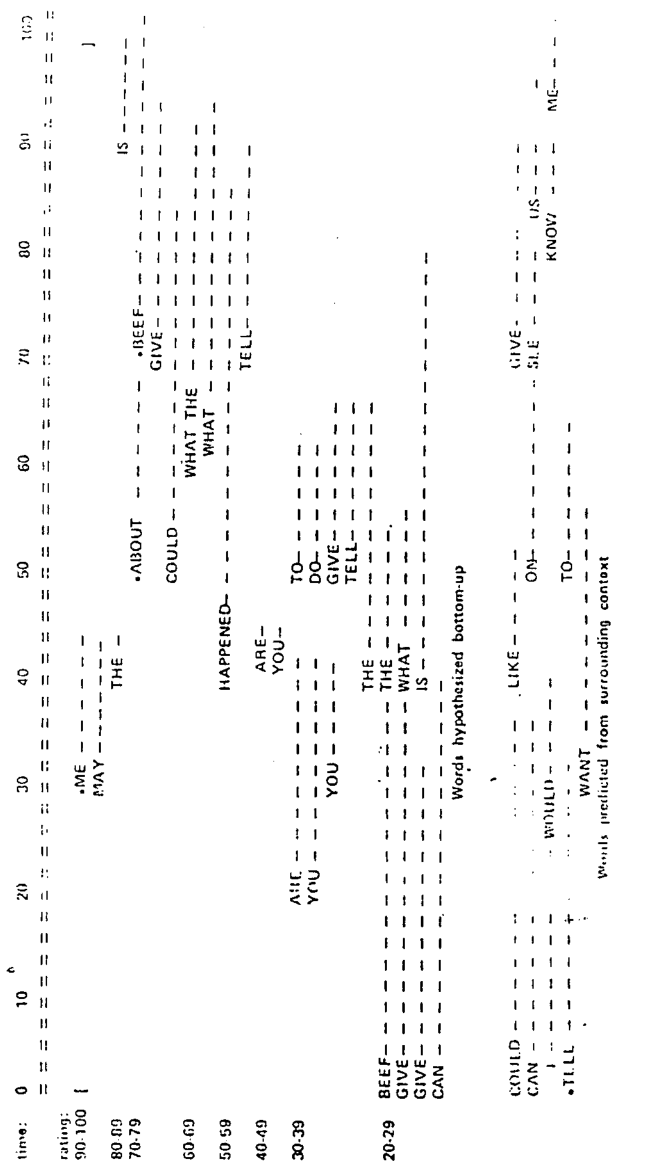


FIGURE 2.2

relevant information. It can be viewed as the step-wise refinement of a problem to a solution. The organization of a PUP6 system is heterarchical. A BEING does not know of the existence of other BEINGS. All questions are posited to all assembled. The answers to a question are rated and ordered with only the best BEING used.

SU/p (Nii & Feigenbaum, 1977) (Engelmore & Nii, 1977) is a system that infers three dimensional models of protein molecules from an electron density map derived from x-ray diffraction data. Its organization is based on Hearsay-II. The system is organized around knowledge sources communicating via a multiple-level representation blackboard using the hypothesize and test paradigm.

Differences are: knowledge in a KS is represented by *production rules*, and the BB has been extended to contain processing (event) history, and a problem list. A problem is the information needed by a KS to draw an inference. The most significant difference is the formalization of control. Hearsay-II used policy modules combined with the scheduler for control of execution. SU/p utilizes a new set of KSs as control KSs. Three levels (types) of control KSs are used: 1) *Hypothesis-formation* KSs containing the domain knowledge for creating and verifying hypotheses, 2) *Activation level* KSs deciding which hypothesis-formation KS to execute, and 3) *Strategy level* KS which analyses processing to decide region of processing and activation level KS to execute. Hence the control is strictly hierarchical. Knowledge in the control KSs are also represented by production rules.

What we see emergent from these three systems is the partitioning of the problem into modules (KSs, BEINGS). Each module has capabilities, e.g., question-asking and self-control, beyond those anticipated in regular programming languages. Secondly, the processing of these systems is complex, using methods similar to human problem-solving. Thirdly, these systems rely upon the representation and storing of multiple solution paths at various levels of abstraction.

2.3 Organization Structuring

The design process, as characterized in section 2.1, is at a low level of the programming task. Structures such as procedure calls, repetition, queues, stacks, are the primitives being used. These primitives have proven useful for the design and construction of low complexity programs. For more complex programs, different primitives -- in fact different theories -- which can deal with the complexity seem necessary to aid the design process. A good example is the area of Operating Systems. The theory and languages used reflect the difference in complexity and thus the level of design necessary to complete the task. Operating system designers talk of *processes* (Horning & Randell, 1973) where programming systems talk of *procedures, mail boxes* versus *variables, data storage* versus *data structures*, etc. The same problem has been faced in AI. Speech Understanding systems (Reddy, 1976) are so complex that typical programming languages are not sufficient. New theories of organizing control and information, and a language to implement these theories have been found necessary (e.g., Interlisp (Teitelman et al., 1975), Planner (Hewitt, 1971), KRL (Bobrow & Winograd, 1977), and Hearsay-II (Erman, 1975; 1977)).

What have we learned from the design and construction of artificial intelligence programs? The design of artificial intelligence systems is clothed in terms that seem *problem-oriented*. Terms such as "communicating experts", "blackboard structures", and "communication channels" are used to describe system features that are similar to, if not equivalent to, problem features. Program and data are merged into knowledge sources, levels of representation, modules, etc. as defined by the problem. Similar ideas appear in abstract data types of Alphard and CLU.

What we see emerging from this morass, is the need to design the system at a level of abstraction in which modules, their goals and intentions, and their interactions are of prime importance. The major foci are how a system of modules and channels is organized, and the behavior of the organization with respect to the processing goal is defined.

We propose that:

> This level of design is distinctly different and separate from the programming language level of design discussed in section 2.1.

We view the latter as design guided by the language, whereas the former is design guided by the problem. We call the former *Organization Structuring*.

It is important to differentiate between organization structuring and control structure. Newell defines a control structure as

> The organization of primitives (memories, encodings, and primitive operations) into an effective processing of knowledge (Newell, 1973b).

This definition can be interpreted as a microscopic view of computation: You cannot analyse the computation to any finer level of detail. Nor can you understand the particular use of a primitive, with respect to the processing goal (e.g., understanding speech, image analysis, medical diagnosis), without understanding the context in which it is executed. Organization structuring is based on a macroscopic view of computation; the primitives focus on the complex modules described earlier. These primitives allow the representation of task-dependent information which describe the intent of the module at various levels. The primitive is a combination of mechanism and information that can be understood outside of its context (context-free). The "end" (goal) of a module can be related easily to the processing

goal of the organization. This allows us to redefine Newell's definition of control structures:

> The organization of primitives into an effective processing of knowledge where the interpretation of the effects of a primitive cannot be related to the task (problem, goal) at hand without understanding the context under which the primitive is executed (context-sensitive).

The line dividing control structures and organization structures is fuzzy. The two concepts should be thought of as being at ends of a continuum of task-related combination of primitives; control structures near one end and organization structures at the other (see Fig. 2.3). Simply, organization structuring can be viewed as a "program design language" whose primitives correspond to major (high-level) portions of a program. For example, the kernal in Hearsay-II provides a programming language whose primitives are knowledge sources, stimulus-response frames, communication channels, data bases (BB), wake-up mechanisms, etc.. These primitives map directly onto a high level description of a speech understanding system. This provides a way of describing and organizing the program at the organization level.

Organization structuring contains three major structures:

Module:	A grouping of knowledge and mechanism that is similar, if not equivalent to, an area or portion of the problem.
Communication Channel:	A method that a process uses to transfer data or control information to other processes.
Data Module:	A structure in which large amounts of information are stored.

Let us step back and look at the word "module". The sense of "module" being used here has more in common with business organizations than operating systems. A module may not only have the characteristics of a knowledge source in Hearsay-II (a single embodiment of one or more mechanisms that recognizes the context under which it is applicable), Actors (Hewitt, 1973), and PUP6 but may also have some of the characteristics of a unit in a business organization (example characteristics: cognitive limits, motivations to participate, organizational conflict) (March and Simon, 1958). Early work in Organization Theory took a non-behaviorist "automaton model" view of organizations. Such simplistic models did not account for the behavior of complex business organization. The smallest unit in an organization is human and thus cannot be simply modelled. I presume that "automaton models" could be applied to the computer software systems of today. The important idea that is just beginning to emerge is that the complex tasks that AI and other areas of computer science are trying to solve require well organized systems of modules where each embodies not only one or more mechanisms (procedures) but the information to allow it to decide when the application of the mechanisms are "necessary".

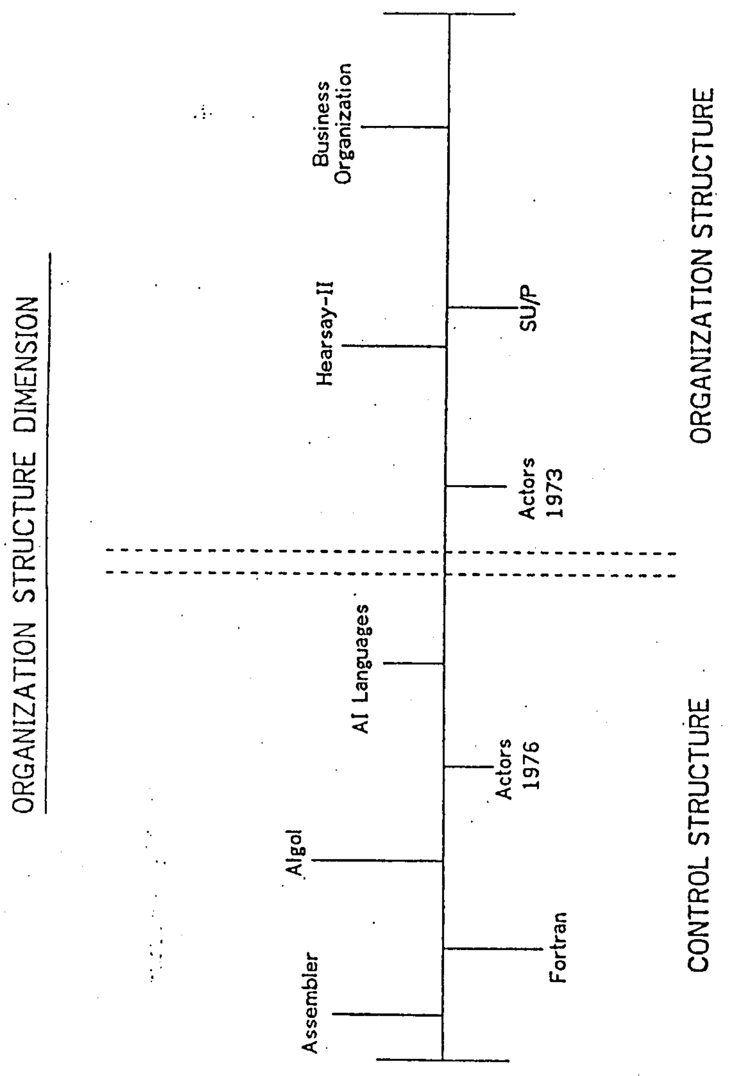


FIGURE 2.3

In order to reduce the complexity of program design, we must understand the relation between the task and the organization structure. What are the various types of modules, communication channels, and data bases and how do they relate to the problem? The first step along the road to such and understanding is the availability of an adequate language to describe organization structuring. This language must contain the primary features of organizations. Second, we must analyse the relationship between the problem attributes and the organization. These issues are discussed in subsequent chapters.

3. Organization Theory

"Organization structure consists simply of those aspects of the pattern of behavior in the organization that are relatively stable and that change only slowly." (March & Simon, 1958, P. 170)

Organization theory deals with the structuring and coordination of large, complex business organizations. Given an organizational goal such as building airplanes, how does one structure and coordinate the organization (company) to satisfy the goal and constraints such as time and cost? Considering the number of people necessary to perform the task, and the amount of materials and machinery to be used, the problem is quite complex. Many attempts at solving this problem have been less than satisfactory. Thus, organization theorists have attempted to discover and catalogue solutions to the major problems affecting large organizations.

Why is organization theory of interest to computer science? As problems become large and complex, program organizations begin to resemble business organizations. The knowledge sources in the Hearsay-II system, and the scheduler, memory manager, etc. in an operating system, each resemble functional units in organizations. Each encompasses both programs of action (decision and control) and information (upon which decisions are based) to carry out its task. Similarily, a unit in an organization, such as the purchasing department, also has programs of action and the information on which to base its decisions. The PUP6 system uses group dialogue similar to a group of managers in a problem solving session. Also ACTORS (Hewitt, 1973; see sec. 2.3), individuals in an organization, and departments themselves, must be motivated to participate in some action or goal. In both business and computing we are faced with the problem of *absorbing* large bodies of information, *deciding* what actions to take based on this information, and *coordinate* our actions and resources, to achieve the action's goal.

This chapter attempts a technology transfer. It is hoped that by surveying organization theory, new concepts for structuring organizations may be learned and applied to program organizations. More important a different way of looking at program design is presented. It is hoped that this view provides new insights into the design process.

3.1 Bounded Rationality

An obvious starting point in the understanding of organizations would be the analysis of the workers that comprise the organization. Oddly enough, classical organization theory took an automaton view of the processes in an organization. Humans were viewed as machines

and research centered around time and method studies: What is the best program of action that can be executed by a worker at his job? E.g., how fast can a nut be screwed on a bolt? March and Simon (1958) were among the first to take a behavioral approach to the analysis of organizations. The behaviorist approach treats workers as humans with human limitations, strengths, and weaknesses. The key concept, defined by Simon (1957), underlying the behavioral approach is called *bounded rationality*. Simon states:

The capacity of the human mind for formulating and solving complex problems is very small compared with the size of the problems whose solution is required for objectively rational behavior in the real world - or even for a reasonable approximation to such objective rationality.

Bounded rationality implies that both the information a person can absorb and the detail of control they may wield is limited. Because of this limitation, Simon states:

Only because organized groups of human beings are limited in ability to agree on goals, to communicate, and to cooperate, that organizing becomes for them a 'problem'.

As tasks grow larger and more complex, means must be found to effectively limit the increase of information a person sees and the complexity of coordination. Bounded rationality is a prime factor in the evolution of multi-person organizations from an unregimented group to more structured alternatives.

Bounded rationality explains the evolution of the standard structure of organizations called the *Information Processing Model* (March & Simon, 1958). This model is hierarchical with rules defining the program (response) to be executed for each problem (stimulus). These programs are defined during organization construction and are based on the processing goals of the organization.

Bounded rationality can be directly interpreted in the programming environment. A processor can execute only a limited number of instructions per second. This limits the amount of information a processor may process and the amount of control it may exercise within a given time period. Hence, programmed systems, whether centralized or distributed may exhibit bounded rationality symptoms when capacities are exceeded.

3.2 Transaction Analysis

Both organization theorists and economists are concerned with the analysis of organizations to elicit relevant signs and their causes. One problem is is determining the proper level of analysis. Organizations are represented typically by organization charts; boxes represent departments (units) or offices; attention is focused on the contents of a box. Recently, researchers have focused on the connections between the boxes, and in particular, the *transactions* that occur. Transactions take on a rather broad definition in this study. They encompass normal contractual agreements, communication of information, monitoring, delegation and control, and most other activities that require interaction among participants within an organization or market.

One result of transaction analysis is Simon's (1962) theory of *near-decomposability*. The viability of an organization requires that the number of transactions amongst units be less than within units; bounded rationality limits the quantity and complexity of transactions taking place.

Williamson (1975), in what he calls the *organizational failures framework*, believes that to understand the efficacy of alternative organizations (e.g., market vs hierarchy) requires an understanding of transaction characteristics. The organizational failures framework attempts to determine environmental and human factors that pose transactional problems. When environmental factors such as uncertainty and small-numbers exchange-relations, combine with human factors, such as bounded rationality and opportunism, transactional problems such as information impactedness and first-mover opportunity may occur.

Information impactedness is a differential of information between parties of a transaction. Impactedness may be due to bounded rationality considerations because of the amount of information, inavailability of information due to one party's inability to communicate, or a party's deliberate hiding of information. Impactedness would be of little concern if the cost of achieving parity were not prohibitive in most cases.

Opportunism occurs when a party in a transaction takes advantage by making self-disbelieved threats or promises, or withholds information. The opportunistic party secures a contract that is less favorable to the other party than might be obtained otherwise. Information impactedness is a recurring condition for opportunistic behavior.

Small numbers is a market condition where the number of market participants is small, circumventing the marginal pricing behavior of competition. Contracting under small numbers condition may result in opportunistic behavior due to participants lack of competitive pressure.

First mover opportunity occurs when a person or organization has idiosyncratic knowledge of a particular function unattainable (due to cost and information impactedness) by other market participants. As a result, a small numbers market condition results enabling opportunistic behavior. A first mover condition can appear when a person in an organization attains idiosyncratic knowledge of their particular job, or an initial contractor attains idiosyncratic knowledge of the contracted job. In subsequent contracting for the same or

similar job, or searches for new personnel, the previous person or contractor has a considerable advantage due to their superior (idiosyncratic) knowledge.

Information impactedness is a condition that appears often in program organizations. Information is transformed continually with computer systems, resulting in omission or substitution of important information. Reducing impactedness requires the wider availability of information. But bounded rationality must be kept in mind. Even though information is available, resource limitations may preclude its processing by interested modules.

Transaction analysis asks for a better awareness of what resources are consumed in transactions. Problem decompositions result in varying communication behaviors. The resource costs involved should be modeled and analysed to discover better decompositions.

Transaction analysis results have shown that transaction characteristics play an important role in the vitality of an organization. Greater emphasis should be placed on the analysis of transactions in distributed systems.

3.3 Organization Structures

Before analysing what attributes of a task affect system structure, it is useful to obtain a picture of the space of organization structures found in business, and analogous structures in software systems. The following organizations represent points on the continuum of structures.

Single-Person

The simplist organization is the *single-person*. The person performs the means to achieve a goal, reacting to information and the environment when necessary.

This simple organization is best modelled by simple program on a uniprocessor. No procedures are available, all code is executed inline. Like a single-person, this system is resource-limited. Only so much processing power is available. As the job grows, it begins to swamp capacity.

Group

As the means requires more resources (mental or physical), the size of the organization increases requiring more complex control and an increase in information processing capabilities. Organizational forms such as a *group* result. A group allows the coordination of individual members to achieve a shared goal. The task is divided up and sub-tasks allocated to members who are best able to execute them. Coordination in a group is achieved through mutually agreed upon decisions. To achieve this type of coordination group members must share all available information. They must understand it, and be able to communicate their views. Finally they must arrive at a decision that satisfies all. Each step in the coordination problem is a series of transactions. The cost of a transaction is dependent upon what is being transacted.

The program model of a group assumes independent modules on separate processors. A group allows the bringing to bear multiple participants of varying specialties to solve a problem. Assuming an existing module decomposition, organizational considerations for groups are:

1. The sharing of a communication language so that information is understand by all participants.
2. The existence of communication channels among participants or the access to shared data base so that all informaton is made available.
3. The acquisition of proper authority. The effectiveness of a group depends on the authority they have to access information and initiate organization change.

Underlying these considerations is the problem of information impactedness and opportunism. Participants (modules) that do not share the group's goals may change or omit information in an opportunistic fashion.

Programs that exhibit group problem-solving organizations have already been constructed. The PUP6 system utilizes group problem-solving to construct a program. But it is at one extreme of the organizational structuring. It is totally *Heterarchical*. Hearsay-II was initially conceived as heterarchically organized. The inclusion of Focus of Attention (Hayes-Roth & Lesser, 1977) (FOCUS)[4] resulted in a hierarchical organization with FOCUS as the primary decision maker; i.e., the decision as to what KS was to execute next was made by FOCUS[5].

A group is limited by size. As size increases so does complexity of information, decision making and control. It also becomes harder to monitor and control motivation and deviational behavior.

[4]FOCUS is not a KS, but a combination of policy modules and the scheduler. For the purposes of this paper, we describe it as if it were a KS.

[5]The selection process was indirect. Focus adjusted priorities of KSs, hence changing their position in the scheduling queue.

Simple Hierarchy

As the size of the group increases, collective decision becomes costly. Cost of information distribution and communication to converge to a common decision increases. Hence a *simple hierarchy* evolves. Coordination in a simple hierarchy is vested in a single decision maker. Complete information must be made available <u>only</u> to that person, and they must have the <u>authority</u> to effect changes in the organization's behavior. A group evolves to a simple hierarchy instead of separate single-person organizations because the transaction costs of coordinating the subdivided tasks is too high in the single-person case. Proper coordination, implying authority relations, and distribution of information is required for the organization to be effective.

A simple hierarchy can be viewed in two ways. First, as a single procedure that has a set of sub-procedures it can call to carry out a variety of functions. Or as a set of independent modules whose efforts are coordinated by a single decision making module.

In a hierarchy, it is assumed that all possible stimuli can be enumerated, identified, and categorized. An appropriate response is programmed for each category. Pre-categorization and response generation enables the application of classical division of labor techniques for problem decomposition. Hence hierarchical structures, such as the Information Processing Model prevail with little exception handling being necessary.

Uniform Hierarchy

Multiple levels of management are created to insure proper and centralized decision making. Each level of the hierarchy acts as a filter on the information and decisions that are propagated up the hierarchy. Decisions are made at the lowest level in the hierarchy that has both the information to make the best decision and the authority to execute it.

As in a simple hierarchy, the uniform hierarchy can be viewed as a hierarchy of procedures or independent modules. The purpose of the hierarchy is the efficient execution of the task at hand. Proper execution requires proper control. If a hierarchical organization is viewed as a tree with the root at the top, and the leaves at the bottom, control flows down the hierarchy. Initially abstract, control information is elaborated as it passes each level. The process of elaboration is well-defined since actions are prescribed. But control is not all that a hierarchy shapes. Information is flowing in the opposite direction. It is not the case that the control hierarchy is always the same as the information hierarchy, but control cannot exist (for long) without information, and vice-versa.

Since information processing capacities are limited, saturation must be avoided. Information absorption allows diverse information sources to be summarized combined, and brought to bear in the decision process. This process of collecting and summarizing is dependent upon the creation of levels of representation of information that coincide with functional levels in the hierarchy[6].

Multi-Divisional Hierarchy

As the uniform hierarchy increases in size and the number of products, powers of control are impaired resulting in transactional diseconomies. With multiple products being produced, competition for resources arises among units. The problem of allocating resources, so that enough are available, and the products are produced on schedule is quite complex.

One approach to reducing these effects is the *multi-divisional hierarchy*. The organization is split along product lines. Each division in full control of the tactics involved in producing their product. Hence control is situated locally where the information that enables control is available. Strategic control is vested in an elite staff assigned to a general office. The general office is concerned with strategic planning, appraisal and control including resource allocation. Separation of general office from product divisions reduces product identification and divisional persistence. The general office is better able to appraise efficiency of divisions without bias and can also access information at less cost than the market. Cash flow allocation to high yield uses and incentive rewarding are facilitated due to better appraisal abilities.

An airline reservation system can be viewed as a multi-division organization. It is a single system that provides a variety of services, but share the same goal, airline efficiency. Distributed problem-solving systems also display a multi-division organization. Distributed problem-solving (Lesser & Corkhill, 1978) is required when an organization and/or its stimuli is physically decomposed and distributed. This requires the physical separation of modules or duplication of the organization at the distributed sites. Distributed problem-solving requires the distributed modules to cooperate in achieving their shared goal. If processing is independent, then global coordination is not necessary. Hence, disjoint heterarchical systems or techniques such as single-level relaxation (Zucker, 1976) are applicable. If local processing is not independent, that is, interpretation at one site is dependent upon other sites, then information must spred through the system and/or global control exercised. Hierarchical systems for instituting levels of control are applicable.

[6]The Hearsay-II system uses multiple levels of knowledge. Each built upon the previous levels. This is not always necessary. From one level of representation different and sometimes uncomparable levels (abstractions) are created, to be combined at another level of abstraction. The levels of abstraction look like a lattice with the maximum being the most abstract summarization and the minimum being the source input.

Price (Market) System

The Price System is an alternate organizational form. The price system relies on the existence of a *market*, a heterarchy of organizations, in which a number of disjoint organizations are available to produce a product or supply a service. Actions are initiated after the successful negotiation of a *contract*. This system eliminates all forms of control between units. All communication is contained in a contract to purchase some product or service. Control is exerted through the *price* of the product. Price (should) reflects the marginal cost of the product. The assumption is that through marginal pricing of goods, all resources will be utilized without waste. If a product is priced too high, it will not purchased, if too low, the unit will go bankrupt.

With the introduction of the price (or market) system concept, an organization does not have to create a new unit for each new function, but can contract for the function in the market place. The next step in the evolution of an organization is a *collective organization*. The hierarchy is split into separate organizations who cooperate to achieve a shared goal. In one sense a collective can be viewed as a set of organizations that share long-term contracts.

The next step in successive reduction of control and information flow is the introduction of competition. Competing approaches to goal achievement is allowed (in the market place) with many organizations available to achieve any goal. Hence, each organization persues its own goals which correspond to another organization's needs. This is the general market situation. Services are contracted for in the market-place for short or long periods of time.

The price system approach to program organization requires a formalization of communication and contract writing. A well defined classification of contract types, their guarantees, and their costs. A language for contracting is required. In addition, market wide reports of performance by contractors should be available for participants to ascertain the eligibility of alternative contractors.

Recent work has demonstrated the feasibility of computer networks supporting a market organization. In their discussion of High-Level Protocols for networks, Sproull and Cohen (1978) describe a Network Plotter Protocol (NPP). NPP is a language for describing graphics plotting tasks which allows market participants to communicate about tasks. The work of Smith (1978) defines a protocol for contracting among modules. Hence the mode of market interaction, contracting and bargaining, and the language for describing the task, NPP, have been created for the task of contracting for plotter printings by market participants.

The following two sections examine the problems of task *complexity* and *uncertainty*. It is shown that large amounts of complexity and uncertainty can be detrimental to organizational effectiveness. A variety of structures are proposed for reducing their effects. Resulting organizational structures approximate points on the structural continuum, admitting an approximate mapping from complexity and uncertainty to the structural continuum.

3.4 Complexity

A major factor affecting the organization of systems is *complexity*. Complexity is defined as excessive demands on rationality. That is, task requirements exceed current bounds on rationality. For example, a manager receives more information than he could possibly read, or must coordinate more workers than he could possibly coordinate. In the following, three types of complexity: information, task, and coordination are described.

3.4.1 Information Complexity

Bounded rationality limits the amount of information a human or processor may process within a given time period (or other resource constraint). Information becomes too complex when it requires more processing than available in order to be properly analysed and "understood". Ways must be found to reduce the complexity of information so that humans and processors can be more effective.

How is the amount of information reduced? By *abstraction* and *omission*. Abstraction is attained by use of several levels of representation. A purchasing manager in the plane company does not care how much material is used every minute in a department, nor how it is used. He is interested only in gross usage of materials. Material/minute is a detail that can be abstracted by using material/day or material/week while its usage can be ignored completely, since the former is actually an abstraction of it. A second approach to information reduction is computer based summarization techniques. Whether statistical or graphical in nature, information can be reduced to a few meaningful parameters.

Examples of these techniques can be found in the the Hearsay-II and BASEBALL (Soloway and Riseman, 1977) systems. In Hearsay-II information is represented at many levels (Fig. 3.1). A KS that makes a decision at the syntax level *only* uses information at the syntax and lexical levels, without going into the more detailed levels. To do so would require a greater information processing capacity and the ability to interpret information at those levels. Program organizations provide the unique opportunity to formalize communication. Languages of communication, abstraction procedures, control commands, etc. can be completely described in such systems.

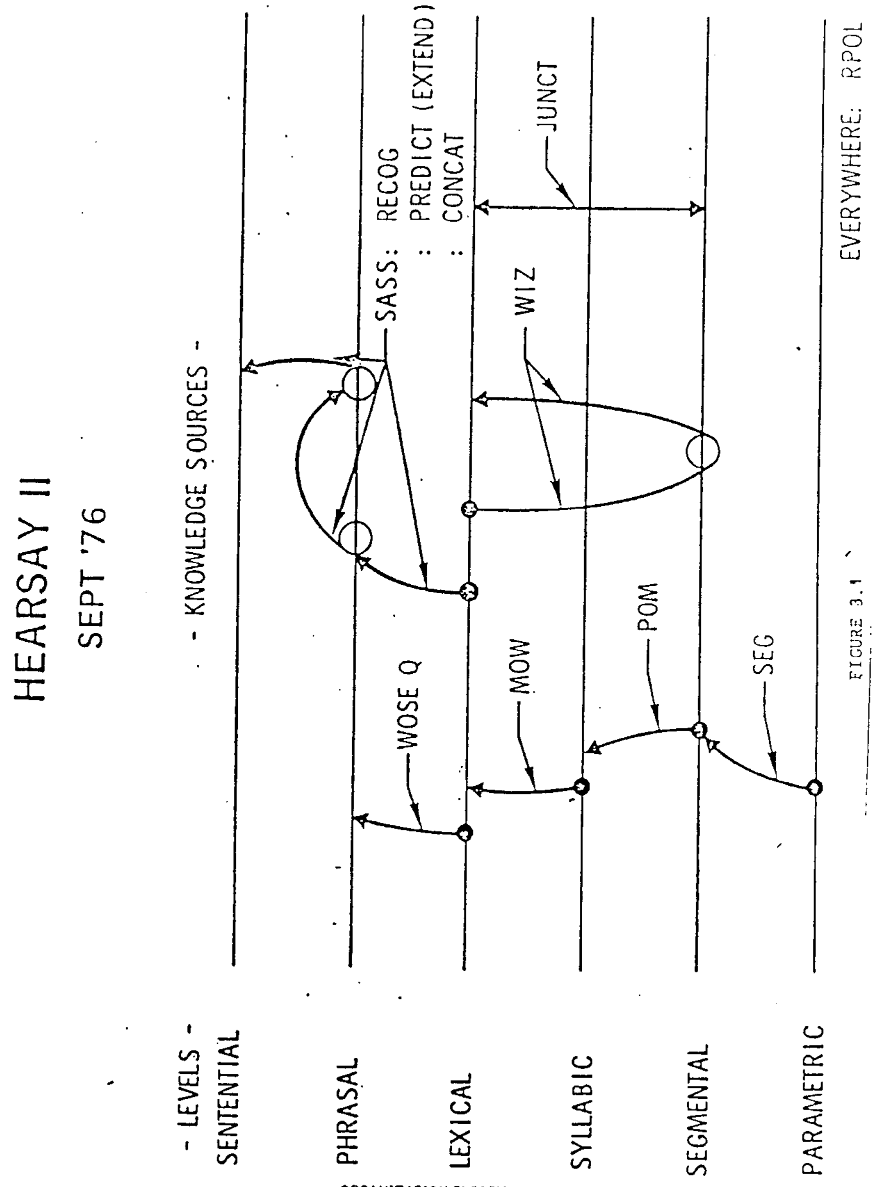


FIGURE 3.1

3.4.2 Task Complexity

Task complexity is concerned with the volume of actions (disjoint or coupled) necessary to accomplish a task. When volume exceeds a manager's ability to grasp the task's "gestalt" then this complexity must be reduced. The solution to this problem is the *Division of Labor* (Smith, 1776). This requires the partitioning of resources (Men and materials; Modules and computer resources) into *units*. Each unit is assigned a specific task related to the organizational goal. The manager delegates jobs to these units, viewing them as *primitives*[7] in the organizational plan. Each unit then interprets the control instructions and expands upon them to control the primitives <u>within</u> the unit. If the manager requires a wing to be built, it is directed to the wing unit. It is up to the manager of that unit to further specify the instructions to his personnel (e.g., machinist, researcher, etc.).

The encapsulation of both mechanism and information is primary to the proper structuring of an organization. It is necessitated by bounded rationality. This melding or mechanism of information has many labels: units, departments, working groups, task forces, corporations, etc. The following describes the characteristics of units that satisfy the constraints imposed by bounded rationality.

1. View the numerous actions (programs) contained in a unit as a single action. (abstraction)
2. Control units as if they were primitive actions. (planning in abstraction spaces)
3. Delegate authority. Commands to a unit are elaborated by and within the unit.
4. Reduce information flow. Information within a unit can be summarized by the unit.
5. Hide detail. Information and control not needed by other units is hidden within the unit.

If a unit is to work in concert with other units, certain inter-unit constraints must be met:

1. The products of the unit must be well defined.
2. The interaction between units must be minimal (near decomposability).
3. The effects of a unit upon other units must be understood.
4. Clear lines of authority must be recognized.

[7] Viewing units as primitives is another example of an abstraction. In this case it is program abstraction as opposed to information abstraction.

5. Clear lines of information flow must be recognized.

The first inter-unit constraint is a minimum requirement. We must know what a unit produces before it can be used. The second reduces the control complexity of the organization which may reduce its effectiveness. The third is necessary so that one unit's action will not undermine another unit's. The last two points insure proper control and the information upon which to base that control.

3.4.3 Coordination Complexity

Once a task has been decomposed to a point where it is comprehensible, coordination must be considered. If the units cooperate in the completion of a task then it is usually the case that there is resource dependence between them, i.e., information, partial products etc. The actions of each unit must be coordinated so that each produces the proper resource a the proper time.

At present there are few heuristics that guide the division process so that coordination complexity is reduced. One of these is the definition of near decomposability of a system (Simon, 1962) which implicitly appears in the contingency theory approach to design: construct the units so that the interaction between units is minimal. Hence reducing the coordination problem. Chanon (1973), Curtois (1977), and McClure (1978) have also investigated the decomposition of systems.

One view held in computer science concerning resource usage is that task size (space and time) can be overcome by adding more processors and memory. Such simplistic views ignore the problem of dependence in task decomposition. Empirical and analytical results on multiprocessor systems such as C.mmp (Fuller & Harbison, 1978) and CM* (Swan et al, 1978) have shown that linear speed ups are not always attainable (Oelnick, 1978; Raskin, 1978). In some problem situations there is an upper bound on the effect of added resources. Conversely, linear speed ups can be obtained if algorithms and information storage are carefully analysed and properly structured. Consequently, greater attention to task decomposition in program organizations and problem-solving must be paid. There follows, a set of approaches to structuring organizations to reduce the complexity of coordination.

Slack Resources

One aspect of coordination complexity is the coordination of "coupled" tasks. Tasks are coupled when the input of one depends on the output of another. Tasks are *tightly* coupled when state changes in one task immediately affect the state of another task. To reduce the tightness of the coupling, *slack* resources are introduced. Buffer inventories are inserted between coupled tasks so that if one task has something go wrong with it, the other tasks are not immediately affected.

Slack has been used extensively in computer systems. Initial versions of systems are quite inefficient (resource consuming), over-consuming time and space[8]. As more experience is gained with such systems, *optimization* occurs. Speech systems such as Hearsay-II and SPEECHLIS (Woods, 1976) have greatly decreased their resource consumption over their years of development. Optimization, as interpreted here can be defined as the recognition of the certain characteristic of the problem and organizing them accordingly. The initial version of a product often requires large quantities of resources to make; replication is cheaper.

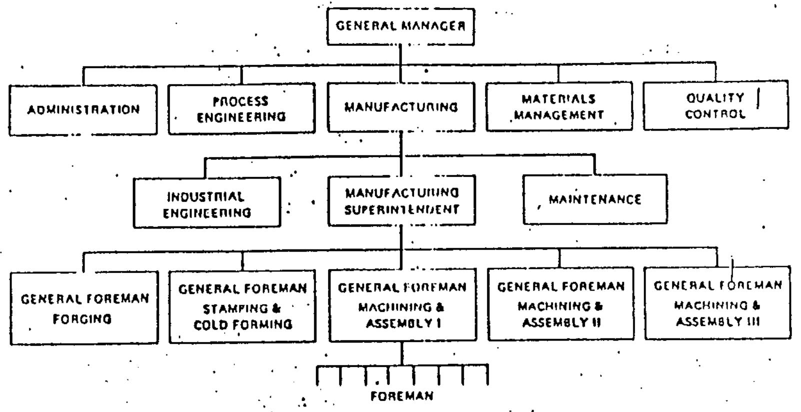
Slack has also been used extensively in computer hardware. The reliability of hardware has been extended by the duplication of functional units. Space vehicles, for example, duplicate essential units.

Two interpretations of slack in distributed systems are 1) the replication of tasks (processes, modules) on alternate processors in case of a processor failure, and 2) the replacement of procedure calls by message queues. Requests and messages to a task are placed in a queue to reduce the synchronizaton (tight-coupling) of tasks.

Function vs Product Division

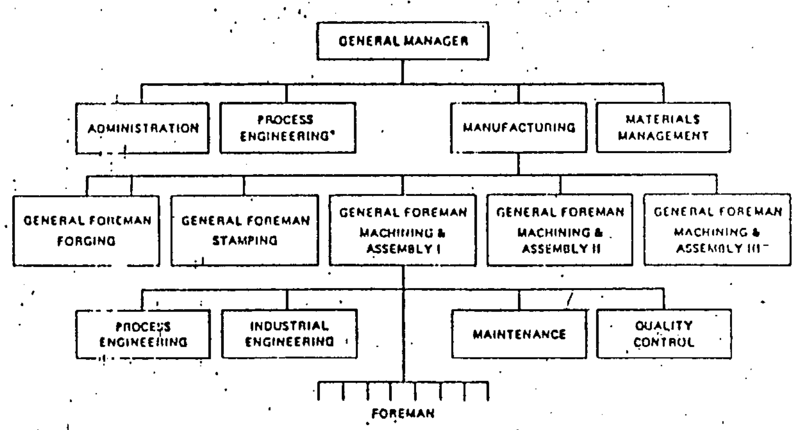
The coupling of tasks can also be reduced by proper decomposition. Organization theory distinguishes between two types of organization partitioning. The first is a *Product or Self-Contained Division*. This division requires that units be centered around the product that is to be produced by the organization. Figure 3.2 is such a division. The second type is a *Functional Division*. A functional division orients the units to the functions necessary to produce the products. Figure 3.3 is a functional division of the plane producing organization. Why do we have these alternate forms of division? Depending on characteristics of the problem being solved by the organization (e.g., producing a plane), one division reduces complexity while the other increases complexity. An important measure of complexity is the amount of *coordination*. Any division of a problem assumes that there is greater interaction within a unit than between units (*interaction locality*). A system that exhibits interaction locality is called a *Nearly Decomposable System* (Simon, 1962). When interaction locality no longer exists, the coordination of units becomes too complex.

[8]Time and Space compose the price system of computer programs.


GENERAL MANAGER
ADMINISTRATION
PROCESS ENGINEERING
MANUFACTURING
MATERIALS MANAGEMENT
QUALITY CONTROL
INDUSTRIAL ENGINEERING
MANUFACTURING SUPERINTENDENT
MAINTENANCE
GENERAL FOREMAN FORGING
GENERAL FOREMAN STAMPING & COLD FORMING
GENERAL FOREMAN MACHINING & ASSEMBLY I
GENERAL FOREMAN MACHINING & ASSEMBLY II
GENERAL FOREMAN MACHINING & ASSEMBLY III
FOREMAN

Organization chart for the manufacturing company.

FIGURE 3.2



*Responsible for developing new processes for new products.

Self-contained structure.

FIGURE 3.3

How can we apply the division methodology to programs? If we were to build an operating system, the accepted method would be to build a separate module for a scheduler, memory manager, I/O controller, file manager, etc. (Fig. 3.4). This is a functional structuring (decomposition) of the program organization and is a general organization that can handle most computing needs. Each job that enters the system interacts with all these modules and indirectly with the other jobs. Thus, coordination between jobs is important and time consuming. Now if one were to use the machine for Basic and APL programming only and wanted excellent system response, this generality would no longer be necessary. The operating system could be divided into two modules, an APL module and a Basic module. Each of these modules would contain their own memory manager, file manager, etc., plus certain physical resources such as disk, a portion of main memory, etc. (Fig. 3.5). Coordination would concern resources shared by both modules only. Thus, the operating system would be specialized towards handling APL and Basic programs. The system's generality would be reduced and so would be the resources spent in job coordination.

Any attempts at applying functional or product decomposition techniques should bear in mind the inter and intra-unit constraints mentioned previously.

Cost Analysis and Contracting

Another method for deciding how to partition the organization is by analysing costs. In an organization there may exist functions that are too costly to carry out. This cost may be due to

- lack of experience within the organization.
- small usage, hence economy of size is not afforded.
- coordination problems.
- Information processing problems.

It is simpler for the organization to contract for this service in the market place. Hence information is reduced to a single price, control to contractual terms. There exist conditions under which contracting is not achievable, this is usually due to the *idiosyncratic* nature of the job. Williamson views most positions in organizations as being idiosyncratic. Though a position may be characterized by a general job classification, the organization, methods of communication, people interacting with, on the job learning, etc. make positions idiosyncratic. A primary consequence is the cost of replacing a person is not negligible nor is the service easily contracted for in the market place.

FUNCTIONAL DIVISION OF AN OPERATING SYSTEM

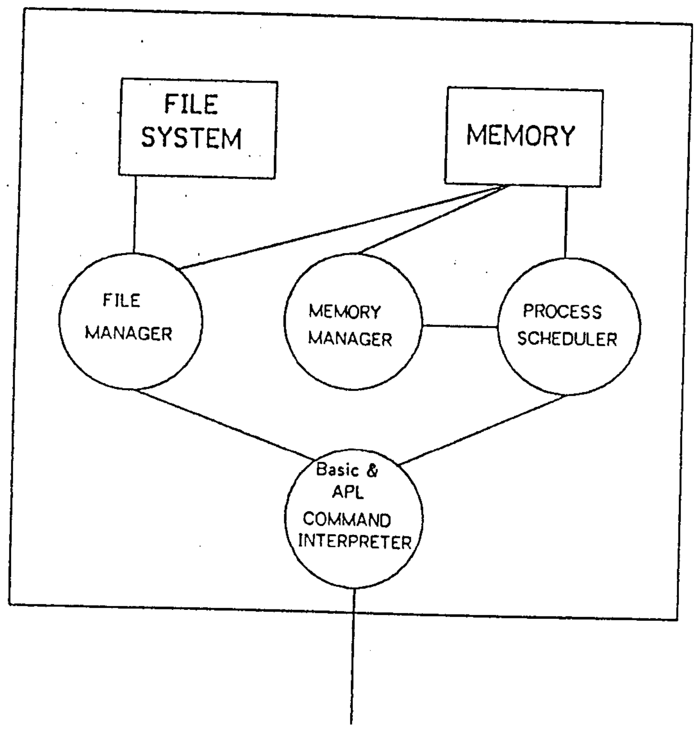
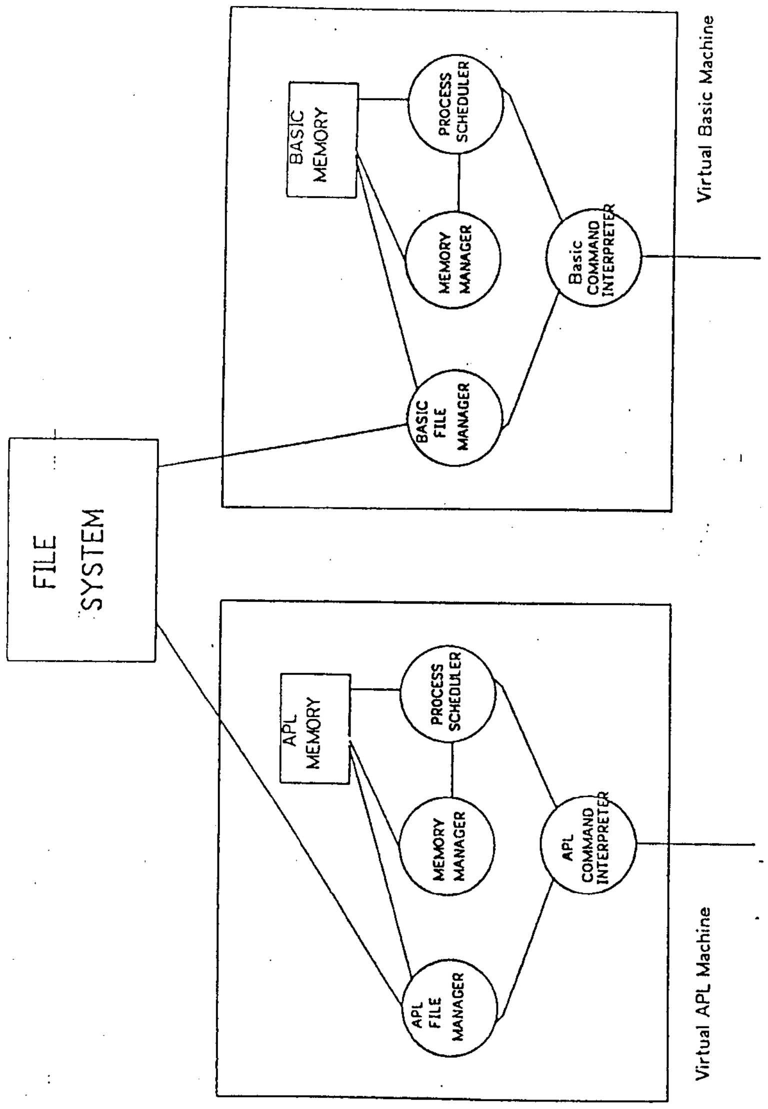


FIGURE 3.4



PRODUCT DIVISION OF AN OPERATING SYSTEM

FIGURE 3.5

One application of this technique in a distributed system is the distribution of specialized tasks. A module may need to multiply large matrices. If the module is running on a standard processor, it may be cheaper (and faster) to contract the task to an array processor.

3.5 Uncertainty

Galbraith (1973) in his explication of the Contingency Theory[9] approach to organization, states that organization structure is dependent upon uncertainty and diversity of task. *Uncertainty* is defined as the difference between information available and the information necessary to make the best decision. Variation in organizational structure results from diverse attempts in reducing uncertainty.

Contingency Theory design strategies are based on the manifestation of uncertain information. We define two types of uncertainty: information and algorithm. First, uncertainty can manifest itself in *Information*. This means that the correctness of the information can be represented by a probability measure; the organization may not fully believe the information (stimuli) it perceives. For example, the correctness of a survey of consumer desires concerning a product the company produces is always under scrutiny. The second manifestation of uncertainty is found in the *Algorithm*. No matter how certain the information a decision is based on, the decision itself may be uncertain due to knowledge lacking in the possible outcomes of the decision. Optimal decisions based on analysis of possible outcomes has been extensively studied (see Marschak & Radnar, 1972).

There seems to be an anomaly in the contingency theory definition of uncertainty and its instantiation in organizations. The definition of uncertainty deals with uncertainty in information, but this is not the case in contingency theory interpretation. At each state in the organization, all information is assumed certain (e.g., how much material is around, how many machines broken, etc.). It is the dynamic characteristic of the environment that is being attended to by the uncertainty reduction methods. We call this *environmental uncertainty*. The environment changes over time (from state to state), and the organization must adapt to these changes in state.

A fourth type of uncertainty commonly found in organizations is *behavioral uncertainty*. An employee, unit (department) or another organization cannot always be depended on to produce the contracted for products or services.

[9]Contingency theory has two premises: 1) There is no best way to organize. 2) All ways of organizing are not equally effective.

The following sections explore the problems of uncertainty in information, algorithm, environment and behavior.

3.5.1 Information Uncertainty

This section is concerned with the problem of uncertainty in information. Organization and economic literature currently consider only one type of information uncertainty: its errorfulness. The primary focus in decision theory has been to optimize the decision given uncertain information.

March and Simon recognized that error is introduced through incorrect classification during the abstraction process. The execution of the correct program of action in a business organization requires the ability to identify and classify the stimuli entering the system. The identification and classification process is dependent upon the cognitive limits (i.e., rationality) of individuals in the organization. A problem lies in the subjectivity of this rationality. Again, dependent upon the information, goals, and views held by the individual, the identification and classification process may produce varying results. Hence the collecting and summarizing of information (stimuli) reflects the frame of reference of the individual. The evidence is replaced by subjective conclusions. March and Simon call this phenomenon *Uncertainty Absorption*. Depending on the frame of reference of the individual summarizing the evidence, uncertainty absorption can be beneficial or harmful to the organization's performance.

If the organization's identification and classification abilities are good, the proper information and decision to execute can be communicated effectively and efficiently with little feedback. But the introduction of uncertainty into the organization requires feedback, hence an increased amount of communication. Also situations that are less programmed require more coordinating communication from higher levels in the organization.

Galbraith (1973) describes another approach to reducing information uncertainty. Simply, it is the aggregation of information from multiple sources in the organization. Due to the possibly large amount, computer-based summarization techniques must be used. Hence, a variety of information in condensed form can be delivered to a manager.

Experience with software systems has shown that information may produce more types of uncertainty than typified by its error. In the following, a closer look is taken of information in computer systems, its variety of uncertainties and methods for reducing these uncertainties.

Information is the data that is passed between and examined by modules in a program

organization. This encompasses not only information that is used to initiate action but also information which lies dormant until some module decides to examine it.

Three types of uncertainty in information are distinguished:

1. INTENTION: The reason for the creation and transfer of the information: who created it, where is it going, what will it cause, etc.
2. VERACITY: The degree of truth or belief in the information being handled. For example, if the information is a segmentation of speech, then the label assigned to the segment may have a belief rating which depends on how well it matches a characteristic template.
3. SEMANTIC: The semantic interpretation of the information. Is the module's interpretation of the information correct? Is there more than one interpretation?

Linguists distinguish other types of information attributes such as emotive vs. cognitive; performative vs. descriptive; signs vs. symbols; analytic vs. synthetic; etc. (Lyons, 1968). None of these seem useful with respect to uncertainty in structuring organizations.

3.5.1.1 Uncertainty of Intention

Apart from the actual contents, there exist attributes that are peculiar to but not contained in the communicated information. These attributes can play vital roles in the system's processing hence organization, depending on the problem uncertainty. The following is a list of information "intention" uncertainty types found in large systems.

1. CONSUMER UNCERTAINTY: Who will receive this information?
2. PRODUCER UNCERTAINTY: Who is the source of this information?
3. FUNCTIONAL UNCERTAINTY: What mechanism will use this information?
4. RESULT UNCERTAINTY: What is the result of the use of this information?

The following sections examine each, and propose methods for reducing uncertainty.

Consumer Uncertainty

Large business organizations and multi-module programs frequently produce information in one part of the organizational structure to be used in another. Frequently the producer does not know who the consumer is[10]. That is, the producer is uncertain of what module(s) will

[10]This is one of the assumptions of the Hearsay-II and Pup-6 systems.

consume the information. Consumer uncertainty is a asymmetric relation induced by the information producer. Hence the distributed information can be viewed as a resource without a receiver (consumer) label attached. There may be one specific consumer or many. The consumers may know what pieces of information to consume, or "taste" all before deciding. Consumer identification requires the produced information be made available (*information availability*) to prospective consuming modules, and they have the ability to identify the information they need (*information identification*).

Information Availability

Uncertainty in possible consumers of information requires a system organization that allows the transfer of information to relevant modules.

There exist three traditional methods of communicating such information.

1. BROADCASTING: where every module is immediately notified of the existence of the information. E.g., PUP6.
2. MESSAGE BOARDS: where information is placed on the message board for perusal by other modules. E.g., Hearsay-II.
3. WORD OF MOUTH: a module informs the modules it knows about and they pass it on to the modules they know about, etc.. E.g., Actors.

Each of these communication methods can be graphically described.

- BROADCASTING: Broadcasting requires a complete graph where each node is a module and each arc a communication channel.
- MESSAGE BOARD: This requires a central data structure to which all modules have a communication channel. This is a star configuration.
- WORD OF MOUTH: This is the least structured of organizations. All that is required is that the organizational graph be connected.

Of course, there can be combinations of the above in a large system.

Information Identification

Information identification requires a prospective consumer module to recognize the types of information it consumes. It also requires the ability to read and understand the communication. The next proposition is one that is usually taken for granted in organization theory but is important in computer systems.

Modules that are to communicate must share a common language of communication.

In an ideal situation, the information being communicated by the producer of the communication will be completely understood, both in intent and control, by the consumer of the communication. In human organizations, such is not the case. Uncertainty always enters into the production, transfer, and analysis of information. In program module organizations there is a chance of reducing uncertainty through the definition and sharing of a language of communication. The following are requirements that a shared language of communication should satisfy:

1. A clean communication channel.
2. Isomorphic procedures for the assembling and dis-assembling of communicated information.
3. The semantics of interpretation of the communication be equivalent at both ends of the channel.

Once the communication is understood, it is up to the module to decide to act on it. In Hearsay-II, this was accomplished by a knowledge source *Precondition*.

Producer Uncertainty

Until recently, uncertainty in who produced information received little attention. Why is there interest in the producer of information? One reason is "accountability." Most systems "chug" along until the processing goal is reached. The path used to reach the goal is never examined (except in debugging systems at a programming language level). Our interest in module accountability stems from the uncertainty in algorithm and information. Uncertainty in algorithm requires the explication of the modules used to arrive at this point in the solution space. Such ideas have appeared in the blackboard graph structure of Hearsay-II. It describes the decisions that led to the creation of a hypothesis. Also MYCIN (Shortliffe, 1976) keeps the tree of productions that resulted in a diagnosis, for knowledge debugging and explanation (Davis, 1976).

Systems make decisions based on information produced by a chain of processing. Most systems do not look at the processing that produced this information. We call this *Producer-History-Free* decisions. Decisions based on both the information and its processing history we call *Producer-History-Sensitive*. A retrospective analysis of the Hearsay-II system (Lesser and Erman, 1977) has shown that the focus of attention module suffered because it did not use the hypothesis processing history. Instead, resource allocation decisions were producer-history-free. Lesser and Erman state that it is necessary to know what processing produced an hypothesis in order to schedule further processing. We believe that it is the uncertainty in the algorithm that requires decisions to be producer-history-sensitive. If the algorithm is strong, there is no need for deciding what processing should be done next.

The following proposition seem to be relevant.

> The effects of producer uncertainty increases as algorithm strength (certainty) decreases.

A strong algorithm guarantees that the best decision will be made at all times. Only when confidence is lacking in processing decisions is there interest in how those decisions were made.

The above ideas have motivated our interest in producer uncertainty. The following describes how it affects an organization structure.

As pointed out previously, producer uncertainty is important when a strong algorithm is lacking. Its importance lies in the system's ability to track down the root of its problems when unable to reach the processing goal. This is analogous to innovation in a business organization when the organization's programs no longer satisfy its goals. Hence, the organization must identify the inadequate programs and modify or replace them. This is also a form of debugging (Sussman, 1973). Similarly, faced with a weak algorithm and various types of producer uncertainty, an organization structure must track down the root of its problems, i.e., the modules that are making uncertain decisions.

An example will serve to clarify this. The Hearsay-II architecture is a organization structure. The algorithm is a combination of the algorithms in the various knowledge sources. During execution, the best hypothesis at the phrasal level may be incorrect. To discover why the hypothesis is incorrect, the system must understand the processing history that led to the construction of that phrasal hypothesis. By tracing back over the links and hypotheses that support the hypothesis, it may be found that at the word level, at some area of the line, the supporting word hypotheses are incorrect.

Three different actions can be taken at this point:

- An examination of the Word Hypothesizer knowledge source (POMOW) to see why it is unable to produce the correct word.

- Examine the modules that produce the data used by POMOW (e.g., segments).

Both of these are currently done by a human expert. The third is a system solution to produce the word at run time.

- Reduce the threshold used to hypothesize words in that time area.

Given full knowledge of producer history, selective analysis can be made. But if the history of producers is uncertain, a different set of capabilities is necessary.

Module Tracking

To track down the errorful modules, a system needs to separate sub-organizations and test them in a controlled environment. In other words, the system must experiment upon itself. This requires the existence within the organization of an ***experimentation module***:

> To reduce producer uncertainty, an experimentation module requires:

1. Control of subsets of modules within the organization.

2. Access and control of the communication lines entering and contained in the subset.

3. A model of computation and intercommunication of the modules in the subset being analysed.

4. Test data.

The first part allows the experimentation module to execute another module and the halting of computation to examine intermediate data states and control. The second allows the creation and monitoring of the data environment that the subset is to execute within. The third is most important, since it provides the information that the experimentation module uses in deciding how to test the subset. Without this information, the experimentation module must blindly search for data and decision paths to test. The fourth provides the data for error testing. We will assume this is provided by the user (although one research path deals with the internal production of the test data).

For example, assume that links on the Hearsay-II blackboard do not exist; i.e., a hypothesis does not point back to the hypotheses from which it was synthesized. Then there is complete producer uncertainty. If there were an experimentation module in Hearsay-II, it would have to exist at a level of control comparable to the user when in debug mode. The experimenter must first freeze the processing of the organization, then choose a subset of the modules (knowledge sources) as candidates for experimentation (to find the source of error).

Subset selection is another area of study. How do you decide what area within the program organization to experiment with? One method is the use of an organization model (3) by the experimenter. The model would define for each module, or groups of modules, what the import of their processing is towards the system goal. The level of detail of the model can vary greatly from general input/output characteristics to detailed descriptions of the internal workings of each module. By associating incorrect information with modules in the model (by structural, relational, attribute, etc. features), a subset of the organization is delimited for experimentation. The Focus of Attention (FOCUS) knowledge source of Hearsay-II used such a model. Each KS defined what is called a "stimulus and response frame", i.e., the I/O characteristics of the KS. FOCUS used these frames to decide which KS was best to prosecute the goal of the organization.

Lets say the Syntax and Semantics module (SASS (Hayes-Roth, Mostow & Fox, 1978)) was chosen. Two approaches could be taken:

BLACKBOX APPROACH: By controlled modifications to SASS input (2), the experimenter can observe changes in the output (1). Knowledge of the existence and use of the inputs and outputs would be provided by the experimenter's model mentioned above. For example, the model could specify that there were two inputs, both either words or phrases. By means of the model or a string matching algorithm, the experimenter may discover that the output is a (filtered) concatenation of the input. In addition, if the correct input (4) is supplied, the correct output should result. Or the model can describe the expected behavior of the module. If the module does not fulfill expectations, then the SASS module is in error; otherwise the producers of input to SASS must have problems. The only intelligence required in the experimenter is the ability to recognize that the modules act as filters on the data and on the relationship between input and output (3).

GLASSBOX APPROACH: First SASS is isolated (1) (2). The experimenter has access to the model of computation used by the SASS module. It can follow the reasoning (3), i.e., sequentially trace (interpret) the flow of data within the module. By analysis alone, the experimenter should be able to decide whether the SASS module erred. Obviously this approach is unatainable today because it requires the experimenter to *understand* the algorithm embodied by the algorithm in the module. The experimenter must understand each decision node in the computation model, how they combine into decision subgraphs, and how they compare to correct decisions (4).

The blackbox approach attempts to isolate errors in specific modules while the glassbox approach attempts to mimic an expert in the debugging of a module's algorithm.

Evidence of the feasability of the blackbox approach can be found in the Fault-Tolerance Domain. In the C.mmp multiprocessor (Fuller & Harbison, 1978), processor error is analysed using what is called the "suspect/monitor" approach to error diagnosis (Sieworick et. al, 1978). This approach is a simplification of the blackbox model.

Functional Uncertainty

The communication of a packet of information by one module to another entails, hopefully,

a bounded amount of processing. For example, the command to sort an array requires processing power defined in terms of time and space used in understanding the packet and the subsequent execution of the sort algorithm. In this example, the required processing is known to the module that produced the sort communication packet (i.e., the functional effects are known). In Hearsay-II the production of an hypothesis may stimulate other KSs to verify or modify hypotheses or create new hypotheses. The producer of the initial hypothesis does not know how its hypothesization action will affect subsequent processing. This is a goal of the Hearsay-II architecture; knowledge sources must know as little as possible about other KSs. Our interest in functional uncertainty stems from effects it may have on the progress towards the system processing goal.

As mentioned above, the sort function requires both time and space. They are resources that a *resource limited* system must use miserly. Given more than one way of achieving a goal, the methods may differ in the resources they use; the unwise use of resources may prolong or prevent the attainment of the processing goal (e.g., deadlock in operating systems). If the function entailed by an information packet is uncertain, the resource demands will be unknown or best described by a probability distribution. In addition, the correctness of the function may be in doubt. What remains to be understood is how this uncertainty affects the organizational structure.

Resource Management

If the function probability distribution is known, resource management can be based on expected function. If all functions are equiprobable or the function is unknown, methods must be devised to monitor the resource consumption of the modules in the system. Monitoring requires the accessing of communication channels between modules and local and global data structures. Hence a resource monitoring module must be created which has access to the above information carrying structures. It must also have the authority to halt and re-direct the processing of the system. An alternative structure is to create a resource depot. All requests for resources are made to and granted by the depot. Simply, more attention must be paid to the allocation and monitoring of resource usage in program organizations.

Function Correctness

A second view of functional uncertainty is the correctness of the executed function (algorithm). The communication of information results in the creation of an expectation by the controlling module. Whether the function satisfies the expectation can only be discerned by accessing the function's results. Either the function returns results to the controlling module, or the controlling module must monitor the system. The problem of monitoring uncertain functions (algorithms) is discussed in section 3.5.2.5.

Result Uncertainty

One method of discovering the degree of control a module has over another is to compare the processing results of a module, M2, with the information communicated to it by module M1. For example, a father says to his son that the room is cold. The result is the son's changing of the thermostat's position and the house's mean temperature. The father has exercised direct and certain control over his son. The same father goes to a restaurant for dinner and finds the restaurant chilly. He tells the waiter that the room is cold. The father expects the waiter to turn up the heat, instead the waiter asks him if he wants another drink. Several minutes later, the manager of the restaurant sits down at a table and voices the same complaint; this time the waiter turns up the heat.

This example illustrates the uncertainty of communication. This uncertainty depends on the authority relationship between the man and the waiter. If they have different perceptions of what the relationship is, the result may be uncertain. Not only does result uncertainty depend on authority relationships, but it depends on uncertainty as to who is the consumer. If you are unsure of who is to receive the communicated information, then the result may be equally uncertain. A third source of uncertainty stems from the lack of a prescribed response. Depending on the state of the information consumer, the result may vary.

This uncertainty in result poses a number of problems for the information producer if it wants to know the result of its communication.

1. A producer of information must have access to the result when it is produced (important when consumer is unknown).
2. A producer must be able to analyse and understand the result.
3. A producer must be able take alternate action if the result does not satisfy expectation.

The effects of these problems on the organization structure are the following:

1. The producer must have access to many communication paths and data modules.
2. The producer must share a common communication language with the consumer (who produces the result) and have a model that allows it to interpret the result.
3. To take alternate action, the producer must have sufficient control (authority) to redirect the system's attention (or part of it).

In addition to the structural aspects, a clear description of authority relations must exist within the organization, enabling modules to act appropriately. Secondly, in an authority

relationship (e.g., employment) the scope of the control lexicon and language must be defined along with the program of actions associated with each control communication.

3.5.1.2 Veracity Uncertainty

Systems that attempt to understand speech must do so using highly uncertain data. That is, the labels (phones) assigned to portions of the utterance are errorful, reflecting the lack of knowledge in the segmentation and labelling techniques[11]. Vision research must also deal with the problem of interpreting errorfully labelled scenes. An alternate way of viewing this error is to say the datum (label) is correct with some measure of certainty (probability). The smaller the measure the less certain the datum's correctness. This problem pervades most decision processes. A strategic analyst, looking at information coming from a country must assign his/her own belief as to how true each datum is. We call this degree of belief in information, *Information Veracity*.

Given that the veracity is uncertain, how does a system continue processing in spite of it? There are three approaches: the system can use the information as is, it can ignore the information, or it can reduce the veracity uncertainty. The following are three approaches to the reduction of veracity uncertainty.

Reduction by Verification

In many actions thre is an expectation as to what will happen next. We expect the car to start when we turn on the ignition, we expect it to move when we press the gas. In speech understanding, if the previous sentence mentioned the desire to go shopping, it is reasonable to expect food to be mentioned in the next sentence. This expectation is used to *verify* hypotheses on the blackboard at different levels of representation.

The important characteristic of uncertainty reduction by verification is the existence of previously acquired data at some level of abstraction (representation) that can confirm or deny the acceptability of a new datum.

Reduction by Synthesis

Image understanding systems construct scene descriptions based on picture elements (pixels) composed of multiple labels. In fact, most labelling schemes assign a set of labels with corresponding probabilities to each pixel. Hence the labels at each pixel have high veracity uncertainty. To reduce the uncertainty, and the number of labels at each pixel, the

[11]See Goldberg (1975) for an in depth discussion of such techniques.

labels of a pixel are compared to its neighbouring pixels to see if they are mutually supportive or mutually inconsistent. Then a consistent subset of labels from the pixels is chosen to describe the scene. Models of scenes are used to describe how information (labels) can be combined. This is called *synthesis*. One approach to synthesis can be found in the relaxation technique (Zucker, 1976; 1977). A Script (Schank, 1976) is another synthesis method used in understanding natural language. The Conceptual Hierarchy (Fox & Mostow, 1977) is yet another domain model in which uncertain information is synthesized.

The key idea is that uncertain data are combined to lend mutual support, thus increasing their collective certainty.

Reduction by Extrapolation

This is a technique[12] that is used in many branches of science and social science. A psychological model's adequacy (certainty) is teted by how well it predicts behavior. In statistics, mean and varizance estimators are used to predict what future samples will look like. Data are compiled, a model constructed or selected, and new data predicted. This method of data compilation and prediction is called *extrapolation*.

In the above examples, the compiled data are certain, the model is not; i.e., the model is not known to be normal or exponential. When the prediction is not supported by future observations, the model is rejected. In our problem, reducing information veracity, it is the model that is certain (hopefully) and the data that are not. The extrapolation process can be modified by decreasing or increasing the data veracity proportional to its predictive power.

3.5.1.3 Semantic Uncertainty

Semantic Uncertainty is the result of incorrect or ambiguous interpretations of information. Two sources of semantic uncertainty are:

1. Lack of agreed upon language for representing information shared among modules.

2. Differences among world models (belief systems) employed by modules during information interpretation.

To deal with the first problem, the following additions to the organizational structure are proposed: To guarantee that modules understand one another, a lexicon and language are defined for each communication channel between modules. Only legal strings may traverse the channel. Secondly, all information storage modules (data stores) have a lexicon and language defined for each. Only legal strings may be stored in the data store. Any modules that have access to channels or data stores must satisfy the requirement that they "understand" the languages associated with them.

[12] Extrapolation can be used to produce information for reducing uncertainty by verification.

The second source is more difficult. In the extreme, a program of action can be defined for every legal string received by modules, hence ignoring the interpretation problem. Accepting that modules develop diverging world models, mechanisms should be introduced to reduce uncertainty. Obviously, the mechanisms described under veracity uncertainty (Section 3.5.1.2) are applicable here. Other mechanisms of a more structural flavor are also applicable. For instance, modules can be organized into an authority hierarchy. If a module detects an ambiguity in interpretation, a message may be sent up the hierarchy asking for clarification. The message would be handled by the module best able to decide. A second mechanism is to tag the information with the producer's ID. The consuming module may attach a communication channel to the producer and query the producer as to the correct interpretation (assuming they share a common language).

3.5.2 Program (Algorithm) Uncertainty

In the section on bounded rationality the information processing model of an organization was introduced. One of its prominent features was the prescription of programs of action for classes of stimuli. In many cases it is almost guaranteed that the program will produce the desired result, for example, the construction of a car (program) given an order (stimulus). But there are many classes of stimuli in which programs produce varying results, no results, or for which no programs exist at all. An example of the former is a method for investing in the stock market, an example of the latter is putting a person on the moon. Programs of this nature are described as being uncertain. Their execution does not entail a single expected outcome. Emphasis has been placed on the ability of an organization to pre-program actions in the presence of various stimuli. This allows consistency of action across all units. It is in the presence of environmental change that the organization's programs become inoperable.

In a changing environment, many dilemma face the decision maker.

1. Performance vs Learning: Is adaptation sacrificed in support of the blind performance of a job?

2. Open vs Closed System: Is the environment monitored for changes or ignored?

3. Performance vs Reflection: Is the job blindly executed or periodically reflected upon?

4. Challenge vs Threat: Are new situations viewed as a desired challenge or an

unwanted threat?

Depending on how the decision maker perceives these dilemma, the adaptation of the organization may be impaired.

Two approaches for dealing with program uncertainty can be discerned from the management literature: decision theory and adaptation.

3.5.2.1 Decision Theory

Classical decision theory is concerned with the maximization of utility; where utility is some measure of the "goodness" of an event. The event of concern is the decision to be made. Given a state of the world $\chi \epsilon X$, where X is the set of possible world states, the decision function α is a function of the signal y, an abstraction of χ.

$a = \alpha(y) \quad y \in Y$ (Y is the set of abstractions on X)

The result of the decision is an action a. The utility (payoff) of a decision is defined by

$\omega(\chi,a)$

which is futher defined as

$\omega(\chi,\alpha(y))$

ω can be described as the payoff of decision α.

The set Y of signals defines a set of abstractions on X. A particular partition of X is called an information structure and is denoted as η. The combination of an information structure and a decision function, (η,α), is called the organizational form.

Incorporating the information structure into the analysis, the utility function for a particular decision function and information structure can be redefined as

$$EU = \sum_{\chi} \omega(\chi,\alpha(\eta(\chi)))\ \phi(\chi)$$

where $\phi(\chi)$ is the probability of world state χ. In single-person decision theory, analysis centers around finding the optimal decision function that maximizes expected utility given a particular utility function ω.

Simply, decision theory is concerned with the achievement of the "best" decision given a utility function that orders possible outcomes.

Work in making decisions with uncertain data has also been done. If:

1. Information can be described numerically,
2. its error distribution can be computed,
3. the interaction of information in determining a decision be ascertained, and
4. the utility of outcomes be defined.

Then the analysis of the sensitivity of decisions to information can be analysed and the uncertain decisions determined. Adjustments can then be made to the decision functions accordingly.

Applying these theories to human or computer organizations is frought with problems.

First, the functions used in decision models are all real valued (e.g., α, η). Except for some obvious categories of applications, a program's computation is symbolic. As a result, a program model interpretation of the decision function α and the information structure η is not obvious.

Second, the analysis of organizational situations requires the depiction of all variables. Few domains are well defined; precluding denotation of relevant variables.

Finally, in addition to variables, probabilities of each joint and conditional state between observed variables must be known in order to make decisions of expectation. Such measures are incomplete, if available at all, in most domains.

The decision theory approach can be classified as making decision with incomplete information. Game theory (Rapoport, 1966) is concerned with making decisions using complete information but involving 2 or more *competing* players (organizations). Procedures such as Min-Max have been proposed for maximizing outcomes assuming rational players.

3.5.2.2 Adaptation

In the decision theory model, the decision function α was itself a "variable" to be optimized. Under the restriction of a quadratic utility function, it was found that the optimal decision function is linear (Marschak & Radnar, 1972). When input data uncertainty characteristics change, adaptation takes place by finding the best decision function.

Much of the work in adaptability in organization theory has diverged from the decision theory approach mainly because of the "uninterestingness" of the model. Most program

Innovation situations cannot be characterized by quadratic utilities or numeric distributions of data. Before parameters of a decision or program can be quantified it must be discovered what these parameters are. Hence, organizational analysis has taken a more basic and symbolic approach. What information is necessary to design a new program, where can it be found and how can it be monitored? Transaction analysis has proven useful in its attempt to characterize the information content of transactions that take place in a particular task. Secondly, it analyses the types of uncertainty creating conditions that appear in transactions, e.g., information impactedness, small-numbers, first-mover, etc. This analysis may take place within the organization, at its boundary with the environment (Boundary spanning) and the external environment.

The second phase of the adaptation process is *program elaboration*. Program elaboration is concerned with the construction of an ordered set of decision and action procedures to achieve a task. What decision variable (internal or external), and actions are useful or available, and what are the possible orderings are two aspects of the elaboration process. It requires the bringing to bear of a variety of knowledge.

Given a set of variables, actions and possible ordering relations, organizational analysis has focussed on the communication and authority structure that support program design activity. In particular, group-problem solving has been studied as one organization for experts to interact in creating programs. Communication and authority channels have been analysed with respect to their decreasing problem-solving time.

There has been considerable work in the understanding of how humans innovate. This work has led to the production of computer systems that model human problem-solving capabilities. Most of this work has been categorized as Information Processing Psychology and Artificial Intelligence. That literature will not reviewed here.

As described above, the management approach to program innovation is either too formal or too abstract to be of value. In the following, an attempt is made to analyse how program innovation may be achieved in programmed systems.

By definition, all algorithms are well defined. But it is not always true that the execution of an algorithm produces the desired results. The less we understand about a domain, the less certain are the constructed algorithms. Even when coupled with "certain" information, the algorithm may never produce the expected results; system resources may be consumed long before hand.

What does it mean to "partially understand" a task? Simply, it is the lack of complete knowledge of the relationship between actions and their effects. Economists attempt to

understand the cause and effect relations in our economic society. They control an economy by prescribing programs of actions. Empirically their results have been dismal, implying that their understanding is lacking and their algorithms uncertain.

An algorithm is actually a plan of *control*. Control, direct or indirect, should not be exercised without an understanding of its effects. Direct control is exercised through the emission of an action initiation message. A decision to initiate an action is based on the module's view of the informational environment. The expected outcome of the action (module) initiation is deemed more beneficial than other alternatives with respect to the goal of the system.

Direct control requires:

1. the name of module to be initiated or a functional description,
2. initiation rights to the module,
3. a communication path to the module, and
4. an explanation for why the module should be executed.

The first three requirements are obvious from a computer point of view. Omission of one may cause unpredictable behavior. Unless each of the requirements are optimally supplied and/or performed, resource consuming side-effects could adversely affect future computation. For example, the transmission of messages along an inappropriate communication path may cause stagnation in the system by consuming an inordinate amount of resources. The fourth requirement is used in the analysis of system behavior to accommodate environmental changes.

Indirect control implies a modification to the environment which in turn affects a later control decision. There are two types of indirect control:

- Knowledge of effects of environmental change (Manipulated field control (Dahl and Lindblom, 1953)).
- Ignorance of effects of environmental change (Spontaneous field control).

It is the second type that plagues complex programs. Uncertainty in both information and algorithm precludes full knowledge of an action's effects, thus allowing unanticipated system behavior.

We have looked at how businesses react to their environment. Given "certain" information and algorithms, programs of action can be prescribed. A stimulus (e.g., a product request) initiates a corresponding program, producing the desired results.

When uncertainty exists in actions, anomalous results may occur. A more guarded approach to program creation (sequence of actions) and execution must be taken. This approach can be divided into three parts: information gathering, program construction, and system evaluation and repair. The following sections explore these and their effects upon system organization.

3.5.2.3 Information Gathering

Before any decision is made as to the program of action to be initiated, necessary information must be made available for analysis. Information gathering requires the recognition of relevant information, and the ability to access it. Information gathered only at the stimulus source may negatively reflect its *locality*. Local views are not always advantageous; information from many sources must be procured.

Due to bounded rationality, there is a limit to the amount of information that can be absorbed. Abstraction and omission must be diligently used. A side effect of abstraction and omission is uncertainty absorption. Abstraction and omission may introduce error that reflects the abstractor's perception of reality. A second problem in gathering information is its veracity uncertainty. Section 3.5.1.2 describes methods of reducing this uncertainty.

3.5.2.4 Program Construction

Once the information has been gathered, a program of action is prescribed. To prescribe a program, information must be classified. Classification, is itself, a difficult problem. Methods are dependent upon the data to be classified; they range from purely statistical methods (e.g., cluster analysis) to symbolic methods as found in artificial intelligence.

Once the information is classified, a program of action is associated with each class. Decision theory (Kassouf, 1970) is concerned with choosing the best course of action. It distinguishes two parameters in the decision process: the *environment* in which the action is to be executed, and the *action* itself. The two parameters determine distribution of possible *outcomes*. Decision theory chooses an action given an environment (or a distribution of environments) and a utility rating of the outcome distribution. Hence, decision theory maximizes the utility of an action when uncertainty is found in the environment and in the outcomes.

Weide (1978) has described an approach to algorithm design and analysis using statistical techniques. Given a problem whose exact solution is too costly to compute, algorithms can be designed and analysed that approximate the solution within some habitable error bound. This

approach fits precisely into the decision theory model. A statistical algorithm is one whose outcomes are uncertain but can be described by some distribution.

In most applications, the set of outcomes is only partially known and cannot be described by a distribution. Any action chosen may result in harmful or useless results. When an outcome is uncertain, the action must be *insured*. In a market system the primary unit of value is the price. All commodities and services are bought and sold using money; insurance is expressed in dollars. But there exist cases in which pecuniary insurance does not suffice. Actions that cannot be reversed, or losses that cannot be replaced cannot adequately be described by a utility function. Thus they are not assauged by money; other forms of insurance must be sought.

One method of non-monetary insurance can be found in *slack resources*. Excess resources are supplied so that outcomes that result in over-consumption of resources are not detrimental. Computer programs can be viewed solely from the point of resource consumption, of which *time* and *space* are the two primary resources. An algorithm can be classified as consuming *infinite resources* (does not work), *effectively infinite resources* (i.e., more than available, effectively does not work), *sufficient resources* (works), and *optimal resources* (works best). By providing slack resources, some algorithms that consume effectively infinite resources can be re-classified as sufficient. Other methods of controlling poor outcomes are described in System Evaluation and Repair.

Business organizations and computer programs alike exist in environments that are continually changing. Subsequently, the classification of stimulus information may result in classes for which no programs exist. What is required is the elaboration of a new program of action. This requires the ability for *innovation* in the organization (March & Simon, 1958). The field of problem-solving is directly concerned with the elaboration of new programs of action. Relevant work can be found in (Newell & Simon, 1963) (Fikes & Nilsson, 1971) (Sussman, 1975) (Sacerdoti, 1975).

A changing environment may also cause the classification structure to be inappropriate. New structures of concepts have to be developed to classify the changing environment (Fox, 1978).

In summary, the decision as to what program of action to take depends upon the careful gathering and classification of information, followed by a weighing of the possible outcomes of action. Information gathering requires the denotation of relevant data structures within a program organization combined with access channels. The variable choice of actions requires the decision process to have control rights to a number of processes needed to carry out various actions.

3.5.2.5 System Evaluation and Repair

Once a program of action has been chosen, uncertainty requires that the organization be monitored. Incorrect results should be spotted and rectified before they run amuck. *Feedback* is a primary process in system evaluation. The need for feedback in controlling organizations has been recognized in business (March & Simon, 1958), and in artificial intelligence (Sussman, 1975) (Zucker, 1977a) (Soloway & Riseman, 1977).

The implementation of feedback in computer programs requires capabilities similar to those found in information gathering system. Multiple sources of information must be tapped and examined in order to discover how the program is progressing. But in this case the system should have a better idea of where to look; direct control exercised by the program points to possible information sources. However, unknown indirect control (spontaneous field control) may result in unanticipated effects that are difficult to detect. Areas of possible change, as predicted by the initiated program, and areas of expected stability (i.e., areas not thought to be affected) need to be monitored.

Once the information has been gathered, it must be evaluated. That is, the present state of the system must be evaluated to decide whether the current program of action is the best. This involves both the recognition of errors and a comparison of the present program's success with other possible programs.

Once a problem is discovered, the system's attention must be re-directed, i.e., a new program of action initiated. Typical methods of redirection used in artificial intelligence are Backtracking, Generated and Test, Heuristic Search, and Hill Climbing. Hearsay-II stores all hypotheses on the blackboard and re-directs attention by scheduling knowledge source execution on what is evaluated to be the currently best hypotheses (Hayes-Roth and Lesser, 1977).

The ability to re-direct a system's attention requires the system to patch or change the information currently stored in data bases, and to initiate new or old programs of action. Thus, both information and control channels must be made available to the process directing evaluation and repair.

3.5.3 Environment Uncertainty

Organizations, like programs, exist within environments that are continually changing. In many cases, these changes are well understood, predicted and planned for. But it is often the case that environmental changes are not predictable and may cause unanticipated effects. For example, the middle east political situation has resulted in the unavailability of cheap

energy forcing industries to search for ways to reduce energy consumption.

Environmental uncertainty can be viewed as resulting from information impactedness conditions and bounded rationality. The appropriate information is not generally available for managers to anticipate environmental changes. And even if it is available, the complexity of interaction in an environment precludes their analysis due to bounded rationality.

The question remains as to what affect environmental uncertainty has on an organization. Under ideal operating conditions, a stimulus' optimal response is known and used. But uncertain environments generate unanticipated stimuli. These stimuli are passed up the hierarchy because, at their point of manifestation, the employee does not know how a solution will affect other parts of the organization. The stimulus is passed up until it reaches a level where a manager has enough information and authority to respond appropriately.

A problem with well-defined responses, communication paths and authority hierarchies is the introduction of rigidity in organizational structure and response. Merton (1940) recognized that rigidity[13] in organizations results in: 1) less personalized relationships between the organization and customer, 2) internalization by workers of organization rules, 3) increase in categorization as a decision-rule. Hence, outmoded programs of decision resulting in restricted and sometimes inappropriate action are used without modification. As a result, customer dissatisfaction increases. Uncertainty becomes unacceptable in rigid organizations when the amount of exceptions passed up exceed the organization's capacity to respond. At that point the system grinds to a halt.

3.5.3.1 Uncertainty and Hierarchies

Galbraith describes three methods that reduce the upward flow of exception information in a hierarchical organization. The goal is to localize the decision making at the level that the uncertainty is introduced or as close to that level as possible, decreasing the amount of work in the higher levels of the organization.

Consider the following example. There exists a manufacturing unit that produces a product. In an "optimal" organization without uncertainty, it is given the exact amount of materials, personnel, and machines necessary to produce the product and the task is expected to be finished in minimum time. All these assumptions and expectations must be true to solve the coordination problem; i.e., that this unit can be coordinated with other units in the organization.

[13] Inability to adapt to new (novel) situations.

The first method described by Galbraith for reducing the upward flow of exception handling is the use of *Slack Resources*. Resources can take the form of time to complete the task, number of workers assigned, extra machinery, extra materials, and so on. For example, if a machine in a manufacturing unit breaks down, this unanticipated stimulus can be handled by increasing the expected time for the completion of the task (it takes time to fix the machine), or having a spare machine on hand to take over the load. Another form of slack is the use of intermediate inventories. Multiple contingent processes (steps) can be separated and their contingency reduced by accumulating slack inventories between steps. If one step ceases, there are enough slack resources for the others to work with until the missing step commences.

Self-Containment is the second method of uncertainty reduction. This was discussed briefly in section 3.4. A unit in an industrial organization may have to interact with other units to carry out its job. Normally this interaction is pre-programmed. With the introduction of uncertainty, programs become inoperable. Verifying the operability of a proposed program result in increased interactions. If the manufacturing unit receives a rush order, it will have to check with the materials unit to see if there are enough materials in stock. This may require the materials unit to juggle its allocations to other units and coordinate this change with them. If all units have to deal with this type of uncertainty, these interactions blossom at an exponential rate. Self-containment reduces interaction hence coordination by grouping together units that frequently interact. Implied in this grouping is the reduction of interactions outside of the group; intra-group interaction remains high. For example, the manufacturing unit can be given its own (internal) resource department which procures and allocates resources for the manufacturing unit only.

The third method of uncertainty reduction is the initiation of *Lateral Relations*: the creation of decision lines across the hierarchy. Given two units in an organization that must interact often, in the face of uncertainty, higher efficiency results if their interactions do not have to be controlled by a mutual superior. Instead, representatives of the units involved can form a committee that meets to discuss their problems. Hence, decisions initiated by uncertainty are not propagated up the hierarchy but are dealt with by a committee representing the concerned units and possessing full information[14] to make a valid decision. The success of this method rests on two key factors, the availability of complete information via the committee members, and the success of *Group Problem-Solving*. Group problem-solving has been shown to be quite successful given the proper operating

[14]The information pooled by the members of a committee should be enough to make valid, productive decisions.

conditions.

3.5.3.2 Uncertainty and Markets

The existance of environmental uncertainty in markets often invalidate contracts. For example, if a car manufacturer contracted for engines, and no one is buying their cars, then there is no longer a need for tthe engines.

Reducing uncertainty in the market place requires more complicated contracts; contingencies for all possible future events must be specified. But the complexity of the contract is limited by bounded rationality and environmental uncertainty. Hence long term contracts are replaced by short-term spot-market contracts. A second method of reducing uncertainty is through *monitoring* of contracts. But depending on the contract, monitoring may be too costly or infeasible.

Uncertainty in the market can be reduced by replacing contracts with employment relations (vertical integration). This allows the decision maker maximum flexibility in task assignments. The typical employment relation in an organization provides such flexibility. An employee is given a general job description. When where and how is up to the employer. At the other extreme is the long-term contract. Once signed it cannot be changed without penalty.

In summary, attempts to reduce environment uncertainty in hierarchies and markets fall into one of four categories:

1. Increased information flow within interested groups (lateral relations, monitoring).
2. Decreased resource dependence (slack).
3. Increased information processing capabilities.
4. Greater adaptation to a changing environment through increased control of functions (vertical integration, authority relations).

3.5.4 Behavior Uncertainty

An organization is composed of complex objects called people. They have their own needs, wants, desires, and problems. With all these complexities, a person must conform to tasks whose prescription often ignores these characteristics. It is obvious then that employing a person to do a task does not mean he will do it. That is, the behavior exhibited by a person in an organization can be uncertain. In the following, different types of behavioral uncertainties are examined from an organizational viewpoint.

3.5.4.1 Motivation

Humans, as individual free thinkers, choose actions, when given the choice, that best coincide with their personal goals. This is the crux of the *motivation* problem. An organization in order to achieve its goal(s) must induce its participants to act in ways best for the organization. To achieve this, an organization must either:

1. Make the participant assimilate organizational goals and/or maintain them.
2. Penalize detrimental behavior.
3. Reduce the set of alternative actions available to the participant.

For participants to assimilate organizational goals, either they must have them when they entered the organization initially, or had their goals changed by education or peer pressure, or were rewarded when correct actions were executed.

Maintaining organizational goals requires maintaining participant satisfaction. An analysis of satisfaction entails an enumeration of problems that may occur. Examples are: participants acquiring new external goals; dissatisfaction with position in organization; lack of rewards, visibility, interaction; etc. Dissatisfaction with organizational position may result when initiatives, views, or disagreements are not given adequate attention. To alleviate the problem, alternate methods of communication within the organization must be made available when traditional modes fail. Two way flow of information to explain and question decisions should be made available.

Little attention has been paid to *motivation* in computer programs. But the advent of the modular approach to constructing complex programs and the limitations on resources (e.g., processors) will necessitate a module's ability to decide when and what problems to work on. Presently, such decisions are built into the control structure (e.g., a priority queue of requests for a resource in a monitor), or left to modules such as FOCUS to decide for them. The PUP6 system requires each BEING to decide when to participate. This is done by partitioning the types of questions being asked, and having a BEING recognize the type of question it may answer. In this case there is a single automatic response. If responses were longer and more complex, hence more costly, and there were many questions to be answered, a BEING or module would have decide what questions it should respond to, and what type of response to make.

The bases for such a decision are:

1. How does the question or command coincide with its own processing goals as defined by the organization and/or some outside source?

2. If a response is to be made, what are the alternative responses available?
3. What are the consequences of these alternative responses?

(1) is used to rate the alternatives and consequences of (2) and (3). In most operating systems, the prevention of deadlock requires a careful evaluation of (2) and (3) while there exists a single goal (deadlock prevention). The introduction of conjunctive, conflicting goals complicates the decision problem. WIZARD (McKeown, 1976), the word verifier in Hearsay-II, has two goals: to verify words, and to use as little time and space as possible. When a large number of words are to be verified, a decision has to be made as to how many words should be positively verified and placed on the blackboard. Verification and placement is costly.

Computer systems constructed today, carefully tailor the goal of a process. This tailoring eliminates goal conflicts. We believe that goal differences will soon appear in processes within complex systems.

3.5.4.2 Conflict

March and Simon describe conflict in business organizations as taking place both at the individual and group (unit) level. The source of conflict may stem from differences in *goals* or differences in *reality perception*, i.e., the information consumed by each individual or group. Hence the choice of direction that an organization can take is complicated by the unacceptability, incomparability, or uncertainty in the available alternatives. The organization is led to search for new, more viable alternatives that satisfy the conflicting views.

Goal conflicts arise when an individual or group views the direction of its labors to be different from others. This difference arises from 1) differences in goals provided by the organization (e.g., research vs. development), 2) goals provided by outside organizations (e.g., professional societies), and 3) individual beliefs. Problems that appear with size are the distortion of goals among units in the organization. One manifestation of goal distortion is program persistence. A unit's goal becomes its own survival whether it is justified by the organization's goal or not.

March and Simon depict the reactions to conflict in organizations as falling into two categories. The first is *analytical*. Analytical methods use either *problem-solving* to discover new alternatives or choose an old one. It requires a careful analysis of the information and its comparison with the varying goals. The final decision being acceptable to all parties concerned.

The second category is *bargaining*. It is a form of group-problem solving where differences are subjugated and ignored for the betterment of the organizational goal. This subjugation is evenly distributed among the differing parties. The final decision does not satisfy all constraints imposed by the groups but satisfies enough to make it palatable. An example of bargaining can be found in both union-corporation talks, and intra-government decisions.

Conflict also arises in inter-organization transactions. In a market environment each organization pursues their own goals. Only when there is a need for services or products do contractual agreements arise. Since an organization is just the extension of its human administrators, divergent goals may result in opportunistic behavior. Taking advantage of information impactness, first-mover opportunity etc., inequitable contractual terms may result.

3.5.4.3 Opportunism

Opportunism was defined earlier as occurring when a party to a transaction takes advantage by making self-disbelieved threats or promises, or witholds information. Opportunism can occur both in a hierarchy and in the market. In the hierarchy, opportunism can be reduced using the motivation techniques described in section 3.5.4.1. In the market, methods such as law creation and inforcement, and monitoring agencies (e.g., better bussiness bureau) are applicable.

Systems such as Hearsay-II and PUP6 have already displayed the module capability of making independent decisions for when to participate. In Hearsay-II, the the final choice of action to execute was made by FOCUS -- enforced goal sharing. In PUP6, each BEING decided on its own when to participate, but they were carefully designed to *cooperate*. It is a small step to alter these systems to display differing reality perceptions and methods of action; a module can display more independent action, even opportunistic action. With the advent of independent hosts on a network, the problem is no longer hypothetical.

3.6 Structural Continuum

Environmental and human factors combine to cause the problems of uncertainty and complexity. The previous sections described their effects on organization structure. The purpose of this section is to show that the effects of uncertainty and complexity are to shift an organization's structure in some direction along the structure continuum (fig. 3.6).

3.6.1 Complexity

The cause of the transition from single-person to group to simple hierarchy was ascribed to capacity excesses stemming from bounded rationality. As tasks become larger, processing

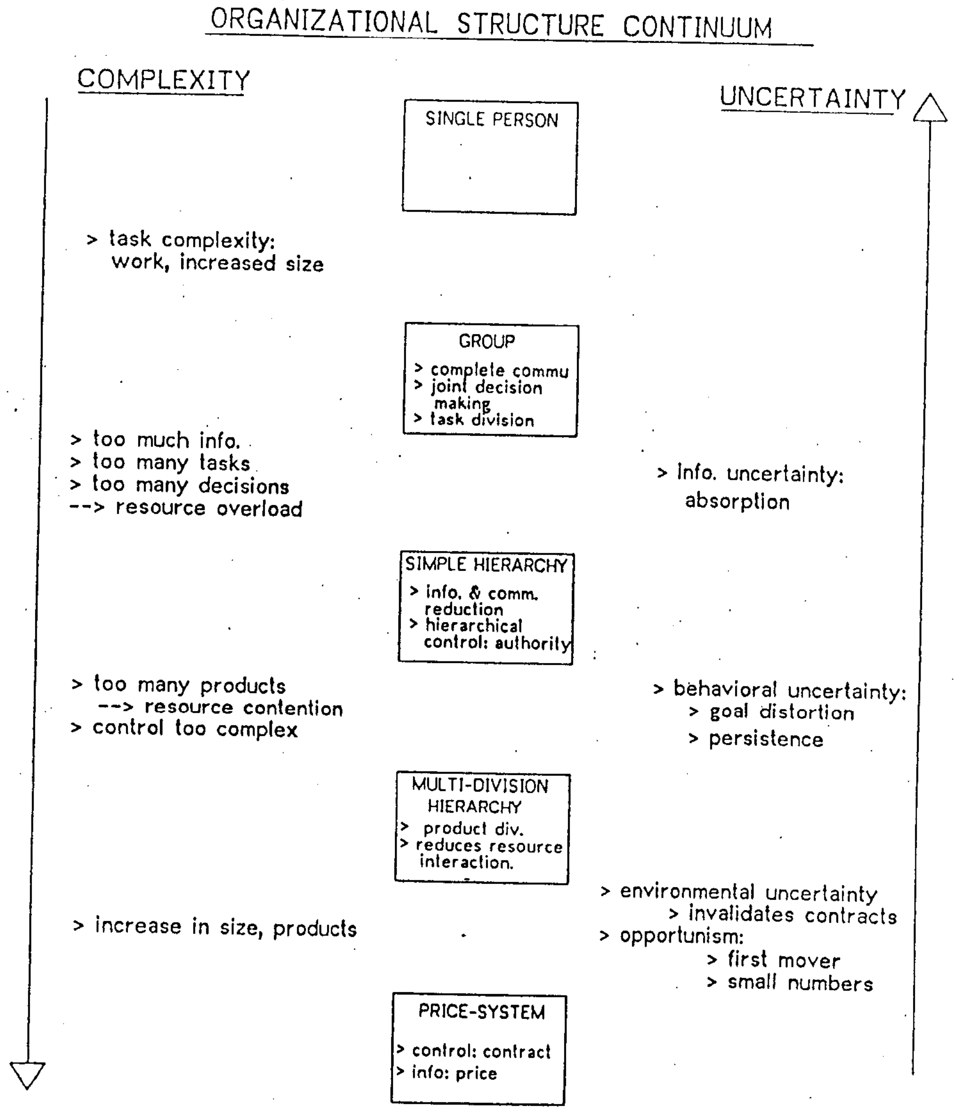


FIGURE 3.6

capacities are exceeded requiring task decomposition and allocation. Groups fail when information sharing exceeds capacity (resources), and decision making and coordination becomes too complex. Simple and uniform hierarchies utilizing information reduction techniques and authority relations appear. Uniform hierarchies usually suffice until the organization grows to a point of producing multiple products. As a result, interaction among modules reaches a thrashing level. This is due to competition among organization participants (departments, people, modules) for other participant's attention (e.g., competition among different products for the same resources). The multi-division hierarchy deals with this problem by decomposing the organization along product lines so that key departments (modules) are duplicated for each product division.

The next step in reducing information and control complexity is to contract for products and services in the market place. Information is reduced to a single variable: price.

3.6.2 Uncertainty

Uncertainty causes a structural transition in the opposite direction of complexity.

The smooth operation of markets requires marginal pricing, little uncertainty, and no opportunistic bargaining in small-number situations. Market uncertainty may invalidate contracts before they expire: materials may not be available, prices increase a strike occurs, etc. Insuring uncertain events requires complex contingent contracts. But at some point, ccounting for all possible contingences becomes too costly. Bounded rationality also limits ontingent contracts. Contract writers cannot foresee all possible contingencies or assimilate e necessary information to do so. Hence uncertainty in market environments must be small.

Reducing information to a single signal: price, results in information impactedness which ens the door for opportunistic behavior. Opportunism will succeed only if small-numbers rgaining situation obtains. Barring that, competition among market participants should enuate opportunistic behavior since no advantage can be attained.

ong term contracts in the market place are not feasible due to uncertainty and bounded onality. Under such conditions it is better to make *spot (short-term) contracts*. Spot racting allows organizations to sequentially adapt to changing environments. This ces the information and certainty required in writing a contract. A problem immediately rs negating the satisfiability of spot contracting. That is, the initial contractor obtains mover advantages resulting in small-numbers bargaining.

uncertainty increases in the market place and bounded rationality reduces the markets to contract accordingly, and opportunism appears in first-mover advantages, an

alternate form of organization must appear. Such a form is the *collective organization*. Market participants are integrated into a single organization to cooperate in achieving a single goal. Hence bargaining costs are lowered and idiosyncratic tasks are undertaken without risk of exploitation. A collective approach allows greater adaptability to uncertain environments since formal contracts do not exist and the collective jointly decides in a sequential fashion how best to adapt to the current situation. Opportunism does not appear since it is a joint venture. Information impactness is reduced by sharing information among the group.

The transition from the market (as typified in the above structures) to a hierarchical organization has been called *vertical integration*. Vertical integration is the process of adding to a hierarchical organization a product or service that was originally contracted for in the market. Uncertainty, information impactness, opportunism, etc. are the attributes in the market that have to be considered when analysing the cost of contracting versus integrating into the firm.

As uncertainty increases, the transition from a heterarchy to a hierarchy becomes preferable. The attributes of a hierarchy that support this are (Williamson, 1975, p. 40):

1. In circumstances where complex, contingent claims contracts are infeasible and sequential spot markets are hazardous, internal organization facilitates adaptive, sequential decision making, thereby to economize on bounded rationality.
2. Faced with present or prospective small-numbers exchange relations, internal organization serves to attenuate opportunism.
3. Convergent expectations are promoted, which reduces uncertainty.
4. Conditions of information impactedness are more easily overcome and, when they appear, are less likely to give rise to strategic behavior.
5. A more satisfying trading atmosphere sometimes obtains.

More importantly, hierarchies are not bound to particular courses of action. Due to the employment relation, employees have with the firm, and the firm's ability to control resource allocation, unexpected situations can be dynamically adapted to; employees assigned new tasks, resources assigned to different products. This flexibility is not typically available in contracted relations. Finally, the type of hierarchy created is dependent upon size and number of products produced, which in turn determines the organization's complexity.

3.7 Observations

Distributed systems and human organizations both share the problem of decomposing tasks, allocating resources and assigning control. The approaches taken by organization theory appear applicable to distributed systems.

Complexity was shown to be an important factor in structuring an organization. But in human organizatons complexity was attributed to bounded rationality. It was argued that resource limitations in computer systems effect bounded rationality. The transaction is the primary tool for measuring complexity. Once complexity is isolated, heuristics such as the structural continuum, near-decomposability, information abstaction, and product divisions aid the design process. But a lot remains to be done. The detection and measurment of complexity is a poorly understood problem in both business and program design.

Uncertainty is an important problem in human organizations. Uncertainty in information and algorithm has appeared in applications such as speech, vision and medicine. In each, the data are probablistic and the algorithms must recover from incorrect decisions. Examples of decision recovery methods are: backtracking in Planner (Hewitt, 1971); multiple competing hypotheses, scheduled knowledge source invocation, focus of attention in Hearsay-II; and multiple interpretations in relaxation.

The problem of behavioral uncertainty has not been considered in computer systems. Such a problem can occur in hierarchical organizations but is more prevalent in market situations. The question can be asked whether distributed systems will ever be organized as markets. Such a possibility is near.

Reduced communication in markets (i.e., price) implies reduced information inviting information impactedness conditions. If modules can be hard-wired to share the same goal, then opportunistic advantages due to an information imbalance will not occur. But when modules have the freedom to choose when and where to do processing, methods of pre-rating module performance (e.g., similar to a credit agency) and monitoring of module performance are required to reduce opportunistic behavior. Transaction analysis suggests the latter can be too costly in the market hence requiring the integration of a market function into an organization.

It has been found that the rigidity of true hierarchies and the flexibility of true heterarchies impose problems. Hence hybrid organizations seem useful. Inclusion of lateral relations and group problem-solving moderate hierarchical rigidity.

An important view underlying this paper is that modules, processes, tasks act like humans

in an organization. Consequently, the problem of motivation, a cause of behavioral uncertainty, must be considered.

Computer systems constructed today, carefully tailor the goal of a module. This tailoring eliminates goal conflicts. We believe that goal differences will soon appear in modules within complex systems.

3.8 Conclusion

Early computer programs bore little resemblence to human organizations. But as the problems attacked grew in size, resource limitations appeared and prevented the success of programmed solutions. Resource limitations can be viewed as the cause of bounded rationality whose effects appear in programs as in businesses. One can view the work in Artificial Intelligence as attempts circumvent resource limitations. Hence, it is not surprising to find programs that (attempt to) exhibit "intelligence" that also display characteristics of human organizations. Systems such as Hearsay-II and PUP6 have shown trends that reflect human organizations and human problem-solving methods. These trends have resulted in modules that contain problem-solving characteristics similar to humans.

By applying an organization theory view to software design, new insights have been achieved. Complexity and uncertainty are important in the structural analysis of a system. The solutions described by organization theory are interesting and useful but not rigorous. Better methods of measuring complexity and uncertainty must be found. Whether these measures will be derived from experimental organization theory or distributed system analysis remains to be seen.

4. Economic Team Decision Theory

Analyses of organizational structure are descriptive (March & Simon, 1958; Galbraith, 1973; Arrow, 1974). Normative analyses of organizations have been precluded because of the complexity and the incompleteness of specification of human organizations. The availability of computer program organizations of moderate complexity and complete specification suggest that normative approaches to organizational analysis may be tractable in the program organization paradigm. This chapter lays the foundation of such an analysis by mapping economic theories of organizations onto program models. The mapping enables a cross-pollination of the two fields. Economic theories are applied to program organizations, and problems in mapping suggest changes to economic models of organizations. As a result, a methodology for measuring both the structure and knowledge organization is derived.

The application of economic theories investigated here are in the area of program decomposition measurement. Given a body of knowledge about how to solve a particular problem, this knowledge can be decomposed functionally (e.g., functional division in organizations) into a set of procedures. The composition of these procedures provide a theory of how to solve the particular problem. Through the application of economic models, we can measure the efficacy of a theory (decomposition). By measuring where information flows (data) and what decisions are made (algorithm) within a theory (decomposition) comparative analyses among alternate theories can be made.

The following sections first describe the economic organization models, then the mapping to program organizations is described, followed by a discussion of the relation between the two fields.

4.1 Economic Theory

We focus on the economic models studied by Marschak and Radner in their book The Economic Theory of Teams. In particular, the network model of organizations which is an outgrowth of Team Decision Theory is investigated. Team decision theory analyses the conditions under which optimum decisions can be made amongst a group of decision makers. Members of the team share the same goals which are analytically realized in a single utility function. By analysing team member decision functions and the information upon which they base their decisions, optimum organizations of information flow and decision making can be discovered.

4.1.1 Single Person Decision Theory

Classical decision theory is concerned with the maximization of utility; where utility is some measure of the "goodness" of an event. The event of concern is the decision to be made. Given a state of the world $X \epsilon X$, where X is the set of possible world states, the decision function α is a function of the signal y, an abstraction of X.

$a = \alpha(y) \quad y \in Y$ (Y is the set of abstractions on X)

The result of the decision is an action a. The utility (payoff) of a decision is defined by

$\omega(X,a)$

which is futher defined as

$\omega(X,\alpha(y))$

ω can be described as the payoff of decision α.

The set Y of signals defines a set of abstractions on X. A particular partition of X is called an information structure and is denoted as η. The combination of an information structure and a decision function, (η,α), is called the organizational form.

Incorporating the information structure into the analysis, the utility function for a particular decision function and information structure can be redefined as

$$EU = \sum_{X} \omega(X,\alpha(\eta(X)))\ \phi(X)$$

where $\phi(X)$ is the probability of world state X. In single-person decision theory, analysis centers around finding the optimal decision function that maximizes expected utility given a particular utility function ω.

4.1.2 Team Decision Theory

A multi-person team is characterized as members who have identical interests. Hence, goal conflict does not occur and overall utility is to be maximized. To accommodate multiple members and actions, the above analysis is extended. The information structure becomes a vector of information structures; one structure for each team member. The same is true for the decision function

$\eta = (\eta_1,\eta_2, ..., \eta_n)$ team information structure

$\alpha = (\alpha_1, \alpha_2, ..., \alpha_m)$ team decision rule
where m is the size of the team.

The team gross payoff is defined as:

$$u = \omega(\mathcal{X}, \alpha(\eta(\mathcal{X})))$$

and the gross expected payoff as

$$EU = \Omega(\eta, \alpha) = E\omega(\mathcal{X}, \alpha(\eta(\mathcal{X})))$$

The maximum gross expected payoff is defined as

$$\Omega'(\eta) = \underset{\alpha}{\mathrm{Max}}\ \Omega(\eta, \alpha)$$

Information incurs two costs, observation and communication, in addition to the cost of making the decision. Hence the net payoff is defined as:

$$E\omega(\mathcal{X}, \alpha(\eta(\mathcal{X}))) - E\gamma(\mathcal{X}, \alpha, \eta)$$

where γ is the cost of α and η at state $\mathcal{X}$.

The analysis of what are optimal decision functions rests on the utility function chosen. Marschak and Radner's analysis focused on quadratic payoff functions of the form:

$$\omega(\mathcal{X}, a) = \mu_0 + 2a'\mu(\mathcal{X}) - a'Qa$$

a - vector of team actions.
$\mathcal{X}$ - world state
m - number of team members.
Q - positive definite m×m matrix.
μ - vector function of state x.
μ_0 - constant (=0).

That is, the utility is a quadratic function of their actions.

It should be noted that the Q matrix defines the measure of interaction between actions.

$$q_{ij} = \frac{\partial^2\omega(\mathcal{X}, a)}{\partial a_i \partial a_j}$$

Two types of information structures are distinguished: complete information occurs when

all information is completely communicated. This results in an optimal decision function but with increased cost of communication. Routine information occurs when no observations are made, and decisions are based solely on world state probabilities. Hence, the value of an information structure η, can be defined as

$$\Omega'(\eta) - \Omega'(\eta_r)$$

where η_r is the routine information structure.

4.1.3 Network Team Decision Models

The team decision models described above can be represented using a network model. Representing organizational form as a network enables a more thorough analysis of organizational structure and the costs entailed. This follows from the greater variety of models that the network can represent and the ease in which they are depicted.

A node in the network is an element which transforms input messages to output messages. External messages (information) entering the network are called observations, and messages exiting the network (to nature) are called actions. Communication involves sending messages from an element to another along a directed arc.

Formally, each element has input:

$$(z', e, b')$$

where

z' = message from outside world

e = stochastic nature of element (noise variable)

b' = combined input messages from other elements.

Messages may pass between any two elements.

B_{ij} denotes the set of possible messages that can be sent from element i to element j. B_{i0} denotes the set of possible messages from element i to nature and B_{0i} denotes the set of possible messages from nature to element i. An empty B_{ij} denotes a non-existent communication path.

B_i is the set of possible output messages of element i

$$B_i = \prod_{j=0}^{m} B_{ij} \quad \text{(m is the number of elements)}$$

B'_i is the set of possible input messages

$$B'_i = \prod_{k=0}^{m} B_{ki}$$

For each element, a task function $\beta_i = (\beta_{i0},\ldots,\beta_{im})$ is defined. β_i transforms input messages to output messages. A system is defined by

$$b_{ij} = \beta_{ij}(b_{0i},\ldots,b_{mi})\ i=1,\ldots,m;\ j=0,\ldots,m$$

$$b_{0j} = \beta_{0j}(X)\ j=1,\ldots,m$$

An important restriction on B is that there exists a numbering of the elements so that each element outputs only to a higher numbered element and there are no feedback loops in the network. This insures the system being well-defined and consistent.

a) B_{ij} is empty for $1 \le j \le i$

b) B_{00} is empty.

Now that the network model is defined, it can be shown the conditions under which the network model is equivalent to the team decision theory model. If the following are given:

1) The message sets B_{ij} $i,j=0,\ldots,m$

2) The task functions β_{ij} $i=0,\ldots,m$ $j=1,\ldots,m$

3) A set B_{i0} $i=1,\ldots,m$ of feasible task functions β_{i0}.

The network described corresponds to the team decision model discussed earlier: the setting of B_{ij} and β_{ij} $(j \ne 0)$ determine a unique information structure η for the organization. All that is to be determined is the β_{i0} which determines the message to the external world, i.e., the action. Hence choosing β_{i0} $i=1,\ldots,m$ is equivalent to choosing $\alpha = (\alpha_1,\ldots,\alpha_m)$ such that

$\alpha_i \in \mathrm{B}_{i0}$ $i=1,\ldots,m$

Reducing the givens results in more degrees of freedom in network structure.

4.2 Mapping Economic Theory onto Organization Structures

This section outlines the possible relationships between the decision theories and network model described earlier, and the organizational structures of computer programs. In carrying out this analysis we reinterpret the decision theory models are interepreted in light of the organizational primitives: modules, communication channels, and data modules. We draw heavily on the Hearsay-II and Production System (Newell, 1973b) architectures for inspiration

and formalism.

4.2.1 Problems

The mapping of economic theory presents problems both in interpretation and feasibility. The following is a short list of the most important:

First, the functions used in decision models are all real valued (e.g., α, η). Except for some obvious categories of applications, a program's computation is symbolic. As a result, a program model interpretation of the decision function α and the information structure η is not obvious.

Secondly, the analysis of organizational situations requires the depiction of all variables. Few domains are defined well enough, precluding the denotation of relevant variables.

Finally, in addition to variables, probabilities of each joint and conditional state between observed variables must be known in order to make decisions of expectation. Such measures are incomplete, if available at all, in most domains.

4.2.2 Mapping the Network Model

In this section we map the Economic Theory Network Model (ETNM) is mapped onto a particular program model creating a Program Organization Network Model (PONM). The program model is a combination of production system and Hearsay-II architectures. Production rules were chosen as the primary representation for decision processes because they are a single uniform representation which allows standardized information to be associated with each rule. This will become evident as the mapping is described. The mapping is begun by considering a single node network, then showing how a multi-node network can be derived. The derivation defines a procedure for constructing network models of programs.

4.2.2.1 Single Node Network

In the single node network, there is a single decision function observing the state of the world and acting upon it. This is equivalent to a production system where all the production rules comprise the decision function, and the world state is represented by short term memory (STM). Making a decision is equivalent to choosing a production rule and executing it. As a decision is based on the world state X, so a production rule is chosen based on the state of STM. The action a may alter the world state just as the execution of a production rule may alter STM.

4.2.2.2 Multi-Node Networks

The creation of multi-node networks is viewed as a decomposition task. Given a "single-node" production system program for some application, the system is decomposed into connected production sub-systems, in a sense similar to the stepwise-refinement decomposition of programs (Wirth, 1971), and the production system networks of Zisman (1978).

The initial set of productions is partitioned by classifying the left-hand-side (LHS) and right-hand-side (RHS) according to the types of information they act upon. For example, viewing each knowledge source in Hearsay-II as a production rule, the "stimulus-response frame" defines the classes of information the KS LHS inputs and the KS instantiation outputs (RHS). Hence, all knowledge sources that input information at the lexical level and output at the phrasal level would be in the same partition. A KS that inputs at the lexical level and outputs at the word-sequence level would be in another partition. Each production rule partition defines a node in the decision network. The fineness of the partitioning depends on the fineness of the information classes.

A directed arc exists between two nodes if the output of one node is in the same information class as the input of another. An information structure flows along the direction of the arc.

4.2.2.3 An Example

Consider the following set of production rules:

1 $a_1 \wedge a_2 \wedge b_1 \text{==>} c_1$

2 $x_1 \wedge x_2 \text{==>} a_1$

3 $x_3 \vee x_4 \text{==>} b_1$

4 $a_3 \wedge b_2 \text{==>} c_2$

We define the following information classes:

$(\forall i)(a_i \in A)$

$(\forall i)(b_i \in B)$

$(\forall i)(c_i \in C)$

$(\forall i)(x_i \in X)$

Applying the above information classes, the production rules can be rewritten (classified) as:

1 $A \wedge B \text{==>} C$

2 $X \text{==>} A$

3 $X \text{==>} B$

4 $A \wedge B \text{==>} C$

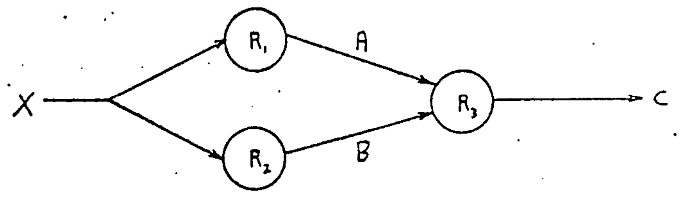
Which induces the following partition on productions:

$R_1 = \{ 2 \}$

$R_2 = \{ 3 \}$

$R_3 = \{ 1, 4 \}$

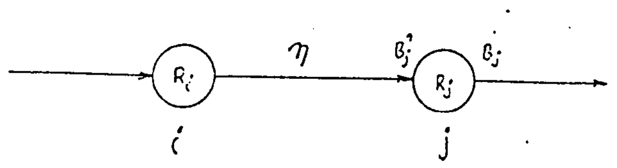
and can be represented in network form:



Depending on the fineness of the information classes, the network can be expanded or contracted.

4.2.2.4 Formalizing the Network

Consider a 2-node serial network:



With:

R_i = set of productions at node i.

r_{ij} = the jth production rule in node i (R_i).

B'_j = set of input information classes of jth node.

B_j = set of output information classes of jth node.

L^k = the kth information class (lexicon).

l^k_i = ith element of the kth information class.

m = total number of nodes in the network.

b_{ij} = the node connection matrix. Zero if node i does not output to node j. One if it does.

L^k is defined as the kth information class. An information class is finite. l^k_i as the ith element of the kth class. L^k is equivalent to a lexicon in Hearsay-II, defining an information class (e.g., phrasal lexicon). B'_j is a set of input classes for node j, and B_j is the set of output classes. Any information, l, passed between two nodes i,j must be an element of the ith nodes's output classes, $l \in B_i$, and be an element of the jth node's input classes, $l \in B'_j$.[15]

The major differences between ETNM and PONM are:

1. The r_{ij} in PONM serves a similar function as the transform β_{ij} in ETNM. But there

[15] The correct description is $(\exists k)\ ((L^k \in B'_j) \wedge (l \in L^k))$

is not necessarily a correspondence between r_{ij} and jth node. The analog of β_{ij} is described in the next section.

2. The functions α and β are reinterpreted as <u>certainty</u> functions in the next section.

3. b_{ij} defines a connection matrix based on output-input information class intersection.

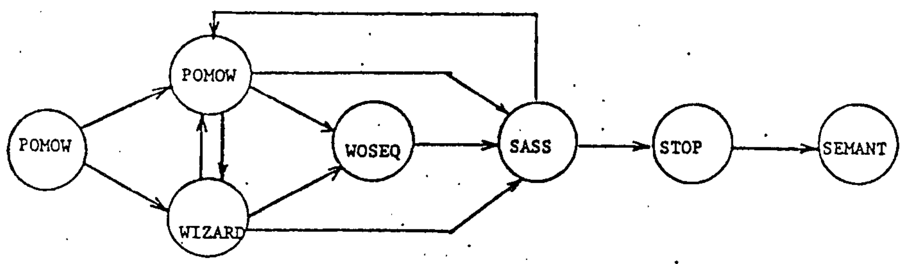
4. Loops (feedback) are allowed in the PONM.

4.2.3 Interpretation of Numeric Functions

It was mentioned earlier that the economic theory analysis maximized real functions. The interpretation of these functions in program organizations is difficult due to the symbolic nature of program data. Marschak and Radner's analysis of team decisions with quadratic utility show that under certain conditions the optimal decision function is a linear function of the information structure. Since information structures and decision productions in programs are symbolic, the optimality condition cannot be defined directly on the program's input and output. For example, the syntax and semantics knowledge source in Hearsay-II (SASS) consumed words and phrases and produced new phrases (Hayes-Roth et al., 1978). How do you create and interpret a linear function of words and phrases?

SASS did not produce only new phrases, but each phrase had a *certainty* rating assigned to it. SASS produced a set of outputs with associated certainty ratings which reflected the certainty of the input data and the knowledge contained in the knowledge source. Many applications (e.g., Speech, Vision, Medicine) share the same approach of producing multiple weighted results. This reflects the incompleteness and errorfulness of the algorithms used to model these domains.

Many systems in these domains can be viewed as consuming signal data and successively transforming it until a conclusion is reached (e.g., a sentence, picture or diagnosis). The following is such a network transformation model of Hearsay-II.



Each KS transforms input symbols to output symbols and assigns each a certainty rating. Each KS can be viewed as producing two outputs: a symbol and a judgement of the certainty (correctness) of the symbol.

The approach taken here in interpreting the decision function α is as a measure of certainty. Hence, η is defined as the information structure's certainty, and α as the decision function's certainty. That is, η and α measure the correctness of program information and algorithm respectively.

A node was defined as inputting an element in B' and outputting an element in B. In Hearsay-II, each KS instantiation can be viewed as satisfying the above condition. But during the running of the system, a KS is instantiated more than once, outputting many information elements. *The network model constructed here is static.* It models the system's structure and expected processing, not depicting individual transactions. Hence, the numeric functions are interpreted as network statistics. The decision function α is defined as a function whose domain is the input certainty, and range the output certainty. That is, it computes the expected certainty of the node's output information imparted by the decision production rules upon the input. The information structure function η^k is a certainty vector over the elements of an information class L^k. Any element in an information class not outputted or communicated has a certainty of zero.

The decision function α_i is derived from the production rules at the ith node. Each

production rule r_{ij} has a certainty transform β_{ij}. The certainty transform defines the certainty of the output based on input. α_i can be calculated as the expectation of the rule certainty transforms:

$$\alpha_i = \underset{j}{E}\beta_{ij}(\eta'_i)$$

$$= \sum_j w_j \beta_{ij}(\eta'_i)$$

where

η'_i = the certainty of the input of node i.

η_i = the certainty of node i's output.

η_i^k = the certainty of the kth information class (L_k) output by node i.

w_j = probability of rule j being executed.

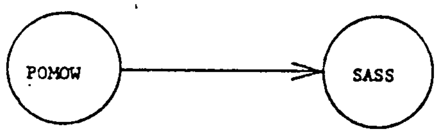
4.2.4 Adding Costs to Network Analysis

In economic decision theory, the utility of a network is defined on the actions emitted by one or more nodes. In our interpretation, the network functions are functions of certainty, consequently the utility is interpreted as a function of certainty. Since the decision functions compute the change in information certainty, the utility function should take into account the expected change in certainty imparted by the whole network.

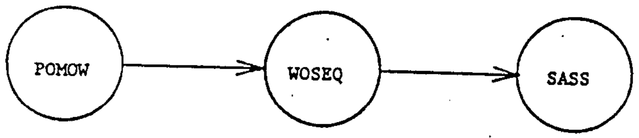
Before going further it is interesting to note that the goal of this analysis is to understand and discover better program and knowledge organizations. The utility function allows us to compare alternate network structures, i.e., rule decompositions and information classifications. That is, given a particular network, organizational form (η, α) its utility can be measured. By changing α or η a new network is produced and the utility measured.

The question remains as to whether this model is the best approach to program organization analysis. It is not. An example from Hearsay-II will illustrate the point. The

Initial configuration of Hearsay-II had the POMOW[16] module outputting information to SASS[17]



This configuration did not work. The final configuration introduced the WOSEQ[18] module. WOSEQ incorporated a subset of the knowledge of SASS and acted as a filter on the information that SASS acted upon.



In essence, the final Hearsay-II configuration was a restructuring that did not affect the certainty of the original KSs (α_is) but limited the processing of SASS to a few good information elements. What plagued the initial configuration was the <u>resource cost</u> of executing SASS and the <u>amount of data</u> it was applied to. The introduction of WOSEQ reduced costs.

Costs were mentioned in the original economic models but were not fully developed. The analysis of program organizations require that cost of information and decision be taken into

[16]POMOW (Smith, 1976) is the word hypothesization knowledge source. It constructs words from syllabic and phonemic information.

[17]SASS (Hayes-Roth, Mostow & Fox, 1978) is the syntax and semantics knowledge source. It constructs grammatically acceptable phrases from temporally adjacent words.

[18]WOSEQ (Lesser et al, 1977) constructs pair-wise grammatically correct word phrases from temporally adjacent words.

account. To achieve this the following alterations to the network model are introduced:

$S(B_i)$ = expected number of information elements output by node i.

$S(B'_i)$ = expected number of information elements input by node i.

C_i = cost of node i processing an information element.

The problem of deriving a node's cost function C_i is similar to deriving its certainty function α_i. Production rules are composed of primitive operations to which space-time costs can be attached. c_{ij} is defined as the cost of testing and executing the jth rule of node i. If each time an information element is processed, all the LHSs of the rules at a node are tested and only one post-condition is executed, then the cost function can be defined as the sum of the costs of the LHSs plus the average cost of the RHSs:

$$C_i = \sum_{j=1}^{N_i} c^-_{ij} + \sum_{j=1}^{N_i} c^+_{ij} w_j$$

where

c^-_{ij} = cost of testing r_{ij}'s LHS.

c^+_{ij} = cost of executing r_{ij}'s RHS.

N_i = total number of rules at node i.

4.3 Interpreting Utility

The utility of a network can be defined now as a function of certainty and cost. A node's utility is proportional to:

utility of expected certainty change - utility of expected cost

$$u_i = f(\int \alpha_i(B'_i) - B'_i \, dB'_i) - g(S(B'_i)C_i)$$

where f and g are utility functions. The network utility is defined as the utility of the certainty imparted by the network to the output less the utility of the total costs of all the nodes:

$$U = f(\int A_m(x) - x\,dx) - g(\sum_i S(B'_i)C_i)$$

where $A_k(x)$ is the composition of certainty functions α_i in the subnet ending at node k. m is the final node of the network. x is the vector of inputs external to the subnet.

4.4 Interpreting Optimality Conditions

One of the goals of our anlysis is to apply the theoretical results in team decision theory to program organizations. One such application is presented. The following is an optimality theorem due to Marschak and Radner (1972, p. 166). Consider a two-person team with real valued actions a_1 and a_2.

Theorem 1: If the payoff function is of the form:

$$w(c,a) = l(c) + 2m_1(c)a_1 + 2m_2(c)a_2 - a_1^2 + 2qa_1a_2 - a_2^2$$

and if μ_i, y_i are normally distributed with $Ey_i=0$, Var $y_i=1$, $Ey_1y_2=r$ and $Cov(\mu_i,y_i)=d_i$ then the optimal decision functions α'_1 and α'_2 are linear.

$$\alpha'_i(y_i) = c_i + b_iy_i$$

where

$$b_i = \frac{d_1 + qrd_2}{1 - q^2r^2}$$

$$b_2 = \frac{d_2 + qrd_1}{1 - q^2r^2}$$

$$c_1 = \frac{E\mu_1 + qE\mu_2}{1 - q^2}$$

$$c_2 = \frac{E\mu_2 + qE\mu_1}{1 - q^2}$$

The linearity result can be interpreted in the following way: Given a network program model (PONM) which has two (external) outputs, a_1 and a_2, a quadratic utility function can be defined on the outputs with an interaction measure q between a_1 and a_2. If the PONM's outputs are being rated with a quadratic utility, then the theorem states that the nodes producing these outputs must have linear certainty functions α_1 and α_2 to maximize utility.

An example can be drawn from the medical diagnosis domain. Programs like Mycin and Internist produce two outputs: a diagnosis and a treatment. The PONM model states that the utility function is applied to the certainty of these two outputs. Hence the utility function distinguishes the importance of certainty in each action. It can assign more importance to a certain diagnosis than to the treatment, vice versa, or rate then equally. In the case where the utility is quadratic, the theorem states that the modules (α_1, α_2) that produce the diagnosis and treatment certainties must be linear.

A more basic question to be answered is what does it mean for a decision function to be optimal. The utility function defines the *relative* importance of each of the outputs. This, in turn, determines what nodes in the model should be producing the more accurate results. The better the certainty function α_i, the more accurate the result. Hence, a decision function α_i is optimal when it provides the level of certainty required by the utility function. And as defined earlier, the utility is the difference between the utility of the decision less the utility of the cost of achieving the decision; providing a balance between the results and the amount of resources needed to achieve the result. Hearsay-II and Prospector (Duda et al, 1978) both use linear functions while Mycin (Shortliffe, 1976) and Relaxation methods (Zucker, 1976) use non-linear functions. As the utility functions are undefined, the optimality of these functions cannot be inferred.

4.5 Network Models for Linear Certainty Functions

The function specified in section 4.3 describes theoretically how to measure the utility of a network. In reality, we are no further ahead; feedback and decision functions of any form complicate the analysis. In the following, several restricted classes of networks are modeled. Our goal is to understand the long run behavior of networks with feedback. All the classes considered are restricted to linear decision functions[19] α_i:

$$f(x) = kx$$

where k is a constant. To understand the models, the network model is modified:

[19] We choose the class of linear functions because of the optimality theorem in the previous section.

$$\begin{tikzcd}
{} \arrow[r] & \alpha_a \arrow[r] & \alpha_b \arrow[r] \arrow[l, bend right] & {}
\end{tikzcd}$$

$$\begin{tikzcd}
{} & & \boxed{\eta_3} \arrow[dl] & & \\
\boxed{\eta_1} \arrow[r] & \alpha_a \arrow[r] & \boxed{\eta_2} \arrow[r] & \alpha_b \arrow[r] \arrow[ul] & \boxed{\eta_4}
\end{tikzcd}$$

The circles still define the decision functions a_i of the network. That is each contains a set of rules. The boxes designate information H_i in particular the state of information. a_a transforms the certainties H_1 and H_3 into H_2.

$$\eta_2 = \alpha_a(\eta_1,\eta_3) = k_{a1}\eta_1 + k_{a3}\eta_3$$

and

$$\eta_3 = \alpha_b(\eta_2) = k_{b2}\eta_2$$

$$\eta_4 = \alpha_b(\eta_2) = k_{b1}\eta_2$$

A matrix representation of the network can be constructed where each element transforms the certainty of one information state into another.

$$M = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ 1 & & k_{a1} & & \\ 2 & & & k_{b2} & k_{b1} \\ 3 & & k_{a3} & & \\ 4 & & & & \end{array}$$

To model the flow of information certainty through the network, we take the prior certainties' vector of the information states (square boxes) are multiplied it by the network matrix. The result is the posterior certainties' vector. Each application of the matrix can be viewed as a pulse of certainty through the network.

The question remains as to the long term effect of the network on information states. In this model, k can range over the positive reals. If $k>1$ then certainty is imparted to the information. If $k<1$ then certainty is removed. Letting k range in this manner introduces instability in the network. Certainties may continually expand, oscillate or converge to a steady state for the outputs. If the certainty matrix has an eigenvalue equal to one and the

others less than one, then the output information states converge to unique limiting certainties.

Observations

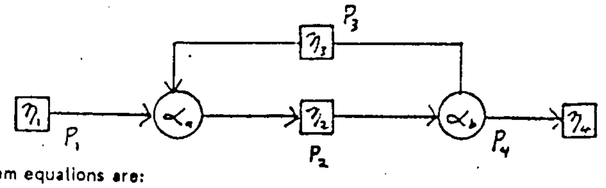
A few comments on this model are required. In each case the statistical behavior of the network over time is being modeled. Each multiplication of the network's matrix ($M^n \rightarrow M^{n+1}$) is another unit of time. As n grows, the effect of the initial input certainties are reduced, dwarfed by the matrix probabilities k. Given the stability conditions, the long run effect of the network certainty functions on the network's output can be calculated. That is, the limiting certainties (η_i) of the outputs of each certainty function α_i can be calculated. Hence, subnets requring further work may be recognized by their low output certainties.

The existence of limiting certainties in a converging network raises a critical issue. Certainty contributed by nodes not in a feedback loop have no effect in the long run. for example, the segmentor in Hearsay-II is executed once to provide rated segments. The certainty values on the segments form the basis of information certainties throughout the system. but the existence of feedback theoretically negates their effect. This does not occur in Hearsay-II because the certainty functions conserve certainty instead of descreasing or increasing (output is average of input). Though the limiting information certainties are of interest, what is of potentially more value is the change in information certainties at each pulse (multiplication) of the network (certainty matrix). Also the rate of convergence of the information certainties to their limits may be useful in determining how sensitive a system is to external (prior) certainties and how long a system should run before no further change could be expected.

Each multiplication forces all paths in the network to be exercised equally. But in typical program organizations not all paths are equally used. Hence the effect of a particular path should be moderated by the amount it is used. One approach is to reduce the contribution of a little used path upon a particular information state. That is, as the probability that a particular path from a node to an information state is used approaches zero, $p \rightarrow 0$, the coefficient of the appropriate term in the nodes certainty function is reduced, $k' = pk$.

For example, if p_i is the probability of using a particular path[20] (assume p_i is normalized) then the network can be redrawn as

[20] p_i can also be interpreted as the flow through an arc, and η_i the quality of what flows through the arc.



The system equations are:

$$\eta_2 = \alpha_a(\eta_1, \eta_3) = p_1 k_{a1} \eta_1 + p_3 k_{a3} \eta_3.$$

$$\eta_3 = \alpha_{b2}(\eta_2) = p_2 k_{b2} \eta_2$$

$$\eta_4 = \alpha_{b1}(\eta_2) = p_2 k_{b1} \eta_2$$

which has matrix form:

$$\begin{array}{ccccc} & 1 & 2 & 3 & 4 \\ 1 & & p_1 k_{a1} & & \\ 2 & & & p_2 k_{b2} & p_2 k_{b1} \\ 3 & & p_3 k_{a3} & & \\ 4 & & & & \end{array}$$

On each iteration the certainty contribution of a nodes inputs are proportional to the usage or flow through each input.

4.6 Observations and Conclusions

The previous sections have shown one approach to mapping the economic-team decision theory models onto a program organization, namely, a production system-Hearsay-II program model. In rendering the interpretation feasible, alterations of the economic model have been made:

Static Model:	The network model has been altered to describe long term statistics, hence depicting expected performance.
Feedback:	The economic network model does not include loops (feedback) though loops are typical of most organizations, business and program. PONM allows feedback because it is a static model.
Costs:	Most economic theories discuss costs (Marschak & Radner, 1972; Williamson, 1975) but do little more than that. Cost functions were introduced at the rule and node levels (rule and node costs, input-output size). Applying PONM to business organizations enables the inclusion of cost analyses.

In creating the Program Organization Network Model, constructive descriptions were provided throughout, enabling the construction of an interactive system that can assess program organizations. By specifying a body of knowledge (production rules), a classification of information, certainty transforms, rule usage weights, and rule costs, alternate networks of rules can be constructed and the utility measured to ascertain the better organization. In addition, incremental changes to rules and organization can be measured for their expected effect, though their dynamic effect in a particular situation cannot be ascertained due to the static nature of the model.

Problems still remain. These problems require human intervention in the organization process. Interestingly, these problem occur in other rule-based systems. The first problem is the acquisition of rules. This problem is truly ubiquitous. Attempts at solutions can be found in Davis (1976). A second problem is the definition of information classes. In Hearsay-II, the lexicons (classes) were generated by hand. A similar problem crops up in Sacerdoti's hierarchical planning system (1974). Rules have to be classified into a hierarchy of importance based on the types of information in the LHSs. A third problem is the acquisiton of rule certainty transforms (probabilities). Typically, these values have been provided by experts. Relevant work can be found in Shortliffe (1976) and Duda et al (1976; 1978).

Another set of problems arise when re-examining the optimality results of economic theory. Do quadratic utility functions model the utility functions of programed systems? If their applicability is in doubt then the door is opened for an examination of the types of decision functions to be considered. Perhaps linear decision functions are not optimal, then what are? The decision functions of Mycin and relaxation methods are non-linear. It would be interesting to carry out a sensitivity analysis of the utility and non-linear decision functions of Mycin and relaxation, and the utility and linear decision functions of Prospector and Hearsay-II[21]. Perhaps this would shed some light on what are the different types of utility functions of interest and when a particular type of decision function is applicable.

A question remains concerning the semantics of the term certainty. The term has been used to express the degree of support the system gives to a datum. But when a node produces a set of *competing* data with high certainty ratings, then these data, though highly rated, are equally valid, hence uncertain. The problem here is one of ***differentiability***. Can the system differentiate among competing, high certainty information? There are three ways in which this is dealt with in the model. First, the ability of a node (module) in the network to

[21] Gashnig (Duda et al, 1978) carried out a sensitivity analysis of the knowledge net in Prospector

filter information can be defined by the input S(B') and output S(B) measures. Second, if nodes are *expected* to output competing data, then there must be a means to choose among them, hence an additional node should be defined to choose among them. Third, the inability of a node to differentiate should be reflected in the node's certainty transform. If it is a poor differentiator then the certainty increase should be low.

In conclusion, a constructive method for creating and analysing program organizations has been created using the Economic Team Decision Theory Models. The two-way transfer of technology between program and economic models has been enabled. It is hoped that an interactive system can be constructed to analyse alternate program organizations. This would allow a detailed analysis of knowledge and its organization in rule-based systems. In particular, questions concerning the type of utility and certainty transforms and the effects of feedback on these systems can be analysed.

5. ODL: Organization Design Language

The previous chapters have presented a multi-faceted view of organizing large, complex programs. They have shown how problem characteristics affect program structure. In particular, chapter 3 showed the need for a complexity, uncertainty, and behavioral analysis of organizations. Interestingly, the characteristics that stimulated the analysis (e.g., bounded rationality, motivation, goal conflict, opportunism, etc.), have appeared in large programmed systems. In addition, the concept of uncertainty, its manifestation in the environment as described by organization theory, and in information and algorithm, was shown to have an important affect upon the program organization. In all cases, the analysis has been at an abstract level of program design. Concepts of goals, motivation, authority, abstraction, problem-solving, etc. have appeared through out the chapters.

The following sections describe a design language which attempts to lift the design problem from the control to the organization structure level. Language constructs are described that incorporate the concepts of the previous chapters. These constructs are added to the module, data module, and communication channel concepts discussed earlier. Next the concepts of the previous chapters are reviewed and their representation in the organization structure language are described. Finally, a portion of the Hearsay-II architecture is presented in the language.

5.1 Language Overview

The purpose of this language is to investigate how the concepts of the previous chapters could be represented. In particular, how goal-oriented behavior, uncertainty, complexity and the variety of structure can be defined.

The language has eight primary constructs.

A *module* consists of procedures, data structures and meta-descriptions of the purpose of the module.

A *goal* is a meta-description associated with a module. A module may have many goals. It describes resources, procedures, and data usage. It also defines the goal's importance to the module, the certainty with which it is achieved and its behavior.

A *data-module* is a set of memory locations with a defined structure, and access and modification routines. It also supplies monitoring routines to inform other modules of the appearance of specified data.

A *view* defines the goals of a module and regions of a data-module that can be seen externally by other modules and data-modules.

A *channel* is a pipe or link which connects modules and data modules. It specifies what information can flow along it and its direction.

A *port* is the part of a module or data-module to which the channel attaches. It defines the views available to a channel, information-type and direction of information flow.

An *Information-type* is a specification of the message format and content types.

Last, an *organization* is a template definition of a structure composed of a set of modules, data modules, channels, etc. Organizations have the same meta-descriptions as a module and can be viewed as primitive constructs.

The language definition outlines the types of constructs required by the previous chapters. It is essentially a static type definition and is incomplete. The specification of replication, dynamic creation of structures, instantiation, etc. were ignored. Lesser et al (1979), Tichy (1979), Mitchell et al. (1978) describe these details in their inter-connection languages.

5.2 Language Constructs

5.2.1 Information

Information is the primary resource of any program. It is what is acted upon and produced. We start our description of the language with a simple characterization of information and how it is communicated within the system. First, this work assumes that data representation capabilities of abstraction based languages are available in the language. This allows the creation of any abstract data-type.

```
Information-Type: <name>
          Lexicon: (Data-Structure*)
          Language:
```

Figure 5.1

There exists in the language an Information-Type. It is composed of a Lexicon and Language. The Lexicon specifies what data types can be in the Information-type. The language defines how the lexicon is composed. For example, one type of language would be for contract negotiation among market members. The lexicon would specify contract parts and the type of data that can fill each part. The language would specify the form of the contract.

Another information type could implement the accessing and communication of information among modules. Data-structure access (read/write), goal initiation, procedure starting, etc. would be messages described by the language.

Information that is passed from one module to another is passed in packets (messages), called an Information-Packet. It specifies the attributes attached to information.

```
Information-Packet: <name>
          Packet:
          Information-Type:
          Certainty:
          Producer:
          Consumer:
          View:
          Other-Attributes:
```

Figure 5.2

An Information-packet specifies the producer, consumer, and certainty attributes because of the usefulness in reducing consumer and producer uncertainty, monitoring execution and analysing information uncertainty. User defined attributes may be placed in the Other-Attributes position. The view attribute specifies under which view, at the consuming object, the message is to be interpreted.

5.2.2 Abstraction

Abstraction mechanisms are prevalent in programming languages. Their use lies in their ability to reduce the apparent complexity of functions. Typically abstraction means replacing a complex structure (function or data) by a single name. That name can be used as a primitive in other complex structures. The organization structuring abstraction mechanism differs from single-symbol abstractions in that it attempts to describe the behavior of a complex algorithm by a state-transition model.

Again it is assumed that a language for creating procedures and data structures is available. For each procedure a model is defined which abstracts the description of what a procedure consumes, produces, and the states it traverses during execution. The approach taken here is similar to that of Riddle et al (1978). It differs in the additional types of attributes associated with the model. For a more complete definition of how models are anchored to procedure descriptions see Riddle et al.

```
Model: <name>
  Object: (Procedure | Module | Organization)
  State-Variables:
  State-Labels:
  States:
  Abstraction:
    (State1 ‿ State2 ‿ ... ‿ Statem --> Staten,
     Resources consumed,
     Resources produced,
     State-Attribute: (Exception: Type)
     Transition probability,
     Certainty transform
    )*
  Test-Data:
    (Test-State:
     Test-Input:
     Test-Output:
     Test-Transitions: (State1, State2, ..., Staten)
  )*
```

Figure 5.3

The model is an abstraction of the specified object. In the model an abstraction is composed of states and transitions between them. A state in the computation is defined by a subset of state-variables with particular values and/or a particular position, state-label, in the object's code. The actual abstract model is specified by a set of productions that detail state transitions. Associated with each production is the amount of resources consumed and produced during the state transition, a probability of the transition taking place, and the change in certainty of the resources. Also arbitrary attributes may be associated with an end state. For example, an exception condition state can be described. Its interpretation is that the state causes an exception which must be handled by communicating with another module. For procedure testing and analysis, a set of input/output test data are also defined along with the actual model transitions for each test datum. The test is carried out in the environment specified in the test-state which can be arbitrarily complex.

A second model of abstraction is a data-abstraction. Typical data-abstractions are structurally defined. For example a stack is a link list of records with a pointer to the top. But the type of data abstraction needed in organization structuring is a *value-abstraction*. That is we want to associate a label with specific *data states*. For example, we may want an abstraction that corresponds to when the top element of the stack is zero. Data-abstraction specifies how a module or organization identifies and categorizes information. It also may

allow the identification of reality absorption conditions. To specify an abstraction we introduce the data-model.

```
Data-Model: <name>
  Data-Objects:
  Data-States: (Label Object-value*)*
```

Figure 5.4

Data-objects specify the data-structures to be used in the abstraction definition. The abstractions are defined in data-state by a label followed by a set of data-objects and value bindings. Each label and its objects define a specific category or data state.

5.2.3 Encapsulation

Experience in artificial intelligence, programming systems, and business organizations has pointed to the need for encapsulating information and mechanism. The merits of such a corpus have been stated both here and in the literature. The primitives necessary in describing encapsulation in organization structures are defined informally.

The main encapsulation construct is the module (from Parnas (1972a)). A module is a combination of control and data structure, acting as a means to a (usually) well-defined end. A module contains three types of descriptions: *data*, *procedure* and *meta* descriptions. Data descriptions define data structures available within and/or outside the module. Procedure descriptions define procedures that are available within and/or outside the module. Meta descriptions provide information about the module that is available to other modules. In the following, only the meta descriptions are discussed. It can be assumed that procedure and data descriptions are of similar form to those in Alphard (Wulf et al, 1977).

Meta descriptions provide information about a module to be used in integrating the module into an organization structure. Essentially they provide the types of information discussed in the previous chapters. Meta descriptions are of three types: 1) A Goal describes the modules "purpose in life" (intent). That is, resources consumed and produced, communication types, how certain it is of achieving the goal, etc. 2) A View is a definition of what other modules can see (view) and use of a module. All messages entering a module (port) are mapped by the View's Mailman to Mailboxes in associated goals. All communication by procedures with other modules are done via messages and mailboxes. 3) A Port defines what channels and modules can connect to the module and the Views they may access. Figure 5.5 depicts the elements of a module's meta description.

```
Module: <module-name>

 Meta:

  (Goal: <goal-name>
    Type:
    Precondition: <data-model>
    Postcondition: <data-model>
    Resource-Consumption:
      (Type, Source, Number, <restrictions>)*
    Resource-Production:
      (Type, Number, <restrictions>)*
    Resource-Transformation:
      (<resource>* --> <resource>*, <certainty-transform-proc>,
           <description>)*
    Initiation: (<message>,<port>)*
    Goal-Model: <model-name>
    Ports: <port-name>*
    Objects: (<procedure-name>*, <data-structure-name>*)
    Data-Models: <data-model-name>*
    Organizations-Membership: (<organization-name>, <role-name>)*
    Utility: <procedure>
    Mailboxes: (<mailbox-name>, <procedure-name>*)*
  )*

(View: <view-name>
   Goal-Views: <goal-name>*
   Data-Views: <data-structure-name>*
   Procedure-Views: <procedure-name>*
   Mailman: (<message>, <goal-name>, <mailbox-name>)*
 )*

 (Port: <port-name>
   Direction: (Input | Output)
   Information-Type: <information-type-name>*
   Connection-Restrictions:
     (<channel-name>*, <module-name>*, <information-Type>*)
   Views: <view-name>*
 )*
```

Figure 5.5

The three meta description types are described in greater detail in the following.

5.2.3.1 Goals

The description of a module's goals is of primary importance to an organization. Its need has been advocated throughout this report. The behavioral analysis stressed knowing a person's goals in order to understand their motivation. While such analyses may not be applicable in single function hierarchical systems, the advent of market-style network-based organizations will require greater attention to the goals of market members.

Goals also serve to summarize the processing of a module so that it can be integrated into an organization model whether for decision analysis or bebugging. They also supercede the notion of a stimulus-response frame in Hearsay-II.

The following defines each of the elements of a goal description:

Type:	Type of the goal.
Precondition:	Data state preceding execution of module.
Postcondition:	Data state following execution of module.
Resource-Consumption:	For each source and type of resource consumed by the module for the particular goal the following is defined:

Type:	The name of the resource.
Number:	The amount of the resource from the source.
Source:	The module that produced the resource.
Restrictions:	Restrictions on how the resource can be used.

Resource-Production:	For each source and type of resource produced by the module for this particular goal, the same descriptions as found in Resource-Consumption are defined.
Resource-Transformation:	Depicts the relationship between resources consumed and produced. An abstraction of resource usage defined in the goal-model. Certainty-transform defines the expected certainty change imparted to resources produced.
Initiation:	Describes the communication and the port in which it arrives that will initiate an instance of the goal.
Goal-Model:	A procedural abstraction of the goal.
Ports:	List of ports that the goal has access to.
Objects:	List of procedures, and data-structures used by the goal.
Data-Models:	Defines the how the data-structures are abstracted.

Organization-Membership:	The set of organizations with whom the goal is shared and the role the module plays in that organization.
Utility:	A function that measures the utility of an instance of the goal.
Mailboxes:	Where messages are deposited in a goal. Specifies the name of the box and the procedure(s) that use the box.

A **module** lies dormant until one of its goals are initiated by an **initiation** message received through a designated **port**. The state of the module before execution and the resources required are defined by the **precondition** and **resource-consumption** attributes respectively. During execution, the module may access only the ports, procedures and data-structures denoted by the **ports** and **objects** attributes. A model of the goal's execution state-transitions and detailed resource usage is defined by the **goal-model**. Once the module finishes executing, i.e., completes the goal, its new state and resources produced are defined by the **post-condition** and **resource-production** attributes respectively. The certainty rating assigned to the new resources is defined by the **certainty-transform** in the **resource-transformation** attribute.

A **module** can be a member of many organizations. The organization and the role the module plays is defined in the **organization-membership** attribute. Whether in one or more organizations, a module may have many goals instantiated. The decision as to which goal to work on is defined by the utility attribute.

5.2.3.2 View

The purpose of a **View** is to restrict the data, procedure, and meta description that another module may access. A channel, outputting from module A to module B, is restricted to the view specified at module A's port. Hence Module B is restricted to a view of module A specified by module A. A **View** essentially specifies the maximum capabilities at a port. The following defines the **View** meta description.

Goal-Window:	Specifies parts of each goal that can be viewed from the communication port.
Data-View:	Specifies the data structures that can be viewed and the access rights.
Procedure-Views:	Specifies the procedures that can be viewed from the port and the access rights.
Mailman:	Describes to what mailboxes in the viewed goals the messages are routed to. For each message type, the mailman delivers it to a particular goal.mailbox.

5.2.3.3 Ports

A **Port** defines the only way one module can access another. It is the central module construct because it defines the language of communication, the **View** that a connecting module has, and, in turn, the goals that can be seen via the **View**. It also places restrictions on who can connect via a **channel** to the **port**.

For each communication channel joining modules, there exists a port to which the channel is attached. The port describes the following characteristics:

Direction:	The port is for **input** or **output** only.
Information-Type:	The information-type that may enter or leave the port.
Connection-Restrictions:	Specifies the type of channels, modules, and organizations that can connect to the port.
Views:	The view(s) that a connecting module may access via this port.

If the **direction** of **module** A's port is **input**, the **view** defines what views of **module** A that **module** B can access (when it sends messages). If the **direction** of **module** A's port is **output**, then B is receiving information from module A. **View**$_A$ defines what information in module A can be accessed by module B.

5.2.4 Data-Modules

Blackboards, file systems, and stacks are structures for storing information. Their secondary function is providing mechanisms to access and manipulate data. A **data-module** is the design primitive that encompasses the above. It is similar to abstract data types, and the data modules in ASTRA76 (Schwald, 1977). The definition of a data-module requires the definition of objects and a way of structuring them. (Assume that a language like Alphard, CLU, or PASCAL is available at the control structure level to define objects.) Next, access and modification procedures must be defined with an appropriate protection mechanism (e.g., capability, or access list).

```
Data-Module: <data-module-name>
  ( Region: <region-name>
      Type:
      Attributes:
      Units: <information-type-name>*
      Structure: (<relation-name> <object-name>*)*
      Access-Procedures:
        (<procedure-name>, <object-name>, <access-Type>)*
      Sub-Regions: <region-name>*
      Sub-Region-Structure:
        (<relation-name> <region-name>*)*
      Information-Models:
      Mailbox:
 )*
 (View: <view-name>
   Region: <region-name>
   Restrictions:
   Mailman: (<message>, <region-name>, <mailbox-name>)*
 )*

 (Port: <port-name>
   Direction: (Input | Output)
   Information-Type:
   Connection-Restrictions:
   Views: <view-name>*
 )*
```

Figure 5.6

Figure 5.6 shows the contents of a data-module description. The following describes each primitive:

Region:	A categorization of data units. Similar to levels of representation in Hearsay-II.		
	Type:	Region is an instantiation of a region type.	
	Attributes:	Arbitrary list of attrbutes to describe the region.	
	Units:	Set of legal information-types at this level.	
	Structure:	Set of allowable N-ary relations between objects in this or other regions.	
	Access-Procedures:	Defines the procedures that may operate on objects defined at this level.	

		Name:	Name of the procedure being accessed.
		Unit:	Name of units that function is applicable to.
		Access-Type:	The type of accesses the function can make on the unit, e.g., read-only.
	Sub-Regions:	Regions within the region. Defines a further partitioning of the region. Can overlap.	
	Sub-Region-Structure:	Define the relations between regions. Such as next-to, Greater-than, etc. Can also include a numeric ordering as the time line in the phrasal level in Hearsay-II.	
	Information-Models:	Provide abstractions of the state of objects within the region. Definitions are composed of arbitrary relations on the region's objects.	
	Mailboxes:	Place where messages are deposited by the mailman. Is composed of a box name and the procedures that may use it.	

Modules connect via a channel to a data-module communication port. The description of a port is the same as in a module.

As in a module, a data-module can restrict a view that a module can have of the data. A View defines the views of a data-module. Figure 5.6 shows the contents of a View. The definitions of the parts of a View are:

Region:	The region to be viewed.
Restrictions:	restriction on the symbols and dimensions of the regions under view.
Mailman:	Mechanism for distributing messages to mailboxes for each region in the view. Specifies the message pattern, region, and mailbox.

A data-module contains a variety of information at possibly many levels of abstraction. All modules do not want and should not have access to all information. Although it can be handled by the protection mechanisms, we prefer to define Views of the data-module. A View can be thought of being similar to a View of a module. In this case a View defines a set of access and protection mechanisms plus regions of the data-module available for perusal by a module posessing that View.

Modules should not continually monitor a data-module for the appearance of relevant information. Procedures associated with regions have the ability to monitor the region for changes. A module interested in a particular change type, e.g., a new phrase, sends a message to the appropriate mailbox describing its monitor expression. When the specified change has occured, the module is informed by the data-module.

5.3 Structuring

The act of structuring an organization constitutes the specification of relations between conceptual units: modules and data-modules. The primary structuring primitive is the channel. A channel specifies a communication path between two modules.

```
Channel: <name>
  Source-Module: <module-name>
  Channel-Source-Information-Packet: <information-packet>
  Sink-Module: <module-name>
  Channel-Sink-Information-packet: <information-packet>
  Translation-Function: <procedure>
```

Figure 5.7

A channel allows information to travel between two modules. Information flows in one direction, from source to sink. In the channel definition the modules can be designated along with the information-packet types sent and accepted. Information consistency is maintained through the sharing of lexicon and language. If the information-type at each end are not consistent then a translation-function may be specified. It translates one information-type to another.

```
Organization: <organization-name>

  (Goal: <goal-name>
    Type:
    Resource-Production:
    Resource-Consumption:
    Resource-Transformation:
    Initiation: <message>*
    Goal-Model: <model>
    Objects: (<role> | <relation>)*
    Ports: <port-name>
    Super-organizations: (<organization-name>, <role-name>)*
    Utility: <procedure>
    Certainty-Transform: <procedure>

    Structure: <name>
      Roles: (<organization-type> | <module-type>)*
      Communication-Relations:
        (<role-name>, <channel-name>, <role-name>)*
    Authority-Relations:
        (<role-name>, <authority-Type>, <role-name>)*
  )*

  (View: <view-name>
    Goals-views:
      (<goal-name>,(<goal-attribute>, <restriction>)*)*
    Modules-views: <module-name>*
    Data-Modules-views: <data-Module-name>*
    Mailsorter: (<message>, <goal-name>, <module-name>,
      <port-name>)*
  )*

 ( Port: <port-name>
    Direction: (Input | Output)
    Information-Types:
    Connection-Restrictions:
    Views:
```

Figure 5.8

Multi-module systems are defined using the organization construct. An organization defines a structure by specifying partial descriptions of modules, data-modules, and the channels linking them.

An organization declaration does not create modules, data-modules, and channels but acts as a type description. Any pre-defined module or data-module inserted into a organization must satisfy the type contained in the organization definition.

An organization is composed of a meta-description, and a set of modules, channels, and data-modules. The meta-description of an organization is similar to that of a module. It provides information about the organization's intent, structure, and views. As in a module, an organization can have multiple goal descriptions. Each goal specifies:

Type:	The goal type this is an instance of.
Resource-Consumption:	What resources are consumed during the fulfillment of the goal.
Resource-Production:	What resources are produced with the completion of the goal.
Resource-Transformation:	Depicts relation between a resouce consumed and one or more produced. Allows domain dependent descriptions to be associated with it.
Initiation:	The message that instantiates the goal.
Goal-Model:	An event based model of the processing done to fulfill the goal.
Objects:	Roles and relations used by goal.
Super-Groups:	The organizations that this goal is a part of.
Utility:	A function that measures the utitlity of executing the goal.
Certainty-Transform:	A function that measures the certainty of the resources produced as a function of the resources consumed.

It should be noted that the goal-model for an organization is event based. Modules and organizations are parallel and asynchronous hence the model should reflect this.

The second part of an organization's goal-descripton is its structure. The role defines each position (unit) in the organization. Roles are organized by the communication and authority structure of the organization. Communication-Relations describe the channels joining roles. Authority-Relations define what authority-types exist between roles. At this time it is unclear what should be put into the authority-type. One approach is to specify a channel and a set of stimuli-action pairs describing commands and their prescribed actions.

An organization has multiple views. They are similar to a module's view in that they limit what external modules and organizations can see. A view specifies what modules, data-modules, channels, structure, and goals can be seen. Message handling is carried out by the mailsorter. It differs from a module's mailman in that it sends copies of a message to multiple modules. Each module's mailman then distributes the message to their

goals.

Finally, an organization has multiple ports. Again an organization port is similar to a module port. The port allows external organizations to attach channels to the current one.

The organization approach to system design permits a recursive approach to organization specification. The role in an organization can be filled by a module or an organization. The similarity of goal, view and port definitions support this substitution.

5.3.1 Loose Ends

Modules, Data-modules, and Structures have been defined. No attempt has been made to make their definitions rigorous or complete. But a few words more about loose ends should be added.

First, the elements of a meta description are defined in the context of larger entities, i.e., modules, data-module, and structure. These descriptions can be defined separate from the larger entities, and be used in them.

Second, protection mechanisms have not been discussed. It will be necessary to define protection mechanisms for the viewing of information, and control of resources[22]. This allows the protection of resources and enforces authority relations.

Last, the dynamic aspect of creating and removing channels, data-modules, modules, and organizations has been hinted at through-out the paper. Run-time mechanisms should be made available for the definition and instantiation of these design constructs.

5.4 Rationalizing the Language Constructs

Most of the primary concepts (e.g., module, channels, ports) described above can be found in current languages (e.g., mesa, modula). The organization structuring language differs in two respects. First, the concept of multiple goals for a single module introduces notions of motivation and behavior. Second, many attributes have been added to the concepts of a module, data-module, channel, and information. Attributes such as certainty, authority, and utility were added to permit the type of organizational analyses described in the previous chapters.

[22]Modules can be viewed as a resource.

In this section the major concepts introduced in the previous chapters are examined briefly. For each concept, the language constructs that relate to it are noted.

5.4.1 Organization Structure Continuum

Organization theory was presented as an analysis of why there exists a continuum from hierarchy to heterarchy of organizations. It was shown that domain dependent characteristics, e.g., uncertainty, complexity, and behavior affected a shift in either direction. The first test of the language is whether it can represent the structures in the continuum. Any organization in the continuum can be described with the primitives provided. The primary method of defining a structure is the organization. Each position in the organization is defined by a role, and the relationship between each role, i.e., control and information, by the structure primitives. Whether the organization is hierarchical or heterarchical depends on the structural relationships defined. Like human organizations, roles have goals and the people who fill them have goals. Accordingly, modules and organizations have multiple goals plus a utility function to order them.

To represent a heterarchical problem-solving organization a central data-module must be equally accessable by all modules. Information types are mutually shared and one module (the organization leader) exercises control over the others. The roles of the modules can be defined in domain-independent problem-solving terms.

A hierarchical labor-division organization is hierarchical with each module controlling its immediate sub-ordinates. Each module could be replaced by another organization definition.

A professional-society organization has a large number of modules communicating information in one direction. There exists a root organization with channels distributing information to each of the members.

5.4.2 Complexity Reduction

The main thrust of complexity reduction is decomposition. Bounded rationality requires the decomposition and abstraction of information and control. To enable the decomposition analysis the language provides Views. A View specifies only the information another module may see. Combined with the specification of a port and channel, the language details exactly the information and control that passes between module. To explicate how modules act and relate to other modules, Goal-Models are provided. A model provides an abstraction of a module's actions including resource consumption, exception conditions, and the probability of a particular action (state) occuring. This enables the description of when

modules must communicate to solve exceptional conditions. The Initiation construct specifes the relationship between a command and its procedural elaboration.

The data declarations of a module also provide data-models. The data-model specifies the abstractions of a module's data allowing an analysis of how information is reduced (or absorbed) throughout the organization.

The Price System approach to complexity reduction can be represented using a goal's resource-production. A module provides a view of its contractor goal to the market. Contractee's can then communicate its needs via a channel to the goal's port using a contract language specified in the port and channel information-packet.

5.4.3 Uncertainty Reduction

Uncertainty reduction in the market place requires increased monitoring capabilities. Views, Ports, and channels provide the mechanisms to specify monitoring structures. The information monitored can be restricted by the view a module is given. The credit bureau approach to monitoring can be implemented by sending certain information-types to a data-module. The data-module views can, in turn, limit the data seen by subscribers.

Function vs. product decomposition as means to reducing uncertainty can be analysed as described in the previous section. Lateral relations can be devised by specifying a sharable information-type, and ports and channels with which to communicate. The information-type is restricted to symbols dealing with specific problems.

Vertical integration is an approach to reducing uncertainty by increasing an organization's control over services, and products. Market contracts are replaced by employment relations. All the structures in the language provide the means of creating any organization structure in the continuum. The organization's structure definition provides primitives for describing roles in an organization and the communication and authority relations between roles.

5.4.4 Behavior Uncertainty

Markets (heterarchies) and data-driven systems require a module to decide when to react to environmental changes (new data states). This open's the issue of a module's motivation. To represent the conditions under which a module may react, the goal primitive is provided. It defines what resources it consumes and produces and conditions under which it is initiated. A module can have multiple goals. The decision of which goal to persue when more than one is enabled is based on the utility primitive. Utility defines the factors and their importance in rating alternative goal instantiations.

Conflict resolution among conflicting goals is enabled through the declarative representation of a module's goals and the views provided to other modules. Bargaining can take place if the proper views, ports and communication channels are provided among the conflicting modules and module with authority to resolve the dispute.

Similarily, identification of information can be described through the proper creation of information-types. Its categorization is described by the data-model.

To support program elaboration, the variables under control, environmental variables, and problem attributes must be known. The view provide the relevant data for a module's model, data-structures, procedures and goals.

5.4.5 Information Uncertainty

In the chapter on organization theory, a taxonomy of information uncertainties was presented. Structures for reducing uncertainties were also described. Here the uncertainties and the corresponding language structures are briefly described.

Consumer uncertainty needed a mechanism of informing other modules of new information. Data-modules allow the representation of message boards, and complex channel structures enable broadcasting and message passing. Restricting communication to the same information-type reduces language problems.

Producer uncertainty is reduced through module tracking and monitoring. By tagging information-packets with the producer, footprints are left in the system. Secondly, experimentation with modules to see if they work properly is enabled by providing of test-data for each procedure, goal, and organization. To analyse the logic of an organization, abstractions are provided by models.

Functional uncertainty is reduced through resource consumption monitoring. Properly defined views allow resource monitoring.

Result uncertainty is reduced by a goal's definition of what communication initiates what action (procedure), and by resource consumption and production information.

Information veracity is not reduced by structural means. But data-models can be used to define how data uncertainty is reduced by synthesis.

Information semantics uncertainty can be reduced by a more careful defintion of communication languages. This is provided by the information-types. How information is interpreted is defined by the initiation primitive of a goal and the mailsorter and

mailman primitives. This is a procedural definition in th sense that it defines how a module reacts to communication.

5.4.6 Algorithm Uncertainty

Reducing uncertainty in algorithms was divided into three parts: information gathering, program construction, and system evaluation and repair. To enable information gathering channels, ports, and views must be provided to the information of interest. Where the information originated and why can be found on the information-packet attributes.

Program construction requires reconfiguring sequences of actions. A module's goals give the program elaborator knowledge of what the module is capable of doing via the procedure and goal models. Hence, the program elaborator can pick and choose among modules and construct a sequence of commands that result in correct behavior. Also, reconfiguring requires that modules have the proper authority to effect change. This is provided in the organization structure authority descriptions.

System evaluation and repair require essentially the monitoring and authority relations mentioned above.

5.4.7 Team Decision Theory

To carry out the analysis of alternative organization structures, the decision theory approach required four types of information: information categories, costs, certainty transforms, and path probabilities. The information categories are used to decompose productions into a network. Categories are defined in the language by information-types. The types produced or consumed are described by a goal's resource-consumption and production primitives. Cost information is also provided in the resource declarations. How a goal transforms certainty is defined by the certainty transform in the resource-transformation primitive. It is defined on the resources specified to the goal. Path probabilities are defined both for state-transitions in a procedure model, goal and organization models. Hence, transition probabilities can be modeled at many levels of abstraction.

5.5 An Example: Hearsay-II

This section illustrates how the organization structuring language is used to represent a portion of the Hearsay-II speech understanding system (see section 2.2). In this example, only two modules are described: SASS, the syntax and semantics knowledge source, and the phrase and word levels of the BLACKBOARD.

```
Organization: Hearsay-II
  Goal: Understanding-Speech
    Resource-Consumption: (Speech-Wave, Organization.Hs2ear, 1)
      (seconds, time, approx(100)) (core, space, approx(100))
    Resource-Production: (Utterance, 1)
      (core, space, approx(100))
    Initiation: ("monitor" <parms>)
    Goal-Model: HSII-Model
    Objects: (Island-Builder, Word-Hypothesizer, Word-Verifier,
      Word-Sequence, New-Phrase-Chan)
    Ports: (Hs2EAR-Port, Semant-Port)
    Super-Organizations: NIL
    Utility: 1
    Structure:
      Roles: (Word-Hypothesizer, Word-Verifier, Island-Builder,
        Segmenter, Word-Sequencer, Blackboard)
      Communication-Relations:
        (Island-Builder New-Phrase-Chan Blackboard)
        (Word-Hypothesizer New-word-Chan Blackboard)
        ...


  View: Hs2ear-View
    Goal-views: Hearsay-II
    Data-Module-Views: Blackboard
    Mailsorter: ("Parameters" <parms>, Understanding Speech,
      Blackboard, Parameter-Port)

  Port: HS2EAR-Port
    Direction: Input
    Information-Type: Parameters
    Connection-Restriction: Organization.Hs2ear
    View: Hs2Ear-View

  Port: Semant-Port
    Direction: Output
    Information-Type: Utterance
    Connection-Restrictions: Organization.Semant
```

To start, the description of the Hearsay-II organization is defined. Structurally, it is composed of many roles. Two roles instantiated in this example are the blackboard and island-builder. How each role instantiation is constrained, is defined by the

organization[23]. The only constraints placed on them in the example are communication-relations. There is a new-phrase-chan between the island-builder and the blackboard.

The Hearsay-II organization has two ports. The Hs2ear-port receives parameterized samples of the speech input from the Hs2ear organization. The Hearsay-II output, a recognized utterance, is sent via the semant-port to the semant organization. When information enters the Hs2ear-port it is distributed by the Hs2ear-view's mailsorter to the blackboard's parameter-port.

Hearsay-II's primary goal is understanding-speech. It consumes the digitized speech signal and produces an utterance. It also consumes time and space resources which are expressed as expected values. The goal specification for the Hearsay-II organization includes many roles including an island-builder and blackboard.

```
Module: Sass
  Goal: build-island-goal
    Resource-Consumption: (phrase, blackboard, 1, (no-delete))
            (word, blackboard, max(2))
            (core, space, 35K)
            (seconds, time, 5)
    Resource-Production:  (phrase, 1, (no-delete))
    Resource-Transformation:
      (<word1> <phrase1> <word2> --> <phrase2>,
       Certainty-Average,
       time(<phrase2) > time(<phrase1>)
            (core, 35K))
    Initiation: (Extend-box)
    Goal-Model: Build-island-model
    Ports: (new-phrase-port, monitor-phrase-port)
    Objects: (extend, parse, find-extensions, concat,
               Certainty-Average)
    Group-membership: (Hearsay-II.Island-Builder)
    Mailbox: (extend-box, extend)

  View: island-builder-view
    Goal-View: build-island-goal
    Mailman: ("New" <phrase>, build-island-goal, extend-box)
```

[23]Note that we differentiate between role description (presecription) and the actual module that fills the role.

```
Port: New-phrase-port
  Direction: Input
  Information-Type: New-Phrase
  Connection-Restriction: (Data-Module: Blackboard)
  View: island-builder-view

Port: monitor-phrase-port
  Direction: Output
  Information-Type: Monitor-phrase
  Connection-restriction: (Data-module: Blackboard)
  View: new-phrase-view
```

The definition of the SASS module has two ports. The New-Phrase-Port specifies that New-Phrase information enters this port only. The connecting module is restricted to the blackboard and the view it may access is the island-builder-view. The Monitor-Phrase-Port is for outputting monitor expressions to the Phrasal Region of the Blackboard. Its view specifies that its is only interested in the New-Phrase-View. The Island-Builder-View restricts the New-Phrase-Port to accessing only the Build-Island-Goal of SASS. Procedures and data-structures in SASS are not accessable via this view. When a message enters the new-phrase-port it is distributed to all the views accessable by the port. Each view's mailman distributes the messages to the mailboxes of each goal that can be viewed. In the island-builder-view a message that begins with "new" followed by a <phrase> is deposited in the build-island-goal's extend-box.

The SASS module has many goals, only one is depicted. It is the build-island-goal. That is, it takes a new phrase and trys to extend it on both sides by adding hypothesized words which result in a new grammatically correct phrase. The net effect is described in the resource-consumption and production primitives. A phrase, some words, time and space are consumed. The old phrase, a new phrase, the words and the space are returned. The primary resource transformation of interest is the transformation of two words and a phrase into a new phrase. In the description portion it is stated that the new phrase is of greater time span than the old phrase and the certainty imparted is the average of the resources consumed. Because of the domain dependency of resource usage, a transformation description language is not defined. It is left to the system builder to define and use in the description field. To initiate the build-island-goal a message is sent ot the extend-box mailbox. Once the goal is initiated, it has access to the ports, procedures, and data-structures listed in the objects slot. How the module's goal fits in within the Hearsay-II organization is specified by the group-membership. SASS's

build-island-goal fills the role of island-builder in Hearsay-II.

```
Model: build-island-model
  State-Labels: (wait-new-phrase, concat, parse, find-extension)
  Abstraction:
    (Begin -> wait-new-phase, NIL, NIL, 100%, N/A)
    (Wait-new-phrase -> parse, phrase, phrase, 100%, 1)
    (parse -> find-extensions, phrase, (phrase, word*), 100%, 1)
    (find-extensions -> concat, (phrase, word), phrase, 50%,
      (wordlen(phrase)*cert(phrase) + cert(word)/
              (wordlen(phrase)+1))
    (find-extensions -> wait-new-phrase, phrase,
     (phrase, word-goal), 50%, 0)
    (concat -> wait-new-phrase, NIL, NIL, 100%, N/A)
```

Included in the goal description is a model of how the goal is accomplished: Build-island-model. It is anchored to the goal by labels in the procedures (not shown). The state-transitions (from label to label) show that the initial state is begin. Upon start up, the goal goes into a wait state for instantiation. Once instantiated by a new-phrase from the blackboard it parses it, then looks for extensions (words) on the blackboard. If it finds them they are concatenated and the wait for a new-phrase state is entered. Obviously this is a simplified model. Alot of what was described above is not in the model. To represent all that goes on requires a rich description language to describe the model's behavior.

```
Data-Module: Blackboard

  Region: Lexical
    Units: word
    Structure: (Sequence-link Syllable,Syllable* word)
    Access-Procedures:
      (Add, write)
      (Find, read)
      (Link, write)
    Sub-Regions: (Time1, Time2, Time3)
    Sub-Region-Structure: (Rational-Sequence, Time1 Time2 Time3)

  Region: Phrasal
    Units: phrase
    Structure: (Sequence-Link Lexical,word* phrase)
          (Sequence-Link phrase* phrase)
    Access-Procedures:
      (Monitor-New-Phrase, phrase read)
      (Add, write)
      (Find, read)
      (Link, write)
    Sub-Regions: lexical
    Mailbox: (new-phrase-box, monitor-new-phrase)


  View: Monitor-New-Phrase-view
    Region: Phrasal
    Restriction: (Procedure: monitor-new-phrase)
    Mailman: ("Monitor" <monitor-type> <unit-type>, phrasal,
      new-phrase-box)

  Port: New-Phrase-Port
    Direction: Output
    Connection-Restriction: (Module Sass)
    Views: island-builder-view
    Information-Type: New-Phrase

  Port: monitor-new-phrase-port
    Direction: Input
    Communication-Restrictions: (Module: SASS)
    Views: monitor-new-phrase-view
    Information-Type: Monitor-Phrase
```

The Hearsay-II blackboard is represented by a data-module. Two regions are depicted.

The lexical region is composed of word units. Each word is connected by a sequence link to syllables in the syllable region. The lexical region is divided into sub-regions which form a sequence (time sequence). To access the region the add, find, and link procedures are provided. They are used by communicating with their mailboxes. The phrasal region is similar to the lexical region. The mailbox for the monitor message from SASS is shown as having the name new-phrase-box and being associated with the monitor-new-phrase procedure. A port and view are shown for providing monitor messages.

```
    Information-Type: new-phrase
          Lexicon: {new} UNION <phrase>
          Language: "new" <phrase>

    Information-Type: monitor-phrase
          Lexicon: {monitor} UNION <monitor-type>
            UNION <phrase>
          Language: "monitor" <monitor-type> <phrase>

    Information-Packet: Sass-Pac
          Information-Type: new-phrase
          Producer: Blackboard
          Consumer: Sass

    Information-Packet: monitor-phrase-pac
          Information-Type: monitor-phrase
          Producer: SASS
          Consumer: Blackboard

    Channel: new-phrase-chan
          Source-module: Blackboard
          Channel-source-information-packet: sass-pac
          Sink: SASS
          Channel-sink-information-packet: sass-pac

    Channel: monitor-phrase-chan
          Source-module: SASS
          Channel-source-information-packet: monitor-phrase-pac
          Sink-module: Blackboard
          Channel-sink-information-packet: monitor-phrase-pac
```

Last, the information types that are passed between the two modules are described. New-phrase defines the message composed of the word "new" and a <phrase>. It is the

Monitor-phrase is the information-type for specifying a monitor expression to the blackboard. The blackboard will monitor the appropriate region for the occurence of the pattern and inform SASS. Corresponding to each information-type is a packet definition. The new-phrase-pac sends new-phrase information from the blackboard to SASS. Monitor-phrase-pac sends it in the opposite direction. The two channel definitions transport each of the packet types.

This example gives a flavor of how the organization structuring language is to be used. The example is deficient in many ways. Roles could be better defined so that the role filler duties (goals and models) are prescribed. Secondly, data-abstraction and exceptional conditions in models were not defined though their importance has been stated often. Finally, a model for the Hearsay-II organization was not defined. There are two approaches to constructing one. The first is to construct one by hand describing how the modules intereact, resource usage etc. An alternate method is to "automatically" construct on from the role definition, the models of the modules that fill then, and a single control assumption: processing is data-driven.

5.6 Summary

A language for describing organization structures was defined and its relation to the concepts of the previous chapters described. A portion of the Hearsay-II speech understanding system was represented as an example of its usage. The primary contribution of this language is the representation of goals. A module or organization may have many goals and the processing done is distinct for each. An important part of goal representation is the description of a goal's resource usage and state change. The resource declarations combined with the goal's model provide other modules with a behavioral description of a module or organization. Another attribute of the language is its capability to represent certainty, utility and transitory state behavior both at the module and organization level. The representation of this information is needed for the analyses of complexity, uncertainty, and behavior. Finally, the definitions of ports, views and goals were kept similar for both modules and organizations to allow the abstraction of behavior from structure. That is a module and organization look the same when externally accessed via a port and view.

6. References

Alexander C., (1965), Notes on the Synthesis of Form, Cambridge: Harvard University Press.

Arrow K., (1974), The Limits of Organization, New York: Norton and Co.

Bobrow D., and D. Norman, (1974), Resource Limited Processing, XEROX Palo Alto Research Center, Tech. Rep., Palo Alto CA.

Bobrow D., and Norman, (1975), Some principles of memory schemata, XEROX Palo Alto Research Center, Tech. Rep., Palo Alto CA.

Bonnen J., (1973). Hierarchical Control Programs in Biological Development, In Hierarchy Theory: The Challenge of Complex Systems, H.H. Pattee (Ed.). New York: George Braziller Inc.

Chanon P.N., (1974), On a Measure of Program Structure, (Ph.D. Thesis), Computer Science Dept., Carnegie-Mellon University, Pittsburgh PA.

Courtois, (1977), Decomposability, Academic Press.

Dahl O.J. and C.A.R. Hoare, (1972), Hierarchical Program Structures, In Structured Programming, O. Dahal, E. Dijkstra, and C. Hoare, New York: Academic Press.

Dahl R., and C. Lindblom, (1953), Politics Economics and Welfare, New York: Harper and Brothers.

Davis R., (1976), Applications of Meta-Level Knowledge to the Construction and Maintenance, and Use of Large Knowledge Bases, (Ph.D. Thesis), Stanford University, STAN-CS-76-552, Stanford CA.

DeReemer F., and H.H. Kron, (1976), "Programming in the Large vs Programming in the Small", IEEE Trans. on Software Engineering, Vol. 2, No. 2, pp. 80-86.

DSN, (1978), Distributed Sensor Nets, Workshop Proceedings, Computer Science Dept., Carnegie-Mellon University, Pittsburgh PA, Dec. 1978.

Duda R.O., P.E. Hart, and N.J. Nilsson, (1976), Subjective Bayesian methods for rule-based inference systems. Proc. of NCC.

Duda R.O., P.E. Hart, P. Barrett, J.G. Gaschnig, K. Konolige, R. Reboh, and J. Slocum, (1978), "Development of the Prospector Consultation System for Mineral Exploration: Final Report", Tech. Rep., SRI International, Menlo Park CA, Oct. 1978.

Engelmore R.S. and H.P. Nii, (1977), A Knowledge-Based System for the Interpretation of Protein X-Ray Crystallographic Data, Stanford University, HPP Report HPP-77-2, Stanford CA.

Erman L.D., (1975), Overview of the HEARSAY Speech Understanding Research, Carnegie-Mellon University Computer Science Dept. Review, Pittsburgh PA.

Erman L.D., (1977), A Functional Description of the HEARSAY-II System, Proceedings 1977

IEEE Conference on ASSP, Hartford, CT, May 1977.

Erman L.D., and V.R. Lesser, (1979), "The Hearsay-II System: A Tutorial", W.A. Lea (ed.), Trends in Speech Recognition, Englewood Cliffs, NJ: Prentice-Hall.

Feldman J.A., and Y. Yakimovsky, (1974), Decision Theory and Artificial Intelligence: A Semantic Based Region Analyser, Artificial Intelligence, Vol. 5, P. 349-371.

Fikes R.E. and N.S. Nilsson, (1971), Strips: A New Approach to the Application of Theorem Proving to Problem Solving, Second International Joint Conference on Artificial Intelligence, London: British Computer Society.

Fox M.S., (1978), Knowledge Structuring: An Overview, Proc. of the 2nd Conf. of the Canadian Society for Computational Studies of Intelligence, Toronto, Ontario.

Fox M.S. and D.J. Mostow, (1977), Maximal Consistent Interpretations of Errorful Data In Hierarchically Modelled Domains, Fifth International Joint Conference on Artificial Intelligence, Cambridge MA, 1977.

Fuller S.H., and S.P. Harbison, (1978), The C.mmp Multiprocessor, Technical Report, Computer Science Dept., Carnegie-Mellon University, Pittsburgh PA.

Galbraith Jay, (1973), Designing Complex Organizations, Addison-Wesley.

Goldberg H.G., (1975), Segmentation and Labelling of Speech: A Comparative Performance Evaluation, (Ph.D. Thesis), Tech. Rep., Computer Science Dept., Carnegie-Mellon University, Pittsburgh PA, Dec. 1975.

Habermann A.N., L. Flon, and L. Cooprider, (1976), Modularization and Hierarchy in a Family of Operating Systems, CACM, May 76, Vol. 19, No. 5.

Hayes-Roth F., (1977), The role of partial and best matches in knowledge systems, In Waterman and Hayes-Roth (Eds.), Pattern directed inference systems. N.Y.: Academic press.

Hayes-Roth F., and V.R. Lesser, (1976), Focus of attention in a distributed logic speech understanding system. Proceedings of the 1976 I.E.E.E. International Conference on Acoustics, Speech and Signal Processing, Philadelphia, 1976, 416-420.

Hayes-Roth F., L.D. Erman, M.S. Fox and D.J. Mostow, (1977a), Syntactic processing in Hearsay-II. In Speech Understanding Systems: Summary of Results of the Five-Year Research Effort, Department of Computer Science, Carnegie-Mellon University, Pittsburgh PA, 1977.

Hayes-Roth F., M.S. Fox, G. Gill, and D.J. Mostow, (1977b), Semantics and pragmatics in the Hearsay-II speech understanding system. In Speech Understanding Systems: Summary of Results of the Five-Year Research Effort, Department of Computer Science, Carnegie-Mellon University, Pittsburgh PA, 1977.

Hayes-Roth F., G. Gill, and D.J. Mostow, (1977c), Discourse analysis and task performance in the Hearsay-II speech understanding system. In Speech Understanding Systems: Summary of Results of the Five-Year Research Effort, Department of Computer Science, Carnegie-Mellon University, Pittsburgh

PA, 1977.

Hayes-Roth F., D.J. Mostow, and M.S. Fox, (1978), Understanding Speech In the Hearsay-II System, In Speech Communication with Computers, L. Bolc (Ed.), Berlin: Springer-Verlag.

Hendrix G.G., (1975), Expanding the utility of semantic networks through partitioning. Fourth International Joint Conference on Artificial Intelligence, Tiblisi, USSR.

Hewitt C., (1971), Procedural Embedding of Knowledge of Planner, International Joint Conference on Artificial Intelligence, London: British Computer Society.

Hewitt C., (1973), The ACTOR Formalism. Third International Joint Conference on Artificial Intelligence, Stanford, CA.

Holt R., (1972), Some Deadlock Properties of Computer Systems, Computing Surveys, 4,3, pp179-196.

Horning J.J. and B. Randell, (1973), Process Structuring, Computing Surveys, 1973.

Jones A.K., (1973), Protection in Programmed Systems, (Ph.D. Thesis), Computer Science Dept., Carnegie-Mellon University, Pittsburgh PA.

Kassouf S., (1970), Normative Decision Making, New Jersey: Prentice Hall.

Knuth D., (1974), Computer Programming as an Art. CACM, Vol. 17, No. 12, Dec. 74.

Lenat D., (1975), Beings: Knowledge as Interacting Experts, Fourth Int. Joint Conf. on Artificial Intelligence, Tiblisi, USSR.

Lesser V.R., and D.D. Corkhill, (1978), "Cooperative Distributed Problem-Solving: A New Approach for Structuring Distributed Systems", Technical Report, Dept. of Computer and Information Science, U. of Mass. at Amherst.

Lesser V.R., and L.D. Erman, (1977), A retrospective view of the HEARSAY-II Architecture, International Joint Conference on Artificial Intelligence, Cambridge, MA, Aug. 1977.

Lesser V., F Hayes-Roth, M. Birnbaum, and R. Cronk, (1977), "Selection of Word Islands in the Hearsay-II Speech Understanding System", Proceedings of the 1977 IEEE Conference on ASSP, pp791-194.

Liskov B., A. Snyder, R. Atkinson, and C. Schaffert, (1977), Abstraction Mechanism in CLU, CACM, Aug. 77, Vol. 20, No. 8.

Lowerre B., (1976), The HARPY Speech Recognition System, (Ph.D. Thesis), Tech. Rep., Computer Science Dept., Carnegie-Mellon University, Pittsburgh PA.

Lyons J., (1968), Introduction to Theoretical Linguistics.

March J.G., and H.A. Simon, with H. Guetzkow, (1958), Organizations, New York: John Wiley and Sons.

Marschak J. and R. Radner, (1972), Economic Theory of Teams, Yale University Press.

McClure C.L., (1978), A Model for Program Complexity Analysis, Proc. of the 3rd Int. Conf. on

Software Engineering, MA.

Mckeown D., (1977), Word Verification in the HEARSAY-II Speech Understanding System, Proceedings 1977 IEEE International Conference on ASSP, Hartford CT, May 1977.

Merton R.K., (1940), Bureaucratic Structure and Personality, Social Forces, 1940, 18, 560-568.

Mitchell J.G., W. Maybury, and R. Sweet, (1978), "Mesa Language Manual", Technical Report, Xerox Parc, Palo Alto CA, Feb. 1978.

Newell, A., (1969), Heuristic programming: ill-structured problems. In J. Aronofsky (Ed.), Progress in Operations Research 3. New York: Wiley, 1969, 360-414.

Newell A., (1973a), Artificial Intelligence and the concept of mind. In R. Schank and K. Colby (Eds.), Computer Models of Thought and Language. San Francisco: Freeman Press, 1973, 1-60.

Newell A., (1973b), Production Systems: models of control structures. In W.C. Chase (Ed.), Visual Information Processing. New York: Academic Press, 1973, 463-526.

Newell A., D. McCraken, and G. Robertson, (1977), L*:An Interactive Symbolic Implementation System, Tech. Rep., Computer Science Dept. Carnegie-Mellon University, Pittsburgh PA, October, 1977.

Newell A., and H.A. Simon, (1963), GPS: A Program that Simulates Human Thought, In Computers and Thought, E. Feigenbaum and J. Feldman (Eds.), New York: McGraw-Hill Co.

Newell A., and H.A. Simon, (1972), Human Problem Solving, Englewood Cliffs, N.J.: Prentice Hall.

Nii P., and E. Feigenbaum, (1977), Rule based understanding of signals, In Pattern-Directed Inference Systems, D. Waterman and F. Hayes-Roth (Eds.), New York: Academic Press.

Oleinick P., and S.H. Fuller, (1978), The Implementation and Evaluation of a Parallel Algorithm on C.mmp, Technical Report, Computer Science Dept., Carnegie-Mellon University, Pittsburgh PA.

Parnas D.L., (1972a), On the Criterion to be Used In Decomposing Systems Into Modules, CACM, Vol. 15, No. 12, 1053-1058.

Parnas D.L., (1972b), On the Problem of Producing Well Structured Programs. Computer Sciences Research Review, Carnegie-Mellon University. Pittsburgh, PA. pp 27-36.

Parnas D.L., (1974), On a 'Buzzword': Hierarchical Structure, IFIP 74, J.L. Rosenfeld (Ed.), New York, N.Y.:North Holland.

Pople H., (1977), The Formation of Composite Hypotheses in Diagnostic Problem Solving: An Exercise in Synthetic Reasoning. Proceedings of the Fifth International Joint Conference on Artificial Intelligence, Cambridge, Aug. 77.

Rapoport A., (1966), 2-Person Game Theory, Ann Arbor, Michigan: University of Michigan Press.

Raskin L., (1978), Performance Evaluation of Multiple Processor Systems, (Ph.D. Thesis), Technical Report, Computer Science Dept., Carnegie-Mellon University, Pittsburgh PA.

Reddy D.R., (1976), Speech Recognition by Machine: A Review, Proc. of the IEEE, Vol. 64, No. 4.

Riddle W.E., J.C. Wileden, J.H. Sayler, A.R. Segal, and A.M. Stanley, (1978), Behavior Modelling During Software Design, Proceedins of the 3rd Int. Conf. on Software Engineering.

Sacerdoti, E.D., (1974), Planning in a Hierarchy of Abstraction Spaces, Artificial Intelligence, Vol 5, no.2.

Sacerdoti, E., (1975), The Nonlinear nature of plans, International Joint Conference on Artificial Intelligence, Tiblisi USSR, Aug. 1975.

Schank R., and R. Abelson, (1977), Scripts, Plans and Understanding, Hillsdale NJ: Lawrence Erlbam Ass. Inc.

Schwald H.A., (1977), "Einfohrung in Die ASTRA76," Verfu/3t von der Sprachengruppe des LRZ, Muchen.

Shortliffe E.H., (1976), Computer-Based Medical Consultations: MYCIN, New York: American Elsevier.

Siewiorek D., V. Kini, H. Mashburn, S. McConnel, and M. Tsao, (1978), A Case Study of C.mmp, Cm*, and C.vmp: Part I - Experiences with Fault Tolerance in Multiprocessor Systems, Proceedings of the IEEE, Vol. 66, No. 10, Oct. 1978.

Sproull R.F., and D. Cohen, (1978), High-Level Protocols, Proceedings of the IEEE, Vol. 66, No. 11, Nov. 1978.

Simon H.A., (1957), Models of Man, New York: John Wiley & Sons Inc.

Simon H.A., (1962), The Architecture of Complexity, Proceedings of the American Philosophical Society, Vol. 106, No. 6, 467-487. (Also in Simon 1968)

Simon H.A., (1968), Sciences of the Artificial, Cambridge MA: MIT Press.

Simon H.A., (1976), Design of Large Computing Systems, In Organisatiewetenschap en praktijk, P. Verburg, P.Ch.A. Malotaux, K.T.A. Halbertsma, and J.L.Boers (Eds.), Leiden: H.E. Stenfert Kroese B.V.

Simon H.A., D.W. Smithburg, and V.A. Thompson, (1950), Public Administration, New York.

Smith Adam, (1776), Wealth of Nations.

Smith A.R. , (1976) Word hypothesization in the Hearsay-II speech system. Proceedings of the 1976 I.E.E.E. International Conference on Acoustics, Speech and Signal Processing, Philadelphia, 1976, 578-581.

Smith R.G., (1978), A Framework for Problem Solving in a Distributed Environment, (Ph.D. Thesis), Memo HPP-78-28, Computer Science Dept., Stanford University, Stanford CA.

Soloway E. and E. Riseman, (1977), Levels of Pattern Description in Learning, Proc. of the Fifth Int. Joint Conf. on Artificial Intelligence, Cambridge, MA, Aug., 1977.

Sussman G., (1975), A Computational Model of Skill Acquisition, New York: American Elsevier.

Swan R., S.H. Fuller, and D.P. Siewiorek, (1977), Cm*: A modular, multi-processor, Proceedings of the National Computer Conference, Dallas TX, June 1977.

Teitelman W., (1975), Interlisp Reference Manual, XEROX Palo Alto Research Center, Palo Alto, CA.

Teitelman W., (1977), A Display Oriented Programmer's Assistant, Proc. of the Fifth Int. Joint Conf. on Artificial Intelligence, Cambridge, MA., 1977.

Walker, D., et al., (1976), Final report of the SRI-SDC speech understanding system research. Menlo Park: Stanford Research Institute.

Weide B., (1978), Statistical Methods in Algorithm Design and Analysis, (Ph.D. Thesis)), Tech Rep., Computer Science Dept., Carnegie-Mellon University, Pittsburgh PA.

Williamson O.E, (1975), Markets and Hiearchies: A Transactional and Antitrust Analysis of the Firm, New York NY: The Free Press.

Wirth N., (1971), Program Development by Stepwise Refinement. CACM Vol. 14, No. 4, 221-7.

Woods W. A., et al., (1976), Final report of the BBN speech understanding system research. Cambridge: Bolt, Beranek, Newman, 1976.

Wulf W.A., and S.P. Harbison, (1978), Reflections in a pool of processors: An experience report on C.mmp/Hydra, Proceedings of the National Computer Conference, Anaheim CA, June 1978.

Wulf W, M. Shaw and R. London, (1977), An Introduction to Constructing and Verification of ALPHARD Programs, IEEE Transactions on Software Engineering, SE-2, Dec. 76.

Zucker S.W., (1976), Relaxation Labelling and the Reduction of Local Ambiguities, In Pattern Recognition and Artificial Intelligence, C.H. Chen (Ed.), New York: Academic Press.

Zucker S.W., (1977a), Production Systems with Feedback, In Pattern Directed Inference Systems, D.A. Waterman and F. Hayes-Roth (Eds.), Academic Press.

Zucker S.W., (1977b), Vertical And Horizontal Processes in Low Level Vision, Report No. 77-4, Electrical Engineering Dept., McGill University, Montreal Que.