Development And Implementation Of A Regional Water Planning Data Management System

A. B. Whinston

Follow this and additional works at: http://docs.lib.purdue.edu/watertech

http://docs.lib.purdue.edu/watertech/97

This document has been made available through Purdue e-Pubs, a service of the Purdue University Libraries. Please contact epubs@purdue.edu for additional information.
DEVELOPMENT AND IMPLEMENTATION OF A REGIONAL WATER PLANNING DATA MANAGEMENT SYSTEM

by

A. B. Whinston

JANUARY 1978

PURDUE UNIVERSITY
WATER RESOURCES RESEARCH CENTER
WEST LAFAYETTE, INDIANA
DEVELOPMENT AND IMPLEMENTATION OF A
REGIONAL WATER PLANNING DATA MANAGEMENT SYSTEM

A. B. Whinston

A completion report for Project B-080-IND
Project period: July, 1975 to December, 1977

Purdue University
Water Resources Research Center
West Lafayette, Indiana 47907

The work upon which this report is based was supported in part by funds provided by the U. S. Department of the Interior, Office of Water Resources and Technology, as authorized by the Water Resources Act of 1964.

Agreement No. 14-34-0001-5076

Technical Report No. 97

January 1978
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>LIST OF FIGURES</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF FIGURES</td>
<td>iii</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>v</td>
</tr>
</tbody>
</table>

## CHAPTER 1 - INTRODUCTION

| Preliminary Remarks                  | 1    |
| Computers and Decision Making        | 3    |
| Decision Support Systems             | 6    |
| Overview of the Generalized Intelligent Decision Support System | 11   |
| Organization of Thesis               | 17   |

## CHAPTER 2 - TRADITIONAL VARIETIES OF DATA STRUCTURE

| Structural Components of a Data Base | 19   |
| Varieties of Data Base Structure     | 22   |
| GPLAN Data Base Features             | 26   |

## CHAPTER 3 - PROCESSING FOR A NETWORK DATA BASE

| Limitations of Non-networks          | 34   |
| DML Interface with a Network Data Base | 37   |
| Query Interface with a Network Data Base | 41   |
| Methods of Program - Data Base Interface | 45   |
| Implementation of Section 208 Decision Support System | 48   |
| The Section 208 Problem              | 48   |
| Water Quality Simulation             | 51   |
| Implementation Considerations        | 54   |
| Details of the 208-GPLAN Implementation | 58   |
| Summary                              | 62   |
| Information Transferral within a Distributed Data Base | 62   |
| Varieties of Data Base Distribution  | 63   |
| Generalized Mapping Language         | 65   |
| Information Transferral              | 73   |
| Additional Considerations            | 76   |
| A General Data Structure for Mapping Support | 78   |
| A Scenario and Summary               | 81   |

## CHAPTER 4 - CONCEPTUAL FRAMEWORK FOR DECISION SUPPORT

<p>| Introduction                         | 84   |
| Rationale for Study of a Generalized Processor | 85   |
| Framework of Decision Making         | 87   |
| Requirements for Decision Support     | 93   |</p>
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Structural Features of a Data Base</td>
<td>21</td>
</tr>
<tr>
<td>2.</td>
<td>Varieties of Data Base Structure</td>
<td>23</td>
</tr>
<tr>
<td>3.</td>
<td>Many-to-Many Relation Between Occurrences of STATE and Occurrences of RIVER</td>
<td>25</td>
</tr>
<tr>
<td>4.</td>
<td>CODASYL Set Description</td>
<td>28</td>
</tr>
<tr>
<td>5.</td>
<td>Sample DDL</td>
<td>29</td>
</tr>
<tr>
<td>6.</td>
<td>A Simple Logical Structure</td>
<td>31</td>
</tr>
<tr>
<td>7.</td>
<td>Data Base Transformation</td>
<td>40</td>
</tr>
<tr>
<td>8.</td>
<td>GPLAN System</td>
<td>46</td>
</tr>
<tr>
<td>9.</td>
<td>Structure of the Mapping Processor</td>
<td>68</td>
</tr>
<tr>
<td>10.</td>
<td>Data Compaction</td>
<td>70</td>
</tr>
<tr>
<td>11.</td>
<td>Data Expansion</td>
<td>71</td>
</tr>
<tr>
<td>12.</td>
<td>Maps for Case IV of Data Base Distribution</td>
<td>75</td>
</tr>
<tr>
<td>13.</td>
<td>General Data Structure for the Support of Data Transferal Within a Logically Distributed Data Base</td>
<td>80</td>
</tr>
<tr>
<td>14.</td>
<td>Functional Facets of Decision Support Incorporated into the GENI Query Processor</td>
<td>90</td>
</tr>
<tr>
<td>15.</td>
<td>Resolution Levels for Representing Information about Cars</td>
<td>105</td>
</tr>
<tr>
<td>16.</td>
<td>Occurrences in Car Information Base</td>
<td>107</td>
</tr>
<tr>
<td>17.</td>
<td>Car Information Base - Structure of Occurrences</td>
<td>109</td>
</tr>
<tr>
<td>18.</td>
<td>Network Structure for the Car Data Base</td>
<td>110</td>
</tr>
<tr>
<td>19.</td>
<td>Structure of Occurrences for Network Car Data Base</td>
<td>113</td>
</tr>
<tr>
<td>Figure</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>20. Logical Structure Example (RSACH)</td>
<td>115</td>
<td></td>
</tr>
<tr>
<td>21. Logical Structure Example (GNC)</td>
<td>117</td>
<td></td>
</tr>
<tr>
<td>22. Logical Structure Example (WQMA)</td>
<td>118</td>
<td></td>
</tr>
<tr>
<td>23. Example of IDL for Figure 21</td>
<td>120</td>
<td></td>
</tr>
<tr>
<td>24. Logical Structure for Support of Problem Recognition</td>
<td>140</td>
<td></td>
</tr>
<tr>
<td>25. Occurrence Structure for Types of Analysis</td>
<td>145</td>
<td></td>
</tr>
<tr>
<td>26. Illustration of an AND/OR Graph</td>
<td>143</td>
<td></td>
</tr>
<tr>
<td>27. Two Problem Reduction Cases</td>
<td>148</td>
<td></td>
</tr>
<tr>
<td>28. Possible Information Base Structures for Representation of the AND Case</td>
<td>152</td>
<td></td>
</tr>
<tr>
<td>29. Types of Nodes in an AND/OR Graph</td>
<td>155</td>
<td></td>
</tr>
<tr>
<td>30. Graph Generation Operators: I</td>
<td>157</td>
<td></td>
</tr>
<tr>
<td>31. Graph Generation Operators: II</td>
<td>158</td>
<td></td>
</tr>
<tr>
<td>32. An Information Structure Defining a Set of Problem Reduction Operators</td>
<td>159</td>
<td></td>
</tr>
<tr>
<td>33. AND/OR Graph Generation (I)</td>
<td>162</td>
<td></td>
</tr>
<tr>
<td>34. AND/OR Graph Generation (II)</td>
<td>163</td>
<td></td>
</tr>
<tr>
<td>35. The Overlap Problem</td>
<td>166</td>
<td></td>
</tr>
<tr>
<td>36. Example of Structural Techniques for Representing Knowledge about Modification and Combination of Models</td>
<td>170</td>
<td></td>
</tr>
<tr>
<td>37. Occurrence Example of the Technique for Representing Model Combinations</td>
<td>171</td>
<td></td>
</tr>
<tr>
<td>38. Example of Strictly Network Structure for Representing Conceptual Information</td>
<td>173</td>
<td></td>
</tr>
<tr>
<td>39. Structural Composition of a Program Record Type</td>
<td>176</td>
<td></td>
</tr>
</tbody>
</table>
ABSTRACT

The focal point of this final report is an investigation and consequent specification of those abilities which a computerized decision support system must possess if it is to be of value to decision makers confronted by complex problems. These specifications are used in the design of a decision support system that is general, in the sense of its ability to support a broad range of decision applications. This system's query processor is intelligent, in that it emulates certain human cognitive processes. These include the decoding of incoming English-like messages; the retention and integration of various types of information in a single mechanism (an information base); the activity of associating outward perception emanating from the system's users, with inward perceptions from the information base; the ability to recognize a user's intent, in spite of non-procedural or ambiguous queries; the formulation of models for solving a problem; and the automatic execution of a desired model with desired data.

This work is an extension to the Generalized Planning (GPLAN) System developed at Purdue University. The approach has been to integrate techniques from the field of data base management (e.g., GPLAN) with artificial
intelligence techniques and concepts, culminating in an intelligent
decision support system. The final report commences with a survey of
the emerging discipline of decision support systems. An example of a
water quality management application is given within the CPLAN framework.
Several extensions and corrections for the CODASYL approach to data man-
agement are given.

There is an investigation of problems encountered in the manage-
ment of data that is logically distributed. An example of this situation
is drawn from the realm of water quality management. A generalized
mapping language is proposed as a mechanism for information transferral
within a distributed data base, and a general data structure for sup-
porting the mapping function is illustrated. The presented method
accommodates a variety of user views, is independent of whether the
data base is geographically distributed or centralized, furnishes a
straightforward security mechanism and provides a basis for treating the
contingency of uninformed or non-programming users.

When considering the design and implementation of systems for de-
cision support, a crucial point is the power and flexibility of available
tools for representing data contexts. The value of such systems is con-
strained by the 'richness' of patterning allowed by their data structure
mechanisms. The notion of an information base is introduced as a natural
step forward in the continuing evolution of data structures. The out-
standing features of the information base are its accommodation of the
horizontal and vertical integration of information parcels into a single
semantic mechanism, and the integration of operators into this semantic
structure.
An important part of research into the development of a generalized and intelligent query processor for decision support concerns processing at the deep structure level. The final report addresses certain aspects of this level of processing, which commences after a user's query has been decoded into an internally usable representation. Since the query processor is more than a mere retrieval mechanism, the deep structure processing involves its abilities of model formulation, problem recognition and analysis. All of these abilities depend upon the processor's access to an independent information storage mechanism called the information base. It is this property which renders the query processor invariant to changes in the decision application to be supported. Particular emphasis is given to the connection between the information base structure and the artificial intelligence technique of problem reduction.
CHAPTER 1

INTRODUCTION

Preliminary Remarks

The news of a seventy million dollar settlement between United States Steel and the Environmental Protection Agency was recently (June 17, 1977) reported in the Wall Street Journal. United States Steel agreed to spend this large sum for the implementation of water pollution abatement systems at its manufacturing facilities along the Grand Calumet River. Underlying this development was the use of an information system in establishing load allocations for the Grand Calumet [43]. This load allocation problem is typical of many complex problems deriving from the Federal Water Pollution Control Act Amendments of 1972, with the objective of enhancing and maintaining environmental integrity. These problems involve large amounts of interrelated information and often times complex models. And as the above example indicates they have broad social and economic ramifications. In addition, these problems usually confront decision makers who are not programmers or information system experts. From a broader perspective, this state of affairs confronts decision makers across the entire spectrum of the public and private sectors. Thus it is that a topical and decidedly significant area of research involves the identification of those criteria which a computerized system must satisfy if it is to be of value to decision makers.
Historians of the future, looking back upon our present era, may very well characterize it as the information age. For it is a period wherein the volume and complexity of information processed by individuals and organizations has grown dramatically. The interdependence and rapidity of information processing is unprecedented. Simon [76] has observed that we are experiencing the initial stages of an information revolution which began a bit more than a century ago. He terms it the Third Information Revolution, indicating that its two predecessors resulted in written language and the printed book.

The present revolution in the technology of information processing includes the photograph (and motion picture), the phonograph, radio, television and the telephone. These are devices for the storage, transmission and display of information. There is yet another device that is prominent in this revolution: the computer. The computer differs from the other devices in that one of its major functions is to transform information, as well as storage and display. This property gives the computer a special significance when it is realized that the human thinking process also involves the transformation of information. The computer is the only means presently available with a potential for modeling human thought processes [37], [76].

To the extent that such modeling can be accomplished, it has at least two primary (and related) objectives [74]. The first involves increasing our understanding of the nature and operation of thought processes. The thought processes with which this dissertation is concerned are those which comprise the decision making activity. And this leads us to the
second objective: that of modeling certain aspects of the decision making activity in order to support decision makers in their efforts to effectively address complex, unstructured problems. The succeeding chapters explore the requirements, design and implementation of a general computerized decision support system.

Computers and Decision Making

At this juncture it is necessary to make a few general remarks about computers and decision making. These are followed by a survey of the emerging discipline of decision support systems. It is well known that the capabilities of computers are constrained principally by elementary processing speeds and by memory sizes (subject to real time retrieval). It has been argued [76] that all limits on the potential scope of "computer intelligence" are also restrictions to human intelligence and that "the scale of available computer memories is increasing rapidly, to the point where memory size may not be much longer an effective limit on the capacity of computers to match human performance."

If these assumptions are rejected then the practical viability of a general computerized decision support system (not to mention a general computerized decision making system) must be severely limited. (The term general system, as it is used throughout this dissertation, indicates a system whose design does not restrict it to consideration of problems in a single application area or in a small class of application areas.) On balance, however, the study of a general computerized decision support system can offer insights that are useful in the study of specialized systems. It can also offer a framework for the implementation of
specialized decision support systems or of general decision support systems which have more modest capabilities than those discussed in the ensuing chapters (e.g., [43]). If time shows the above assumptions to be correct then the practicality of general decision support (and decision making) systems is, of course, greatly enhanced. Thus, in the very least, the basic research to be presented here posits a conceptually viable framework for studying and communicating about decision support systems (that are general and "intelligent") in terms of techniques which are known to be implementable.

What does it mean to make or support a decision? It has been observed [33] that most definitions of decision making intimate that there is a decision maker faced with several alternatives and that the decision maker chooses an alternative based on comparisons among the alternatives and on evaluations of their (anticipated) outcomes. Simon [75] declares that decision making "comprises three principal phases: finding occasions for making a decision; finding possible courses of action; and choosing among courses of action." For the present we shall utilize Simon's definition of decision making in terms of phases. Later a model of decision making will be proposed in terms of the abilities decision making requires on the part of the decision maker. These abilities serve to indicate functional facets needed for a decision making (or support) system. (In the context of a computer system, a "functional facet" refers to computer code which performs the function in question.) Each of the three phases is executed by utilizing one or more of the appropriate functional facets.
Decisions fall along a continuum that ranges from highly structured to highly unstructured. These two endpoints are also referred to as programmed and nonprogrammed respectively [42]. The former refers to decisions that are routine and repetitive. The latter is descriptive of situations where there is not a "cut-and-dried method for handling the problem because it hasn't arisen before, or because its precise nature and structure are elusive or complex, or because it is so important that it deserves custom-tailored treatment" [75]. A program is a strategy for processing information. The psychological perspective taken here is that human memory contains programs or strategies for processing information [32], even though they may be unable to articulate (or are unconscious of) the strategy employed in a given decision making instance. A discussion of this view and the contrasting behaviorist position appears in [36].

The question arises concerning what governs a decision in the unstructured, non-programmed situation. Simon [75] maintains that the governing agent is a set of rules of procedure, that the individual has some general problem solving strategies (i.e., programs) in addition to specialized strategies. It will become apparent in later chapters that this is the approach to be adopted in the design of the generalized intelligent decision support system. Indeed this approach is used at two levels. First, within the system itself, the specialized (varying degrees of specialization are permitted) strategies are maintained along with more conventional kinds of data in a memory mechanism called the information base. The most general strategies comprise the decision support
system's query processor. This is a step in the direction of matching human flexibility; with respect to the trait of flexibility, humans hold a definite advantage over extant decision making (and decision support) systems. The second level at which the above approach is used is at the system-user interface. That is, the user's own general problem solving capacities are used to pose a series of queries, each of which evokes specific skills and specific knowledge of the system.

A related issue which is not directly addressed by this thesis, but which is of interest as a topic for future research, concerns the extent to which the proposed general support system can learn. Human strategies can be improved by instruction and experience. In the system to be considered here, instruction can be accomplished by adding to, modifying and deleting from programs existing in the information base (just as the same operations can be applied to more traditional data). The notion of program improvement through experience suggests the need for usage tracking and a program which can modify the programs existing in the information base (according to protocols involving past usage).

Decision Support Systems

With this background we can consider more precisely what is meant by computerized decision support. The broad outlines of this emerging discipline have only recently begun to take shape. It has been estimated [76] that, to date, ninety-five percent of all computer power has been consumed in record keeping and in performing large-scale scientific and engineering computations. But as we shall see, computerized decision support involves a good deal more than this. A survey of the small, but
rapidly growing, literature concerned with decision support reveals comparatively little practical experience and an absence of fundamental theory. It further reveals that the primary impetus for interest in this area comes from advances in the field of data base management, rather than from any advance in our understanding of the mechanisms of decision making.

The various definitions that have been suggested for decision support (e.g., [7],[8],[38],[40],[52],[56]) will not be listed here. It suffices to observe that these definitions agree that the system must aid a decision maker in solving unprogrammed, unstructured (or 'semi-structured') problems. A review of the definitions also shows a widespread agreement that the system must possess an interactive query facility, with a query language that resembles English (or is at least easy to learn and use). It has been argued [52] that the main reason that computerized information systems have not been used for decision support is their inaccessibility to nonprogramming decision makers.

An examination of existing systems which have been called decision support systems (e.g., [8],[42],[52],[55],[56]) shows that each is composed of a data base management package plus some sort of query facility. The query facility typically permits ad hoc queries for retrieval and in some cases handles ad hoc analyses. Alter [3] has discussed six types of decision support systems, including examples of each. This typology is based on uses of the system and consists of 1) retrieval of isolated data values, 2) performance of ad hoc analysis, 3) production of standard reports, 4) estimation of consequences of proposed decisions, 5) proposal
of decisions, and 6) making decisions. With few exceptions (e.g., [43]), systems which have been termed decision support systems are application specific.

The perspective on decision support which will be held throughout the remainder of this work is that it can (and should) be approached as a synthesis of both data base management techniques and artificial intelligence techniques. That is, the data base and query facilities can make use of artificial intelligence techniques. In a rather exhaustive survey, Wong and Mylopoulos [84] have exhibited that there is a strong relationship between recent research in artificial intelligence and topical issues in the data base management area. These two can be compared and contrasted with respect to three questions: how is knowledge represented; what is the nature of the system-user interface; what applications are dealt with? Thus they share the same broad objectives, although their approaches to meeting these objectives have different starting points.

Regarding the representation of knowledge, artificial intelligence has concentrated upon the modeling of abstract, rather than concrete, knowledge. The way in which these models have been implemented is "often haphazard and rarely formalized to the point where they could be duplicated with any degree of success" [84]. In contrast, data base management, has been primarily concerned with representing large volumes of concrete knowledge and its treatment of abstract knowledge has been comparatively meager. However, as one traces the evolution of data base management, there is a discernable trend in the direction of representing more and
more complex types of abstract knowledge (see Chapter 2). In Chapter 5, the information base is introduced, providing an added dimension to the customary representation of abstract knowledge in a data base. A final point about the data base management approaches to knowledge representation is that they are considerably more formal with respect to implementation [47] than those of artificial intelligence.

A common ground between the two approaches to knowledge modeling is that researchers in both areas have used some of the same logical constructs for design of their models. Both have been formulated in terms of predicate calculus (e.g., in artificial intelligence [39],[70],[77] and in data base management [29],[78]). Both have been formulated in terms of binary predicates (e.g., [6],[67] and [41],[72]). The connection between semantic network formalisms in artificial intelligence and data base networks (with certain extensions) is considered in Chapter 5. In addition it has been argued [14],[84] that data base networks and predicate calculus schemes are compatible and complementary.

Proceeding to the second question of user-system interface the trend in data base management has been a progression from procedural, programming languages towards nonprocedural, English-like interface languages [47]. These are based on the myriad parsing and compiling techniques developed by computer scientists.

The emphasis in artificial intelligence has been on nonprocedurality and on natural language processing. This nonprocedurality is customarily accomplished by evaluating queries with predicate calculus theorem proving methods [18]. These methods have been used for special purpose
systems (e.g., [67]) by inserting inference rules into the system's programs. The resolution principle [69] has been used to build more general systems (e.g., [39]) that are independent of predicate calculus rules which they use. Others [9],[60] have suggested interaction with the user in order to select the rules needed to execute a proof.

Another method for answering queries involves search and backtracking mechanisms such as those used in PLANNER [79]. In this case, the query is taken as an assertion to be proven by searching (with backtracking) a data base. Automatic path determination in the GPLAN system [13] bears a strong resemblance to the PLANNER technique, despite GPLAN's data base management origins. GPLAN can also be characterized as using the state space problem solving method from the field of artificial intelligence [48]. Yet another artificial intelligence technique for question answering is problem reduction. This method receives extensive consideration in Chapter 6.

Artificial intelligence research also involves natural language processing. This may be viewed as a type of automatic code generation. Existing studies in this area (e.g., [10],[83],[85]) are concerned with allowing human-computer interaction via some subset of the English language and they are closely related to the previously mentioned representation of knowledge by semantic networks. The query language to be described in Chapter 3 has an English-like appearance; but it is not, strictly speaking, a 'natural language.' It is an approximation. The desirability and utility of strictly natural language discourse is a point of controversy [51],[59]. From one viewpoint marginal utility diminishes as we
approach a strict natural language capability, while marginal cost increases sharply.

With respect to the third question raised previously, data base systems have been extensively used in practical applications. Most natural language systems (and other artificial intelligence query answering systems) deal only with toy problems and are application-specific. There are exceptions. The NLP system [49] is used for constructing and executing simulation models. More general systems, in terms of application areas supported, are REL [80] and REQUEST [66]; but these systems are primarily for retrieval. Descriptions of natural language processing systems typically detail various schemes of knowledge representation and describe specific implementations of semantic nets. In Chapter 5 well-known constructs (with certain extensions) from the data base management discipline are utilized to provide a single mechanism for data storage (consisting of both programs and traditional data) and abstract semantic representation. In summary, the convergence of data base management with artificial intelligence, culminating in systems for decision support, is demonstrated throughout the remaining chapters.

Overview of the Generalized Intelligent Decision Support System

The generalized intelligent decision support system developed in this dissertation has two principle components: the information base and the generalized intelligent query processor. A brief overview of each is provided to serve as an introduction to the chapters which follow.
A point which cannot be emphasized too strongly is that the information base is not a part of the query processor. It may be thought of as a single device that contains all knowledge which is available to the query processor, independent of interaction with a user. This stored knowledge may be empirical, conceptual, algorithmic etc. Thus the information base is application area-specific in terms of its structure and content. It will become apparent that it is this property which allows the query processor's invariance to the application area being considered, since the processor can access the knowledge of any information base regardless of its structure and content.

The information base notion is basically an extension of well-known database techniques to provide a more complete semantic mechanism for storing knowledge. From the database management viewpoint, the information base furnishes a fundamentally new technique for structuring information. It has all of the capabilities afforded by network structures, but it also provides a means for integrating networks, of varying levels of semantic resolution, into a single structure. Here network structures are referred to in the CODASYL sense [27], subject to some important extensions and modifications detailed in Chapter 2. The information base also incorporates another departure from data base management as it is customarily practiced: namely, the integration of program modules into the information base so that they can be managed just as the more traditional type of data.

We can begin our consideration of the query processor by recalling that it is intended to support decision makers, rather than to make
decisions. This support is furnished by virtue of the processor's ability
1) to collect information from the user and from a central repository
which we call the information base, 2) to recognize the intentions of
the user (even though they are not explicitly stated) and express those
intentions as a set of primitive problems which are known to be solvable,
3) to formulate models by combining modules of computer code, and 4) to
interface formulated models and desired data with subsequent execution.
The query processor is said to be general in the sense that it can sup-
port decision making in a wide variety of application areas (e.g., water
quality management, nuclear power plant siting, health care management,
etc.). Indeed, the query processor is itself invariant to the application
area being supported, to the kinds of formal models which are to be used,
to the ways in which a model may be used in carrying out desired analysis,
and to the vocabulary which a user desires to utilize in conversing with
the processor. That is, the query processor's two sources of information
(i.e., the user and the information base) may undergo many changes, but
no change in the query processor's code is required.

The query processor could be said to be intelligent because of its
generality which enables it to successfully cope with new situations. In
this discussion however, the term "intelligence" should be considered in
the sense of artificial intelligence. Thus the query processor endeavors
to emulate certain human cognitive abilities, namely the four abilities
enumerated in the preceding paragraph. With respect to the information
gathering ability, information is obtained from the user by decoding
English-like messages to give the shallowest of deep structure expressions.
This is accomplished by applying inverse transformations (in the sense of 
Chomsky [26]) to the surface structure query. In the subsequent process 
of 'understanding' a query, there is a progression from one deep structure 
(which could be called intermediate) to a 'deeper' structure, to a still 
deeper structure, and so forth. One could consider the code of the 
model, which is eventually formulated, to be the deepest structure of all. 
Information is obtained from the information base by finding paths through 
a semantic network.

The second ability (i.e., problem recognition) forms the core of the 
query processor, in that it makes extensive use of the ability to formu-
late models and the ability to gather information from the two sources 
just mentioned. Its primary role is to discover the model and the data 
which the user desires to be interfaced and executed. The user is not 
required to provide an explicit statement of how the problem is to be 
solved, nor even a precise statement of just what the problem is.

In intelligent inter-human discourse there are frequently implica-
tions which are attached to messages by both recipients and senders 
(i.e., implications that are not explicitly stated). In order to 
recognize the intent of a message, the recipient must examine that 
message in the light of stored information about the sender and the prob-
lem area (i.e., the context) under consideration. Discourse between an 
intelligent query processor and its user should also address these 
economies of expression. Thus during the problem recognition activity, 
the processor utilizes pertinent stored information (and possible inter-
action with the users) in order to precisely determine the user's intent.
Parenthetically, it must be noted that just as there may be a misinterpretation of intent during inter-human discourse, so is there the possibility of the same type of misinterpretation during human-computer discourse.

The third ability, that of model formulation, involves the generation of potential algorithms to be used for the analysis of data. This is accomplished by modifying and combining various known program modules. Rules for the permissible modifications and combinations are incorporated into the structure of the information base and the modules themselves are treated as data in the information base. Thus program modules may be added and deleted from the information base just as easily as other sorts of data. The model formulation activity for a given query begins in concert with the problem recognition activity. Suppose that a user's intentions are not explicitly stated with respect to the model which is to be used in answering a query. Before the problem recognizer can produce the explicit statement, it must have available to it a description of the models which could possibly have been intended by the user. It is through the model formulation activity that these descriptions are generated. One way to view the formulation activity is as a problem reduction process; this view is adopted and developed in Chapter 6. Briefly, a portion of the decoded query is viewed as an initial problem description which can be reduced to one or more sets of directly solvable primitive subproblems. It will be shown how the structure of the information base is used to specify permissible reduction operators. If it occurs that more than one set of primitive subproblems is generated (i.e., more than
model is formulated), then the problem recognizer resolves the ambiguity through interaction with the user or through the use of pertinent stored information. The problem recognizer must also eliminate any ambiguity as to the data which the user intends to be accessed by the model.

Once the explicit statement of a user's intention has been produced, it is used to direct the fourth ability of the query processor. This ability is referred to as analysis, since it involves the interfacing (and execution) of the desired model and the desired data in order to provide some facts or expectations for decision support. The "explicit statement" which governs the analysis process is directly analogous to the JCL (Job Control Language) which a typical computer-user submits to that computer's operating system. And the analysis process corresponds to a limited sort of operating system. In the remainder of this presentation, mention of a query's deepest structure refers to this "explicit statement." The two terminologies are used interchangeably. The analysis process commences by compiling the deep structure query. It proceeds to sequence the job steps, assign I/O locations, perform data mappings, load code, and execute.

To conclude this overview, it is useful to briefly outline the major contributions of this work to the field of decision support systems. There is, first of all, the introduction of the information base and all that it entails. The information base can be viewed as a natural step forward in the evolution of data structures, a step that bridges the gap between knowledge representation schemes of artificial intelligence and data base management. Secondly, there is the notion of managing program
modules by utilizing data base (or information base) management techniques. A framework for decision support is proposed in terms of the abilities which must be implemented. Using linguistic principles and artificial intelligence techniques, it is shown how the needed decision support abilities can be implemented such that the resultant general query processor has far-reaching properties of invariance. Thus the present work could be described as a pioneering effort in the largely unexplored realm of generalized decision support. This effort proceeds along the two complementary approaches of artificial intelligence and data base management.

Organization of Report

The material in Chapter 2 offers a unified treatment the traditional varieties of data structures, showing the evolution of data structuring techniques. In so doing, the basic terminology for discussing logical data structures is established. Unless it is noted otherwise, all references to data structures are at the logical level; physical aspects of data structuring are examined in [47]. This chapter also introduces several extensions and corrections for the CODASYL [27] data definition proposals.

Chapter 3 presents a description of processing based upon the data structures of Chapter 2. The focus is upon the Generalized Planning (GPLAN) System developed by investigators in the Krannert Graduate School of Management, Purdue University. A procedural data manipulation language, a nonprocedural (English-like) query language, and the issue of data base-application program interfaces are addressed. GPLAN is the forerunner of
the generalized intelligent decision system. Another major aspect of this chapter is the examination of processing data in a distributed environment, via a generalized mapping language. Finally it is shown how the foregoing tools can be applied to the construction of a decision support system for area-wide water quality planning.

The fourth chapter delineates the conceptual model of decision making which is used as a guide in the subsequent development of the generalized query processor for decision support. It serves to identify the abilities which that processor must possess, abilities which can be implemented using data base management, artificial intelligence and linguistics techniques. The fifth and sixth chapters have already been alluded to; they consider the design and implementation of the information base and query processor, respectively. The final chapter summarizes this work and outlines avenues for future research.
CHAPTER 2

TRADITIONAL VARIETIES OF DATA STRUCTURE

Structural Components of a Data Base

For the purposes of this discussion, the term data base is defined by two attributes: a schema and a set of data value occurrences which are logically organized according to that schema. A schema is an archetypal logical structure, a blueprint of the contents of the data base; physical storage structures are not considered here. The fundamental components of the schema are data item types; for instance, RIVER-NAME, BASIN-NAME, and ELEVATION are types of data. Each data item type represents many data values; RIVER-NAME may have WHITE, WABASH and CALUMET as data value occurrences. The schema also specifies the relationship of each data item type with other data item types.

There are two varieties of relationship among data item types: aggregation and association. Data item types may be aggregated into records types; for example BASIN-NAME and ELEVATION may be aggregated to form the record type BASIN. A sample record occurrence is CALUMET and 780. Alternatively, record types (and therefore data item types) may be associated with one another by means of an in-set relation. An in-set is defined in terms of owner and member record types where there is a one-to-many relation between owner and member occurrences. That is, for each owner occurrence, there are many member occurrences associated with it; but for a particular in-set, a given member occurrence may be associated with no more than one occurrence of the owner record type.
It should be noted that an in-set is very similar to the 'set' concept outlined in the CODASYL Data Base Task Group (DBTG) Report of 1971 [27]. The DBTN term 'set' is entirely unrelated to the mathematical notion of a 'set.' In order to avoid confusion, the convention of using the term 'in-set' (INformation SET) has been adopted to refer to the DBTN 'set,' subject to extensions explained below.

The member occurrences associated with an owner occurrence may be logically ordered according to some criterion; so that an in-set provides not only information regarding the relation among owner and member records, but a means for accessing data of member occurrences as well. Suppose that RIVER is the name of a record type composed of RIVER-NAME and various other item types to represent information relating to a river's state characteristics. Since there are a number of rivers associated with each basin, we can define the in-set CONTAINS with BASIN as its owner record type and RIVER as its member record type. Whereas an in-set indicates a one-to-many relationship among data occurrences, data occurrences of item types that have been aggregated in a record type have a one-to-one, one-to-many, or many-to-many relation. For the sake of completeness, it must be pointed out that the DBTN Report suggests that a data item type may be defined to be an array (e.g., let the data item type POLLUTANT-NAME be an array, then an occurrence of this item might consist of PHOSPHATE, NITRATE, NITRITE, and IRON).

It is frequently convenient to represent a data base pictorially. The notation in Figure 1 is used for this purpose. Figure 1a shows the record type BASIN (represented by a rectangle) composed of the data item
a. The Record Type BASIN

b. Association of Item Types Via an In-set

Figure 1. Structural Features of a Data Base
types BASIN-NAME and ELEVATION. In Figure 1b, the set CONTAINS is denoted by the arrow pointing from the set's owner (BASIN) to its member (RIVER). Thus for each occurrence of BASIN there may be several occurrences of RIVER (e.g. the 'OHIO' basin may own the rivers 'WABASH', 'CUMBERLAND', etc.). But a particular river cannot be owned by more than one basin via the set CONTAINS (e.g. the 'WABASH' river may not be owned by both the 'OHIO' and 'AMAZON' basins).

Varieties of Data Base Structure

Using the terminology developed in the previous section, we can illustrate the sorts of data base structures which are in common use today. The simplest data base structure consists of a single record type (Figure 1a). A more complex situation occurs when the data base is organized according to a number of disjoint record types. Figure 2a presents a linear structure; that is, each record type is the owner of no more than one in-set, nor is it the member of more than one in-set. A data base may consist of data values organized on the basis of several disjoint linear structures. The more flexible tree structure is shown in Figure 2b. Note that a record type may be the owner of more than one in-set, but it can be the member of no more than one. Hence, there is a unique path between any two record types. The most general of all data structures is the network (Figure 2c), since the record type, the linear structure and the tree are special cases of it.

Because a record type in a network may be the owner of many in-sets and the member of many in-sets, it is permissible for multiple paths to exist between two record types. The term path is used to indicate a
Figure 2. Varieties of Data Base Structure
sequence of in-sets that are related via record types, such that any two record types on the path are related by a unique sub-sequence of the path's in-sets. Be careful to note that though a path is composed of directed arcs, the path definition contains no notion of direction; the direction of arcs within a path is immaterial for purposes of traversing record types. Figure 3 illustrates a special kind of path that may exist in a network. The path between STATES and RIVERS is composed of the two in-sets CONTAIN and CONTAINED-IN, related through the record type LENGTH. A path of this variety is used to indicate a many-to-many relationship; i.e., a RIVER may be CONTAINED-IN in many STATES while a STATE may CONTAIN many RIVERS. Each occurrence of LENGTH denotes the length (and related characteristics) of a particular river within a particular state. To reiterate an important point, the direction of the arrows in a path is unimportant for purposes of path specification; within a given path, we may move 'upstream' and/or 'downstream'.

The distinctions among the types of data structures can be highlighted by several analogies and examples. The data structure consisting of a single record type corresponds to a FORTRAN array. Disjoint record types are analogous to several FORTRAN arrays. Linear structures may be thought of in terms of various levels of header and detail records within a single file. A collection of disjoint linear structures is the most complex data structure which can be handled by FORTRAN. Tree structures can be defined for use by the COBOL programming language. A collection of disjoint trees is the most complex data structure that can be handled by COBOL. Observe that trees can be decomposed into disjoint linear structures, which can
Figure 3. Many-to-Many Relation Between Occurrences of STATE and Occurrences of RIVER
themselves be decomposed into disjoint record types. But each such decomposition necessarily introduces redundant data and the associated problem of maintaining consistency among the disjoint structures in updating situations. Reversing the decomposition process, linear structures can be constructed from disjoint record types by eliminating redundancies. Disjoint linear structures can similarly be combined into trees.

It should be noted that the progression of data structures from record types through networks reflects a trend towards the ability to incorporate increasingly complex types of abstract knowledge into a database. The remainder of this chapter focuses upon the most powerful of these abstract knowledge representations, namely the network. This examination involves certain extensions to the CODASYL notion of a network. Observe that a network can be decomposed into disjoint trees and that disjoint trees can, through the elimination of redundancies, be used to construct networks. A basic principle behind the information base structuring techniques to be presented in Chapter 5, is that it is possible to integrate disjoint networks into multi-level networks.

**GPLAN Data Base Features**

The GPLAN data base features are largely based on the CODASYL DBTG Report. Thus GPLAN supports network data bases. Data bases for a wide variety of applications have been studied within the GPLAN framework (e.g., [17],[18],[19],[21],[23]). The schema of a data base is formally defined in terms of a Data Description Language (DDL). The DDL is not a programming language, nor should it be biased in the direction of a particular programming language. Rather, it provides a means for complete
specification of the data base's logical structure (independent from physical structure considerations) such that that structure can be utilized by various programming languages for storage, retrieval and modification of data values in the data base [28]. The DDL associated with GPAEL differs in several respects from that proposed by CODASYL groups [27],[28]. Interestingly enough Nijssen outlines similar changes in his proposal of an industry standard for DDL [61],[62].

1. First of all the GPAEL DDL [12] is considerably less complicated syntactically, thereby facilitating data base definition. For example, in order to define an in-set in terms of the CODASYL DDL, the syntax depicted in Figure 4 is required. The GPAEL DDL to define the in-set RVBR whose owner is the record type BASN and whose member is the record type RIVR is shown in Figure 5b, where the record types have been defined using the DDL depicted in Figure 5a. As the DDL indicates, the order in which the occurrences of RIVER, associated with a particular occurrence of BASIN, may be accessed is based on the item occurrence RIVER-NAME. That is, membro occurrences are logically organized according to the alphabetical order of their respective river names.

2. Secondly, unlike the CODASYL specification, the DDL for the GPAEL system allows a record type to be declared to be both the owner and member of a single in-set. For example, in a pollution control agency, there are many employees. Most of these employees have managers, and the managers are themselves employees who may work under higher level managers. This logical relationship can be compactly represented
SET SELECTION [FOR set-name-1] IS
THRU set-name-2 OWNER IDENTIFIED BY

\[
\begin{align*}
\text{SYSTEM} \quad \text{CURRENT OF SET} \\
\text{DATA-BASE-KEY} \quad \text{EQUAL TO} \quad \{ & \text{data-base-identifier-7} \\
\text{CALC-KEY} \quad \text{EQUAL TO} \quad \{ & \text{data-base-identifier-8} \\
\text{MEMBER record-name-2 SELECTION} \quad \{ & \text{data-base-data-name-2} \\
\text{THEN THRU set-name-3} \quad \{ & \text{data-base-identifier-9} \\
\text{WHERE OWNER IDENTIFIED BY data-base-identifier-10} \quad \{ & \text{data-base-identifier-11} \\
\text{EQUAL TO} \quad \{ & \text{data-base-data-name-4} \\
\text{PROCEDURE data-base-procedure-2} \quad \{ & \ldots \ldots
\end{align*}
\]

Figure 4, CODASYL Set Description
<table>
<thead>
<tr>
<th>RECORD</th>
<th>ITEM</th>
<th>NAME</th>
<th>TYPE</th>
<th>SIZE</th>
<th>MAX</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>ITEM</td>
<td></td>
<td>BASIN</td>
<td>INTE</td>
<td>1</td>
<td>1</td>
<td>BASIN</td>
</tr>
<tr>
<td>ITEM</td>
<td></td>
<td>BSID</td>
<td>CHAR</td>
<td>20</td>
<td>1</td>
<td>BASIN ID</td>
</tr>
<tr>
<td>ITEM</td>
<td></td>
<td>BSNA</td>
<td>CHAR</td>
<td>20</td>
<td>1</td>
<td>BASIN NAME</td>
</tr>
<tr>
<td>ITEM</td>
<td></td>
<td>E</td>
<td>REAL</td>
<td>1</td>
<td>1</td>
<td>MEAN ELEVATION</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RECORD</th>
<th>ITEM</th>
<th>NAME</th>
<th>TYPE</th>
<th>SIZE</th>
<th>MAX</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>ITEM</td>
<td></td>
<td>RIVR</td>
<td>INTE</td>
<td>1</td>
<td>1</td>
<td>RIVER IN BASIN</td>
</tr>
<tr>
<td>ITEM</td>
<td></td>
<td>RVID</td>
<td>INTE</td>
<td>1</td>
<td>1</td>
<td>RIVER ID</td>
</tr>
<tr>
<td>ITEM</td>
<td></td>
<td>RVNA</td>
<td>CHAR</td>
<td>20</td>
<td>1</td>
<td>RIVER NAME</td>
</tr>
</tbody>
</table>

a. Formal Definition of Record Types

<table>
<thead>
<tr>
<th>SET</th>
<th>OWNER</th>
<th>MEMBER</th>
<th>NAME</th>
<th>ORDER</th>
<th>KEY</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RVID</td>
<td></td>
<td>RVIB</td>
<td>SORT</td>
<td>RVID</td>
<td>ALL RIVERS IN BASIN</td>
</tr>
<tr>
<td></td>
<td>BASN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>BASIN</td>
</tr>
<tr>
<td></td>
<td>RIVR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RIVER IN BASIN</td>
</tr>
</tbody>
</table>

b. Formal Definition of an In-set

Figure 5. Sample DLL
by a single record type containing employee information and a single
in-set representing the possibility of many employees having a single
manager.

3. The CODASYL DDL proposes that item types be allowed to be arrays and
that an in-set be allowed to have more than one record type declared
as its member. Although both of these features were implemented in
the original version of the GPLAN DDL neither is currently used, since
they detract from the conceptual simplicity of the logical network
structure, without adding any intrinsic knowledge representation
capabilities.

4. By convention, in-sets in a GPLAN schema description are defined such
that a special record type named SYSTEM is made the owner of every
other record type of the data base. Not only does this provide a
simple means for allowing entry into the data base through any record
type, but it also allows a direct access path to the occurrences of
each record type in a network. Consider the schema depicted in
Figure 6, where each box represents a record type, and each arrow
represents an in-set with the arrow pointing towards the in-sets member.
Note that there is a many-to-many relation between STATE:RIVER, and
also between STATE:BASIN. Suppose that we can enter the data base
only through the record type STATE (or BASIN); then a fairly involved
procedure is required to obtain a list of the RIVER occurrences that
exist in the data base. If, however, we can enter the data base
through the record type RIVER, then access of the RIVER occurrences
is straightforward. This feature is particularly important where data
Figure 6. A Simple Logical Structure
structures are large and complex. It should be noted that utilization of SYSTEM-owned in-sets is only one means (and not necessarily the most efficient) of organizing entry points. Nijssen [61] suggests an alternative syntax for handling this.

5. Each record occurrence in the data base has a unique physical identifier (address) called a data base key. The GPLAN system supplies this key whenever a record occurrence is created. With the DDL, the user is allowed to declare a logical identifier for the record type with respect to each in-set of which it is a member. Given a particular owner occurrence of an in-set, this logical identifier is the key by means of which member record occurrences associated with that owner occurrence may be accessed (and stored). In contradistinction to the CODASYL DDL which permits declaration of logical identifiers in several places, this may be accomplished in only one place (e.g., R-NAME in Figure 2b) in the GPLAN DDL.

Nijssen suggests that, for reasons of data integrity, each record occurrence in the data base should be uniquely identifiable, based on the value of one or more data item types that have been declared to be that record type's identifier; and a record type must have one, but may have more identifiers. This has yet to be implemented within the GPLAN/DMS. In the present system, a record type may have no identifiers (a dummy record). Or, it may have one or more logical identifiers since the occurrences of the record are allowed to be sorted in different ways with respect to different in-sets. Nijssen proposes that each record type should have one or more identifiers
and that identifiers should be defined independently of in-set definitions. Compared to the present implementation, this would remove the burden of responsibility for integrity of the logical database from the data base administrator, at the expense of increasing storage requirements and reducing user flexibility. That is, in the present implementation, the data base administrator must guarantee integrity (to the extent that it is desired) via checks for it in the application programs that create, alter, or delete data value occurrences.

6. Nijsen further submits that attributes should be allowed to be represented as data items only, and not be permitted to remain implicit in in-set definitions. He contends that allowing attributes to be represented both ways tends to create a lack of symmetry, undue complexity and the possibility of DML reprogramming in response to schema alteration. The GPAN/DMS allows user to adhere to this rule if desired, noting that this diminishes flexibility and increases storage requirements. The issue of schema-subschema translation and reprogramming in an environment where the schema is subject to frequent modification is discussed later.
CHAPTER 3

PROCESSING FOR A NETWORK DATA BASE

Limitations of Non-networks

Examination of network processing is prefaced by a brief discussion of the limitations of processing which does not handle network data bases (i.e., processing for linear or tree structures only). The limitations are examined primarily in terms of 1) amenability to the formation and maintenance of an integrated data base, and 2) facility of the user - data base - application interface. Therefore, the methods for manipulating the data of a given type of data base are very important. A language for performing such manipulation is termed a Data Manipulation Language (DML). Recall that the language for defining data structures is called a Data Description Language (DDL). These concepts are illustrated in the following two examples of primitive data management.

The Fortran language has the capacity for manipulation of data in the form of linear structures. As such, FORTRAN commands can be viewed as a DML that allows the manipulation of data which is organized into the structure defined in FORTRAN declaratives. In effect, these declaratives form a DDL that allows linear structures. A simplistic data management system can be built by writing a collection of FORTRAN programs which manipulate data residing in a group of sequential files. Typically, these programs perform sorts, merges, and extractions of data from various files, and with subsequent computation, display it in some reports. Every time
a new kind of report is needed, new programs are written. Such a system may well suffice in situations where the number of data item types is small, the inherent relationships among data items are simple, the kinds of reports needed are static, and the users have the skill and time to write programs. As a second example, the Data Division of a COBOL program can serve as a DDL and the commands of the Procedure Division serve as a DML. This DDL admits the use of tree structure and the DML provides enhanced data manipulation facilities relative to FORTRAN commands. The standard COBOL language, then, can provide the basis for a rudimentary data management system similar to that described for FORTRAN, but more flexible. See [73] for the proposal of a more sophisticated tree-base data management system.

We first of all observe that in systems organized in this fashion, the user serves as the interface between the database and application programs that use the data; the tasks of fitting data to programs and of writing programs compatible with existing data structure is a human one. And these tasks become very cumbersome as the database tends towards increasing integration. The term integration is used to designate a condition wherein the database is capable of being accessed by users with different data needs, such that a given user has some data needs in common with some of the other users; but the total information needs of one user are not identical to those of another. One might say that there is a "many-to-many" relationship between users and data item types.

Thus it can be seen why the task of being an interface between an integrated database and the programs which access it is cumbersome. For
the tendency toward integration implies more data item types, more data values, more complex relationships among data items, and more complex data manipulations in order to obtain the necessary information for report production. In environments where trees are the only data structures available, this circumstance of more volume and more complexity must necessarily result in either a proliferation of the number of disjoint trees or an inordinate increase in the complexity of record types in a smaller number of trees. Both approaches entail significant degrees of redundancy with its attendant difficulties of inefficiencies and the maintenance of internal consistency. This volume and complexity has a decidedly negative effect upon performance of the human interface. The reason for the magnitude of disjoint trees and/or the inner complexity of record types is a direct consequence of the lack of a mechanism to handle all data item relationships within the confines of a single data structure. The network data structure provides such a mechanism.

Prior to examining the capacities of a network data structure, mention must be made of a common method for ameliorating the user - data base - application interface that is to some extent independent of the variety of data base structure employed. This involves the replacement of the human interface between applications and data base, with software. Software accepts user requests (which may be more or less procedural) and responds by interfacing the appropriate data with an application routine from a library of applications; results of the execution are returned to the user. While such 'self-contained' systems alleviate many onerous details from the user's concerns, their major drawback is a typically
very limited degree of flexibility with regard to the integration of additional data item types into the existing data base and the consequent introduction of additional application routines for processing that data. The basic difficulty is that such systems are designed to support a special, limited managerial function. The result is the existence of several self-contained systems (e.g., one for MRP, one for PERT/CPM, one for accounting, etc.) each of which has a data base that contains some of the same data item types as the other systems' data bases; once again, there is the problem of data consistency within an organization.

DML Interface with a Network Data Base

Given a DDL which permits the definition of logical network data structures, a DML is required which allows the storage, modification and extraction of data values for a particular structure. Specifications for such a language have been proposed by the CODASYL Data Base Task Group [27]. One implementation partially based upon these specifications is the GLPLAN (Generalized Planning System) Data Management System [12], [20], [47]. This DML is utilized within the framework of a host language, e.g., FORTRAN, COBOL. Each command is the DML consists of a FORTRAN subroutine call. Thus, these DML subroutines essentially extend the FORTRAN language to give it complete data manipulation capability with respect to data organized into network structures. An important feature of this method of implementation is its fairly high degree of machine independence; i.e., subject to a few minor modifications, this DML can be used on any machine that has a FORTRAN compiler and some sort of random access mass storage facility. One special group of these DML commands allows a user to perform
structural manipulations on an already existing data base (e.g., the addition or deletion of item types, record types or sets) [20].

In addition to these special commands, the DML consists of approximately seventy commands which perform the following kinds of functions: 1) opening and closing of the data bases; 2) creation and deletion of record occurrences; 3) setting of currency indicators; 4) addition and removal of record occurrences from sets; 5) retrieval and storage of data from current record occurrences; and 6) searching through sets for particular record occurrences. Ten to twelve of these commands are all that are required for most applications. The others are used for infrequent special applications. The strategy has been to develop a relatively large number of simple commands in contrast to the CODASYL approach of a relatively few commands which are of comparatively complex syntax. The typical GPLAN DML command has three arguments and no clauses.

As an aid to dealing with data base creation and the volatility of data values another general and high level DML command may be used. If we observe several utility routines concerned with loading data for various data bases, we find that the same basic steps (i.e., DML commands) are repetitively used in each. In order to obviate this repetition and make the activity of loading data as effortless as possible, we introduce a generalized load command.

The typical loading process takes data values from a sequential file of some degree of complexity and inserts them into a data base having a network structure. The procedure for taking one network structure into another is discussed in [22]. Factors involved in transforming one data
base to another are exhibited in Figure 7. DDL_1 is the schema of the source data base and DDL_2 is that of the target data base. DB_1 represents the source data values, organized according to DDL_1. The target data values, organized according to DDL_2, are indicated by DB_2. A non-procedural map must be supplied by the data base administrator to specify which data values of DB_1 are to be mapped into DB_2 (for loading or change or deletion) and identifies the data item type in DDL_2 with which each of these values is associated. The map may also specify (implicitly or explicitly) alterations that are to be made in DDL_2 (via the dynamic restructuring commands) and/or DB_1 if needed. The generalized loader proposed in Figure 7 is a routine consisting of DML commands referenced within the host language context; and may itself be viewed as a higher level DML command which can be invoked by a host language program. The most prominent feature of the generalized loader is its ability to traverse a network according to a non-procedural map. In this respect the loader, which (given a map) explodes sequential files into networks, is related to the CPLAN query system (discussed in the next section) which draws sequential files from network structures in response to a query requesting retrieval or program execution.

In summary, for large and integrated data bases a data manipulation language that supports network data structures is superior to one which does not; for the user is no longer required to keep track of a collection of trees (with its previously mentioned limitations), but has a compact and organized method for utilizing data from a single repository. Moreover, in such a situation, the user does not need to know the entire
Figure 7. Data Base Transformation
structure if only a small portion of the network is of personal concern. Each user may be allowed to have a simplified view (called a subschema) of the entire logical structure \[20], [22]. Note that in order to access the data or interface it with pre-existing applications routines, the user is still required to write programs using DML (even though the number or size of programs may be less, due to the power of DML commands that operate on a network). So with respect to the ease of interface, the network-based DML is inferior to specialized, self-contained data management systems. The objective, therefore, is a system for manipulating data which provides a facile interface for the user; but the system must also furnish a generality that allows it to be used in a wide variety of managerial contexts, including the situation where decisions are to be predicted upon 1) an integrated (rather than compartmentalized) data base, and/or 2) rapidly changing information needs. This has been the motivation behind the development of an English-like, non-procedural, general query system capable of interrogating network data structures.

**Query Interface with a Network Data Base**

The query language (GPLAN/QS) \[13],[16] discussed here extricates users from the position of interfacing programs with a data base; the user is not required to be a programmer. The query system provides the appropriate interface in response to non-procedural requests by the user. That is, the user's query needs merely to specify what is to be done; there is no statement of the procedures to be followed in order to accomplish the task. The query system may be used in either interactive or batch mode. In batch mode, the queries are entered on cards. In interactive
mode, queries are entered at a terminal and output is given at the terminal or at a high-speed line printer, according to the user's needs. CPIAN/QS allows both selective and unconditional retrieval. The query syntax is:

\[ \text{<COMMAND> <FIND clause> <CONDITIONAL clause>} \]

or alternatively,

\[ \text{<CONDITIONAL clause> <COMMAND> <FIND clause>}. \]

The COMMAND denotes which application program the user desires to have executed. The FIND clause specifies what data are to be retrieved from the database for use by the application program. This retrieval is subject to all conditions satisfied in the CONDITION clause. Any combination of data items in the database may appear in the CONDITION clause. The query system allows the use of synonyms and noise words in the query language. This feature makes a query easier and more natural to use; it also enables interested observers to understand the meaning of the query. Typical queries are:

\begin{verbatim}
LIST REACH.NAME, REAERATION.PARAMETER AND REAERATION.EXponent FOR
DATE = 11-1-75 AND REACH.LENGTH < .9
PLOT DATE VERSUS BOD.CONCENTRATION, AMMONIA.CONCENTRATION AND DO.
CONCENTRATION FOR REACH.NUMBER = 53
STAT NITRO.DEOXIDATION WHEN TEMPERATURE > 22
\end{verbatim}

A user of the query language is allowed to present arbitrarily complex retrieval clauses. Not only may this clause contain the names of data items to be retrieved, but arithmetic operations (using literals or data items) and both single and multivariate functions may also be in-
roduced. The conditional clause is composed of a Boolean expression which may contain data item names, literals, arithmetic operators, relational operators, logical operators, single-variable functions and multivariate functions. The query language also permits the use of noise words, synonyms and various other cosmetic features for the convenience of the user.

Perhaps the most outstanding feature of the GPLAN query language, aside from its ease of use, is its capacity to support queries directed not only at tree structured data bases, but at network structured data bases as well. The initial version of the query system was able to handle tree structured data bases only. In the process of implementing various information systems within GPLAN framework, it became apparent that in order to express all needed interrelationships among data items, a network was sometimes needed. Since the GPLAN data management system was implemented using doubly linked lists to represent sets, the extension to complete networks was entirely compatible with the existing DML commands. Though the problems of navigation through a network structure were more complex for the data base administrator and other persons desiring to use DML commands, the critical problem was that the initial algorithm used by the query language to determine a path through the data base was designed strictly for a tree structure. Development of a new algorithm led to the extension of the query system to its current capacity for searching through networks.

As the question of querying a network data base was being considered, it became apparent that the work being done on relational data bases [29]
by Codd et. al. might be of some use in resolving this problem. Careful analysis indicated that an efficient method of implementing a relational data base capability would be through the use of a network structure. A paper by Bonczek, Haseman, and Whinston [16] discusses how such an implementation has been performed. It is of interest that there is a trend towards utilizing networks in artificial intelligence to organize predicate calculus knowledge representations [64]. In addition, proposed implementations of the relational data base idea [53],[81] involve the use of networks.

The GPILAN query system implementation for network differs in two major ways from the original tree implementation. Whereas a tree structure possessed only one entry point, the root node, from which to begin a search for data items, the network has multiple entry points. This was described in the preceding chapter. A second and related distinction is that there exist many more paths to answer a given query in a network, than exist in a tree structure.

Further investigation of the relational approach resulted in the development of an extension to the existing query system that allowed querying of network structures and which was shown to be relationally complete [14]. The outcome of this excursion into the relational data base area was the demonstration that the GPILAN query language does indeed have all of the retrieval power of any of the languages proposed for relational data bases, and is more natural for the non-programming user while it operates on a network data base. This allows the user to view the data base as either a network or as a relational structure. In
addition, the GPLAN system is operational (not just theoretical) and does not become ineffective as the data volume becomes large. It should be noted that the approach of allowing the user to view the network data structure as a relational data base is consistent with the CODASYL view that the data base should be accessible by alternate classes of users with different languages, functions, and capacities.

A conceptual overview of the standard GPLAN system is portrayed in Figure 8. The library of application routines (right side of the diagram) is composed of two sections: standard routines and special routines. The standard library of applications consists of routines to generate reports and plots and to perform linear regressions, statistical analyses, and data modification. The library of special applications may include such routines as linear and non-linear optimization programs. The next section elaborates on the issue of interfacing such optimization routines with a network data base by means of the three methods alluded to in the introduction.

Methods of Program - Data Base Interface

The first two methods described here are presently fully available to GPLAN users; the third method is crucial to the design of the generalized decision support system presented in the chapter which follow. The first method examined here makes use of existing programs that have been devised independently of data base management considerations. Each such program requires that input data be in the particular format that it can use. Thus if the data that a linear program (LP) requires is stored according to a data base structure [19], then that data must be extracted from the
Figure 8. CRIAN System
data base and written onto a file in the format amenable to the LP routine. An obvious way to accomplish this is to write a program that uses DML commands to find and extract pertinent data from the data base. Output statements of the host language are used to transfer extracted data to a sequential file that is formatted for use by the LP routine. This DML extraction routine can serve to interface the LP routine with any LP problem that resides in the data base.

A second method of application-data base interface makes use of DML commands within the mathematical programming routine proper. Thus, the entire formulation of the problem need not reside in the program's arrays; instead, DML is used to withdraw and return particular coefficients (or groups of coefficients) to the data base as needed by the algorithm. The standard DML can do little more than store, find and extract data values. But what is a pivot operation if not a type of data manipulation? It is therefore as a logical extension to the DML, that we introduce a group of extended DML commands applicable to mathematical programming. In so doing, this second method of application-data base interface is greatly enhanced, for the programmer can issue DML commands to perform such tasks as pivoting, finding inverses and determinants, and even the solution of a linear system. It should be noted that programs using this interface method are, of course, amenable to integration into the special application library of the query system. The issue of automatic interface of programs with a data base is examined in [46].

The third method for interfacing applications with the data base consists of treating applications as part of the data base, i.e., application routines are accounted for in the logical structure of the data base.
The details of this technique are presented later in connection with the generalized intelligent decision support system. The major attributes of this approach are that it 1) allows the manipulation of program modules using the same procedures that are used for more traditional data; 2) permits a program module to be used on either a stand-alone basis or in combination with other modules; and 3) provides a method for the representation of abstract knowledge about the modules.

Implementation of Section 208 Decision Support System

This section outlines requirements for an implementation of a computerized decision support system which addresses the technical aspects of area-wide water quality planning. Section 208 of the Federal Water Pollution Control Act Amendments of 1972 calls for area-wide implementation of technical and management planning, with the objectives of meeting 1983 water quality goals and establishing a plan for municipal and industrial facilities construction over a twenty year period. Emphasis is placed on locally controlled planning, on dealing with non-point sources as well as point sources, and on consideration of both structural and non-structural control methods. The scope of the present examination is limited to those aspects of technical planning which are amenable to implementation within the framework of a computerized decision support system.

The Section 208 Problem

In order to be designated under section 208, an area must be in need of a complex control program and must exhibit either impairment (water quality limited segments) or a need for preclusion of desired uses. An
area-wide plan is constrained in that it must conform to the basin-wide plan, must take existing facilities plans into consideration, and must be capable of implementation from the managerial perspective. The plan may eventually act as a constraint on the issuance and conditioning of N.P.D.E.S. permits [44]. The area-wide objectives of technical planning are to identify the priorities of water quality problems in the area; recognize constraints in methods for dealing with these problems; formulate alternatives for satisfaction of water quality goals; develop cost data for each alternative; select the least-cost alternative that is feasible, given existing regulatory authority and qualitative restrictions; and periodically update the plan.

In order to meet these objectives, the planner must be able to store, manipulate, and retrieve large volumes of data. It is important that the data be organized in a manner that accounts for their intricate inter-relationships. The planner must be able to utilize a collection of pertinent application programs which provide selective retrieval of a multitude of data configurations; which offer statistical analyses, plots and projections; and which perform large scale simulations and optimizations. Each of these programs requires a particular configuration of data as input. In a comparatively unstructured decision making process, the types of reports that are needed tend to change rapidly; in traditional file-oriented systems this necessitates the writing of a new report generator every time a new report type is called for. The typical local planner is not a computer programmer; indeed, the planner's time is presumably too valuable to be concerned with writing report generators, maintaining
data files, and interfacing large scale application programs with data files. It is therefore proposed that the planner be provided with a decision support system which automatically handles the tedious, cumbersome task of data management and program interface, which provides organized data storage capable of representing all pertinent data relationships while obviating redundancy, and which can be controlled by the planner through the use of a non-procedural, English-like query language. The Generalized Planning System (GPLAN), as described earlier in this chapter, provides the framework for the implementation of such a decision support system.

Prior to examining the considerations involved in implementation, it is useful to enumerate those capabilities which are useful for technical planning and the resources required to support those capabilities. In order to assist area-wide water quality planners, a decision support system should include the following capacities:

1) Selective retrieval of any configuration of stored data values

2) Prediction of water quality for all area reaches, given a particular area-wide treatment strategy for both point and non-point sources

3) Generation of alternative strategies that meet a specified set of water quality goals

4) Production of waste load projections based upon land use plans, population statistics and employment statistics

5) Prediction of the effects of proposed land use plans on water quality

6) Determination of least cost alternative among those generated in 3) above

7) Permit basin-wide simulations in order to determine the effects of area-wide plans on each other

8) Permit regional monitoring of local planning activities
Such a system requires data of the following types:

1) Basin and area descriptions
2) Quality goals
3) Point source and non-point source information
4) Cost information for the various types of treatment
5) Data on land use plans, population, and employment
6) Municipal facilities information (including design flow data)

Major application programs which the system utilizes are simulations to predict water quality (given a treatment strategy for point and non-point sources), models to predict waste loads, models to predict the effects of alternative land use plans on water quality, and optimizations to select a least cost alternative from a set of alternatives.

Water Quality Simulation

Water quality simulation predicts the values of certain water quality characteristics that will result from adoption of a particular treatment strategy in a particular portion of a stream, river, or basin. There exist several models which can perform such simulations; these include DOSAG, DOSCAL, QUAL I, QUAL II, AND MULQAL. The basic core of data needed by each of these is the same, though there is variation in required data configurations. MULQAL [64],[65] is examined here, since it is the most comprehensive of the models, in the sense that it takes into account the effects of heat concentration on both the dissolved oxygen level and the rates of oxidation and reaeration.

Consider the MULQAL simulation of a given basin segment that is the subject of area-wide planning. In general, this segment may be described
in terms of its headwaters, its incremental flows and junctions, its pollution sources and their characteristics, its water treatment activities, and its various state characteristics. For purposes of analysis, the segment is viewed as being composed of an ordered set of reaches. The scope of a reach is defined such that certain state characteristics (e.g., reaeration parameters, velocity parameters, deoxidation rates) are relatively stable within that scope and there is no more than one 'significant' source of pollution in the reach. It should be noted that none of the models enumerated above directly handle non-point sources of pollution. Within the MULQAL framework, it may be possible to treat certain non-point sources as point sources with appropriate adjustment of reach scopes.

The segment, and therefore each reach within the segment, has water quality goals with respect to carbonaceous biochemical oxygen demand (BOD), nitrogeous BOD, heat concentration, various conservative mineral concentrations, and the measure of dissolved oxygen. A treatment strategy is a statement of the amounts and kinds of treatment associated with each reach. The water quality of a particular reach depends upon the amount and kind of pollution and treatment within that reach as well as the quality of the waters entering that reach (which is determined by pollution and treatment activities and assimilative capacities in upstream reaches). MULQAL permits the comparison of the quality implied by an area-wide treatment strategy with the water quality goals. MULQAL can be applied to any set of contiguous reaches, be it a small segment or a large basin.

MULQAL uses data of the following types: 1) Basin (or segment) structure description, including by-pass piping and junction configurations
and locations; 2) state characteristics of all river reaches; 3) specifications of all discharge activities, such as flow rate, temperature, and concentrations of BOD, ammonia and conservative minerals; 4) descriptions of water quality of all incremental flows; 5) characterization of headwaters in terms of BOD, dissolved oxygen, temperature and conservative minerals; and 6) a treatment strategy. The result of the MULQAL simulation is a description of the water quality of each reach in terms of BOD, temperature, conservative minerals, etc. along with the load allocation of each discharge point implied by polluter and treatment statistics.

The planner could use MULQAL iteratively, varying the treatment strategy with each iteration until a strategy is found in which water quality goals are met on every reach. MULQAL could also be used to discern the effect of a proposed new source of pollution (e.g., building a new industrial facility at a specific location) on the water quality of the segment as a whole. This would be accomplished by specifying the projected pollutants of the proposed outfall and defining a new reach to correspond to that outfall. In much the same way MULQAL can assist in predicting what the water quality will be at a future date, given a treatment strategy and projected discharge levels for that date. (Presumably the projected discharge flows would be derived from population and employment statistics by using models available for that purpose).

It is clear that there are many alternative area-wide treatment strategies which satisfy the area-wide water quality goals. The alternative which is selected will depend upon such considerations as cost, legal restrictions, implementation and managerial constraints, etc. MULQAL has
been interfaced with another large application package: NONLIN, which is a non-linear optimization model that determines that area-wide (or basin-wide) treatment strategy which has the smallest cost subject to the satisfaction of all water quality goals. NONLIN requires the collection of cost information regarding the various kinds and levels of treatment. So NONLIN may be useful where selections of a treatment strategy is based wholly or predominately on cost.

Implementation Considerations

Regardless of the type of system used, whether manual or CPLAN, data and models must be available. Given these, a decision support system can be designed. The perspective presented here is that this design is an ongoing process; the system must be adaptable, flexible and capable of handling volatility in the types of models used, in the types of data required and in the values of data occurrences.

The data and models are highly interdependent. There are physical, technical and monetary constraints on the collection of data, with regard to both data types and data detail. Data is usable according to the models which are available to analyze it. On the other hand the use of models is limited by the availability of data and to some extent by the complexity and size of the model and by the system available for supporting the model.

Given a local planning board's decision regarding the resolution level from which it intends to attack the area-wide water quality problem, it can be determined what data must be collected, what methodologies are to be used in collection, and what application programs are necessary.
Parenthetically it must be pointed out that the level of resolution may depend upon such factors as the technical expertise of the local planners, the availability of funds, the nature of existing data, and the characteristics of the existing (or anticipated) decision support system. Having settled managerial issues and hydrological methodology problems to some extent, attention can focus upon the development of the decision support system. Given data and models, these can be integrated into the GPLAN framework.

The initial step involves devising the appropriate logical data structure. Collected data, after passing through data validation routines, is then loaded into the database. At this juncture the user may query the data base, using any command in the standard library. MULQUAL, the Multiple Quality River Simulation model, has been interfaced into the GPLAN/QS as a special library. Thus the user may also request this simulation run, by typing a single command at the computer terminal. Other applications may be added to the special library in the ways already indicated. It should be noted that the addition of applications may be an ongoing process. For instance, if after using the system for a period of time the planners decide to augment the special library, the additional models can be interfaced with the GPLAN/QS in the one-shot operation already described; subsequent to this a new program may be executed by submitting a single query.

Another important area of system adaptability relates to changes in data. Basically there are two kinds of modifications. First there are additions, changes, and deletions of values of existing data item types
(i.e., those defined in the schema). Second there are modifications to the schema itself, which includes such contingencies as the addition of data item types, the addition of record types and the alteration of set relationships. There are two methods for handling the first variety of modification. The data base administrator can write a DML program to perform the necessary updating; this procedure is generally used when an update problem is relatively large in scale or is repetitive in nature. Small, non-repetitive modifications are best handled through the query language by usage of ADD, CHANGE and DELETE commands; these commands are currently available in the CPLAN framework. These two methods of revision provide the data base administrator with flexibility in treating the problem of keeping data values up to date.

In order to change the logical structure of an already existing data base there are two major approaches. The first involves destruction of the existing data base. The new schema is defined with DDL, the programs which load the data are revised to reflect the new schema, and the entire data base is loaded using this new logical structure and the revised load programs. For a large data base this is a cumbersome, time-consuming procedure and may necessitate the data base being dumped before reloading. The second approach, permits a dynamic restructuring of the schema without destruction of an exist data base and subsequent reloading. This is accomplished via six extended-DML commands (alluded to earlier in this chapter) which permit the addition and deletion of data item types, record types and sets. These commands (which were not proposed in the CODASYL report) automatically reorganize the data base so that it is con-
sistent with the revised logical structure. As a further aid to data base restructuring, the query language could conceivably be extended to utilize the capabilities of these extended-DML commands. As a result, the data base administrator would be able to accomplish dynamic restructuring by submission of simple English-like queries.

Another important factor to consider is the scope of the 208 system within the CPN framework. In this connection there are many possible alternatives, ranging from a collection of relatively small data bases (one data base for each area), to moderately sized data bases (one data base for all designated areas in a basin), to one or a few very large data bases (e.g., one data base for each EPA region). The topic of a distributed data base is considered in the final section of this chapter. Another dimension of scope involves the degree to which N.P.D.E.S., construction grant and land use information is integrated into the data base. A logical data base structure can be devised to permit basin-wide analyses in addition to area-wide (segment) analyses; it can include permit data and treatment plant construction information. Suppose that through the use of HULQAL and various cost and projection models, all area-wide planning agencies have arrived at respective local treatment strategies. Then regional administrators can monitor their activities by means of usage counters built in to the data base and by submission of queries that retrieve local plans. Furthermore, the water quality of the entire basin predicated upon current local strategies, can be simulated in response to a single query. This could permit the identification of the effects of various local strategies upon each other and upon the basin as a whole.
In this example there are two levels of data base entry: the segment level and the basin level. Typically, a group of area-wide planners is concerned with its own particular segment and maintains control over the data base information for that segment. The issue of restricting a user's access to certain portions of the data base in order that the user need not be confronted with data that is irrelevant to the local problem and so as to preserve data integrity, is an issue of data security. A method for treating this issue within the GPLAN framework is detailed in a paper by Cash et. al. [25].

Some examples of possible queries have already been presented. It is important to emphasize that GPLAN/QS allows retrieval of any configuration of data items using conditions on any combination of data items as the criteria of retrieval. Also GPLAN/QS is quite capable of accommodating special application programs in such a manner that submitting a query causes them to be executed, using any specified pertinent set of data from the data base.

Details of the 208 - GPLAN Implementation

Consider the four contexts of MULQAL usage: 1) Simulation of water quality throughout a basin, given a particular basin-wide treatment strategy; 2) Simulation of area-wide water quality for a particular treatment plan; 3) Prediction of the effect of a new pollution source on area-wide quality; and 4) Prediction of water quality within the segment over some planning horizon.

Current implementation permits a basin-wide execution of MULQAL. To extend the system to allow area-wide simulation, the MULQAL extraction
routine must be modified to allow the extraction of data from the database for a particular segment and the query command to run MULQAL must be adjusted to accept a segment number. This extraction is predicated upon a particular logical structure. Therefore, a change in the logical structure may necessitate a modification in the extraction routine. A decision must be made from the outset as to whether basin parameters (e.g., equilibrium temperature) can be used in the simulation of any segment within the basin; this will affect the DDL. Another issue to be considered in the construction of the logical structure is the relation between area and river since there can be more than one area in a river and more than one river per area. The latter occurs when an area encompasses the junction (s) of one or more rivers. These two considerations have definite implications for the logical structure and, therefore, for the extraction routine.

Upon resolution of these questions MULQAL can be used to simulate area-wide water quality in two ways. The distinction between the two methods lies in the manner in which an inflow from upstream is handled: 1) Treat the inflow as an incremental flow (a flow for which no control over quality or flow is possible) to the areas appropriate upstream boundary reach; 2) Simulate the quality of the entire basin upstream from the area in order to determine the quality of waters entering the area. The first method requires that actual measurements of inflow quality be taken and subsequently loaded into the data base. This has implications for the data base design and extraction routine, since such measurements must be ignored for basin-wide simulations. In essence the second method
simulates an area and the entire basin lying upstream, making use of the current treatment plans of all upstream areas in order to predict the quality of waters entering the area.

The third context requires the ability to restructure reaches within the data base. To determine the effect of a proposed pollution source whose anticipated effluent characteristics are known, we must first of all define a reach for this outfall (MULQAL permits only one significant polluter per reach). This is accomplished by dividing an existing reach into two parts, the upper part representing the existing outfall (or incremental flow or headwater) and the lower part representing the proposed source. The upper reach retains data of the existing reach except that the length of the reach is decreased appropriately. The lower reach has the same reach state parameters as the existing reach and it has data for the proposed polluter. Existing incremental flow, headwaters, polluter and treatment data are associated with the upper reach, not the lower. Having divided the reach in this way, MULQAL can be executed (using method 1 or 2) to ascertain the effect of the proposed pollution source upon the area's water quality. There are several technical considerations. First a distinction must be made, for purposes of MULQAL execution and data base integrity, between permanent and non-permanent reaches. Second, MULQAL requires that all reaches in a basin be uniquely identified and that the reach identifiers be ordered sequentially to reflect actual physical relationships. For instance reaches may be numbered 1, 2, 5, 9, 13, 18, ... (but not 1, 2, 9, 5, 18, 13, ...) to reflect upstream to downstream ordering.
In order to use MULQAL in this third context 'scratch' occurrences of a reach record type could be utilized. An alternative method is to utilize ADD, CHANGE, and DELETE query commands to temporarily alter the reach description and then restore it to its original state. Since GPIAN operates on a local copy of the data base, a third alternative uses the ADD and CHANGE query commands to alter this local copy; MULQAL is executed using this altered local copy to obtain the desired results. The user then ends the query session by typing ABORT, which fails to save the local data base; thus, the original data base is intact for the next interactive session.

As an example of the fourth context, suppose we want a prediction of area-wide water quality for five years hence, based on employment and population projections and land use plans. Using these as the basis, models would be used to predict pollution activities, which in turn are stored in the data base with the date for five years from the present. MULQAL can then be run using either of the two previously indicated methods. For the first method we would have to project the quality of the incoming incremental flow(s) in some way. The second method would require a uniformity of inter-area planning horizons; i.e., all upstream areas would have to have stored projected pollution data associated with the five year horizon date or else such data would have to be computed, stored, used for MULQAL execution and then deleted (to preserve data integrity) in response to a user's query.
Summary

This section of Chapter 3 has addressed factors involved in the development of a decision support system for area-wide water quality planning. Given a set of data and a special group of models for analyzing the data, these can be integrated into the GPLAN framework in order to provide timely, reliable and complete information; at the same time we recognize the need for a natural easy-to-use planner/system interface and the need for system flexibility that allows adaptation to changing conditions. The interface is provided by the GPLAN query system which allows the execution of large application models, as well as retrieval of any configuration of data (subject to any conditions), in response to simple English-like queries. System flexibility is enhanced both by the extended-DML commands which enable dynamic restructuring of the data base schema and by the generalized load program which loads data values from any sequential file into a pertinent network data base.

Information Transferral within a Distributed Data Base

In view of the rapidly expanding data base field and the frequently decentralized organizational environment within which a data base management system must function, the issue of distributed data bases is of topical concern. This final section of Chapter 3 presents a method for addressing the problem of distributed data bases. A data base is considered to be distributed if users in various locales are responsible for creating and maintaining their own portions of the data base and a user in one locale can access data maintained by a user in another locale. Such a situation
can arise on a variety of machine configurations (e.g., [5],[24],[35]). Although this section deals specifically with the problem of co-ordinating area-wide management of water resources, it should be noted that the method presented is generally applicable to settings of physical or managerial decentralization in both the private and public sectors. The perspective taken here is logical in nature; the question of resolving physical or hardware incompatibilities [4],[71] is not within the present scope. Resolution of incompatible data structures to allow transferral of data within a distributed data base will serve as our focal point. Briefly, the methodology being proposed involves the utilization of a non-procedural mapping language [22] to define a mapping of data values out of one network data base into another network data base without requiring user specification of any intermediate data structures. The user has merely to supply the map; neither pre-processing nor post-processing relating to normal forms is required. The mapping is able to handle special cases of the above situation such as linear list to tree, linear list to network, tree to network, network to tree, etc.

Varieties of Data Base Distribution

First of all, a taxonomy is presented for identifying the basic types of distribution which can arise. This taxonomy is developed within the context of congruent data base implementation among local data bases. An example of a local data base is the data base for a particular designated area of water quality management. An example of a global data base is the set of local data bases for all designated areas within a basin. The
local data bases may or may not be situated on the same hardware; as
previously noted word size or character representation code conversions are
not considered here. If on the implementation level, all local data
bases within the global scope utilize the same basic Data Description
Language (DDL) and the same basic Data Manipulation Language (DML) com-
mands, then we say that they are congruent. Where there are hardware
variations within the global context, utilization of a data management
system that is largely machine independent is required (e.g., [47]) there
may well be variations from one locale to another in certain non-essential
DDL features and in high-level or extended DML commands, but at some im-
plementational resolution level the DDL and DML are identical. Another
type of inter-locale disparity may occur in user views of the data manage-
ment system. Even though one local data base system utilizes implementation
features identical to those of another, the users of the first data base
system may have a very different conceptual view and may therefore utilize
different commands than the user of the second system [20].

The global data base is characterized according to two dimensions.
The first represents the relative degree of volatility in underlying
logical structures of the local data bases. The second is indicative of
the degree of uniformity of underlying logical structure among local data
bases. The degree of volatility ranges from static, wherein logical
structures of local data bases are not subject to change over some time
horizon, to dynamic, wherein there are frequent alterations of logical
structures over the time horizon. Uniformity refers to situations where
identical logical structures are defined for all local data bases; or at
least an appreciable subset of the logical structure of each local database is identical to a subset of the others. This framework yields the following four extreme cases:

I. Static structures and uniformity of structures.

II. Static structures and non-uniformity of structures.

III. Dynamic structures, but desire to maintain uniformity across all local structures.

IV. Dynamic structures with no uniformity.

Prior to describing how these cases can be handled by the previously mentioned mapping language and processor a cursory illustration of the mapping procedure is in order (for greater detail see [11]).

Generalized Mapping Language

The generalized mapping language has the following distinctive characteristics:

1) A linguistic formulation conducive to structural analysis and facile implementation of language constructs;

2) A high level of non-procedurality that renders the language convenient to non-programming users;

3) Generality in the sense of capacity to directly map information from one network data structure into another network data structure, thereby obviating the crude necessity of constructing intermediate linear data structures;

4) Flexibility in terms the ability to handle traditional functions of retrieval (typically, network data structure to linear data structure) and loading (typically, linear to network data structure); these are actually special cases of the third characteristic.

The mapping language has a context-sensitive grammar. Using Chomsky's concept of transformational grammars [26], the mapping processor applies
inverse transformations to statements in the mapping language in order
to arrive at corresponding expressions in a language derived from a context-
free grammar. An expression in this context-free language is compiled
using well-known methods of syntax-directed analysis [2]. The specific
implementation used, including the precedence tables, is described in [11].
Information from the compiled expression serves as input to network travers-
ral routines [13] which make the requested extractions from one data base
and insertions into another [22]. This is subject to any conditions
specified in the original mapping statement and conditions imposed by
the security system [25]. If a mapping statement is ambiguous (i.e. does
not identify a unique path in the source data base and a unique path in
the target data base) the mapping processor prompts the user for clarifica-
tions. All of these processes (ambiguous statement resolution, security,
etc.) can be characterized as inverse transformations on the original mapp-
ing command; thus the language processing techniques are sufficient for
processing the map.

The mapping processor utilizes a Data Manipulation language for the
purpose of interfacing with a data base. As previously noted, the DML
which has been implemented is in FORTRAN and so it is largely machine
independent. FORTRAN is also used for implementation of the mapping
processor proper. Since FORTRAN can serve as a host language for the DML,
the mapping processor can treat any DML command as a FORTRAN subroutine.
Thus the mapping processor is able to perform any needed manipulation of
a data base by using appropriate subroutine calls.
The mapping processor is maintained as a collection of overlays as indicated in Figure 9. The CONTROL overlay is responsible for initiating input, buffering internal data and calling the other overlays of the system. The PARSER is composed of several parts. The PREPROCESSOR overlay performs dictionary lookups for synonyms and noise words; as such it performs inverse transformations. The I. T. overlay performs the bulk of the inverse transformations (e.g. restructuring the mapping statement and identifying levels of retrieval when multivariate functions appear in the mapping statement). The SECURITY overlay also consists of inverse transformations based upon the security status of the user issuing the mapping statement. The COMPILER overlay performs the precedence parsing of the expression (context-free) that results from the inverse transformations, thereby producing an internal object code for evaluation in the execution routines.

The components of the PATH FINDER overlay find a path(s) in a network data base that corresponds to the map command. This path finding function can be considered to be an inverse transformation, which suggests that the path finding phase should precede execution of the COMPILER overlay. This is precisely what occurs and the compiler makes use of information from the path finder to order conditional clauses of the mapping statement in a heuristically optimal fashion. The INTERACTIVE module of the path finder is also responsible for resolution of ambiguities as they arise.

The EXECUTION routines consist primarily of two parallel sets of programs: one for retrieval of data from a source data base(s) and the
Figure 9. Structure of the Mapping Processor
other for storing information into the target(s). As data are fetched, they are tested for satisfaction of all conditions in the compiled map.

Data which meet all conditions are stored into the appropriate path of the target(s). The other portion of EXECUTION is labeled BLACK BOX. This is not currently implemented; its purpose is to perform necessary encoding and decoding of data that is to be transported to a data base with a physical structure that differs from that of the source data base. Issues such as word length and character codes must be considered in this type of operation.

Mapping problems may be partitioned into two categories. These are data base compaction and data base expansion. Figures 10 and 11 present simple examples of these two cases. The mapping problem is that of specifying the conditions under which data values organized according to one logical structure (source data base) are to be inserted into the form of another logical structure (target data base). Since defining the logical structures is not strictly a part of the mapping process, throughout the ensuing discussion all logical structures are assumed to be previously defined. As can be observed from Figures 10 and 11 the procedures of data base compaction and data base expansion are of a complementary nature. The most familiar variety of compaction is data retrieval which extracts data values from a source data base into a linear target which is used for display or as input to application routines. The most familiar variety of expansion is simple data base loading which 'explodes' data values from a linear source into a network target. Thus the complementary functions of the previously described generalized loader and
Figure 10. Data Compaction

a. SOURCE data base

b. TARGET data base
the previously described query system are incorporated into the mapping processor. The compaction map for Figure 10 is given by:

Taking SOURCE to TARGET, map STATE-NAME to SNAM, POPULATION to POP, LENGTH to LENGTH, RIVER-NAME to RNAM, and BASIN-NAME to BNAM relating S1 to S2 with T1, and S3, S4, and S5 with T2.

The expansion map for Figure 11 is:

Taking INITIAL to FINAL, map STATE-NAME to STATE-NAME, POPULATION to POPULATION, RIVER-NAME to RIVER-NAME, LENGTH to LENGTH, BASIN-NAME to BASIN-NAME, relating II to F1, I2 to F2 and F3.

An important special case of network structure is a linear or strictly tree-like data structure. In this case, there is only one possible relationship between two record types; therefore no relationships need to be specified in order to execute a map. In a full network a multitude of relationships may exist between record types; inclusion of relationship correspondences in the map command prevents ambiguities that arise in their absence. An alternative to user specification of relationship correspondences is to allow the mapping processor to prompt the user in order to elicit this information. Finally, it should be noted that conditional restrictions may be included in a map command in order to limit the data values which are subject to the map; this is useful for security purposes as well as checking for errors in the data. For example the compaction map for Figure 10 could be restricted by adding the clause --

...for 1,000,000 < POPULATION < 18,000,000 and 0 < LENGTH < 375.
Information Transferral

We first of all examine how the generalized mapping procedure is used to handle case I of static and uniform structures within a distributed data base. Suppose the user of data base A requires information from data base B. Since there is uniformity of structure user A needs only to match like names in the mapping statement, specifying B as the source and A' as the target. A' is probably a scratch data base for temporary use, though the data values could be mapped directly into A. Having the data in this form at A, it is immediately usable by applications set up to handle analyses of data base A. For cosmetic purposes it may be desirable to define another command which does not require redundancy; this is a special case of the MAP command. Note that MAP is also quite capable of handling different names for the same attribute within the context of uniform structures; if the query system already permits definition of synonyms, accomplishing the map in this manner adds unnecessary complexity.

In case II, user A is allowed to view data base B as being uniform with data base A. Since the data bases A and B are static the data base administrator can devise and store a map of B into the terms and structure used by A, as a one-shot operation. If this is done then the user can proceed as in case I. An alternative method, which is more cumbersome for the user is the performance of a single direct map; this requires that the user be knowledgeable in the particulars of other data bases than his own.
Suppose that in case III changes are made to local data bases on a weekly basis. Since the changes are to be made uniformly across all data bases the data base administrator needs to prepare only a single mapping command which for any locality will map data base $X_t$ into $X_{t+1}$. This command is used simultaneously at all localities and thereby preserves global uniformity. Data transferral among localities can then proceed as in case I.

For case IV a number of maps are stored in the data base to reflect the relationships shown in Figure 12. We first of all note that Map 1, which takes data base B into a form that is uniform with data base A, also contains all information necessary to accomplish the reverse procedure. Also observe that maps are stored only if one area is permitted to access another (e.g., there is no direct access between area A and area D). This case is handled just like case II except there must be a continuing update of stored maps. Notice that in any instance where one locale makes a change in its own data structure, no reprogramming is required in the applications at other locales. At most, all that is needed is a modification in the appropriate stored map. The instigator of the change is responsible for modification of the pertinent stored maps; such modifications may be effected automatically, since all information needed for the modifications is present in the locale's specification of the change to its own data structure.
Figure 12. Maps for Case IV of Data Base Distribution
Additional Considerations

Maps stored for the purpose of providing a user with a uniform view of all data bases, can also be used as security mechanisms. Data items or particular paths of a given local data base that are to be protected from outside scrutiny may be precluded from all (or some) stored maps that are available to users in other locales. Since maps may contain conditional clauses, conditional security [25] may also be accommodated. Not only are certain large portions of the data base protected, but on a more detailed level, access of a range of data values for a particular data item type may be effectively restricted.

Thus far we have considered the specification of a map that takes data from a single source into a target data base. At first glance, the situation wherein data is to be gathered from multiple sources into a target may seem to be quite complex. However, by virtue of the utilization of stored maps which effectively provide uniformity of data structures, the data transferral is fairly straightforward. This is particularly true in the instance where the values of the same data item types are drawn from all of the sources. For example, a target data base may be constructed from all (or some, if the map is conditional) values of the data item type POLLUTER-NAME that exist in some specified source data bases:

taking AREA-1, AREA-2, AREA-3 to AREA-4 map

POLLUTER-NAME to POLLUTER-NAME...

In order to obtain a target where different data item types are drawn from various sources a series of maps of this variety may be needed.
When the types of data stored in data base A (e.g., water quality data) are incommensurable with those stored in data base B (e.g., land use data) then the user (or Data Base Administrator) must define the target data base. This is not required (though it is allowed) for the previously examined cases, where the existing local logical structure furnishes an implicit target that is usable by existing application routines. When data bases are incommensurable then the source structure (or a view of it according to an existing map) must be known, the target must be defined and a map is written accordingly.

The preceding discussions presuppose that the user knows the location of data to be retrieved. However, the uniformed or non-technical user may be unaware of the data base(s) which contain the information required; it could even occur that such a user is unaware of the logical distribution of the global data base. Such a state of affairs is not uncommon in situations where users are planners and managers who typically are not expected to be experts in data base technicalities.

The solution depends upon the volatility of the types of information the user needs from other data bases. If the same type of data is repeatedly required from the same sources then the appropriate map may be stored and executed in response to the pertinent user query. For instance in order to simulate water quality, the simulation package requires certain types of data from other locales. The types of data required are invariant, though the data values undergo frequent modification. A user query requesting execution of this application, invokes an execution of the pertinent stored maps prior to performance of the desired simulation; and the entire mapping process is invisible to the user.
If the types of information requested by the user are subject to frequent change then the query system must be devised in such a way as to elicit the information needed to construct the required map. This involves interactively prompting the user.

A General Data Structure for Mapping Support

A crucial aspect of successful implementation is the existence of an organized, comprehensive and compact means for representing information about various user views and the maps required to support those views. The term deep data structure is used to refer to the description of a locale's data structure according to which its data values are actually organized. Surface data structure refers to a particular user view of a locale's data structure. There is one deep data structure per locale. Recall that in the foregoing discussion, the mapping feature has been used in two ways, namely:

I) It serves as a means for specifying the sources, targets and conditions of data transferral. As such, it may be
   A) defined by the user at the time of transferral, or
   B) stored if it is subject to frequent use.

II) The mapping facility provides a method for supporting a variety of surface data structures (each subject to its own security constraints) for a single locale's deep data structure; i.e., a map may serve as a transformation that takes a deep data structure into many surface views. A map is stored for each of the available surface data structures based on a particular deep data structure. There may be
A) many surface data structures that can be used at one locale with reference to its own data base,

B) many surface data structures used at many locales with references to a particular locale's data base, and

C) a given surface data structure may be shared among some locales.

Figure 13 illustrates a data structure capable of supporting both of these methods of utilizing the mapping feature; some version of this structure is incorporated into all local data bases. Furthermore, the depicted structure:

1) allows the acquisition (subject to security constraints) of a full description of the logical structure of any deep or surface view;

2) provides a dictionary facility for detailed description of the meaning of each data item in each surface or deep structure;

3) provides a mechanism for specification of access privileges for all data item types (a particular locale's data retrieval ability is constrained by the views which are available to it);

4) can accommodate information about the time of the most recent update of a particular map;

5) can serve as a basis for monitoring the most recent update to occurrences of a particular data item existing in a particular user view.

Within the data structure of Figure 13b, all deep and surface structures are represented as networks. Since the relational data base view has been shown to be equivalent to the network view [15], an appropriate
a. Data Structure for Representation of Maps

b. Data Structure for Representation of a Network

Figure 13. General Data Structure for the Support of Data Transferral Within a Logically Distributed Data Base
network surface structure is stored should a relational view be desired; translation of this structure to give the appearance of a relational data base is an exercise in cosmetics. From the figure, we see that each VIEW is DEFINED in terms of its RECORD-TYPES. Each RECORD-TYPE is COMPOSED of zero, one or many DATA-ITEMs; and a pair of RECORD-TYPES may be associated by declaring one to be the MEMBER and the other to be the OWNER of a SET. This information is helpful when a user is devising a map from one view (or several views) into another, since it provides a complete description of the structure of each view as well as a dictionary facility for ascertaining the meaning of any data item.

The structure depicted in Figure 13a is utilized to store all maps (see uses I-II and II outlined above). First of all notice that a VIEW may be either deep or surface. Moreover, for a given LOCALE, there may be many VIEWS and a particular VIEW may be used in many LOCALES; a LOCALE is associated with a VIEW by means of the LOCALE-VIEW record type. There may be many TARGET-VIEWS for each MAP, or there may be many SOURCE-VIEWS. Associated with each MAP are a series of ITEM-MATCHES and a series of RELATIONSHIPS, each of which relates a series of SOURCE-SETS with a series of TARGET-SETS. This general structure is capable of simplification for specialized purposes, as well as elaborations for further refinement.

A Scenario and Summary

The discussion of distributed data bases is concluded with a brief scenario that summarizes the procedure of information transferral via a
generalized mapping language. Each locale has its own copy of query processor, which includes the mapping processor and is based on underlying DML. This supports all queries that exclusively access the local data base. When a request is made that requires a map (the map being either user supplied or drawn by the query processor from the local data base) that request is analyzed by the local mapping processor and undergoes the following conceptual steps. For each requested source, the mapping processor generates a query in compiled form. This query is dispatched to the source, requesting extraction of data. Upon arrival at the source, the query is executed (subject to some priority scheme and concurrency policy) by the query processor of the source locale. The result is the generation of an extraction file which can be accessed by the query processor of the target for purposes of display, insertion into a target network (as specified in the map), or execution of a local application routine. Further research is required for a determination of both the machine configuration(s) most amenable to utilization of the concepts outlined herein and the precise protocols to be observed in update of local data bases.

The term distributed data base has been used to connote a situation where control (i.e., creation and maintenance) of data values and data structure is distributed (without overlap) among users in various locales, but where access is general, subject to security constraints. The generalized mapping language was briefly described and shown to be capable of handling data transfer in the four extreme distributed data base environments. A generalized data structure for supporting the mapping
function is illustrated. The presented method accommodates a variety of user views of data base structure, is independent of whether the data base is geographically distributed or centralized, furnishes a straightforward security mechanism and provides a basis for treating the contingency of uniformed users who may even be unaware of the logical distribution.
CHAPTER 4

CONCEPTUAL FRAMEWORK FOR DECISION SUPPORT

Introduction

This chapter introduces the conceptual framework which underlies the development of a generalized intelligent decision support system. Specifically, the GENI (GENeralized Intelligent) query processor is constructed in the light of this framework, building with database management, artificial intelligence and linguistic techniques. In some respects this query processor may be viewed as an extension to (or elaboration of) the previously described GPLAN query processor. There are, however, some important fundamental differences between the two.

First of all the code of the GPLAN query processor is not invariant to application changes. This is because the application routines are treated as part of the query processor. Thus changes in the special application library involve alterations of the query processor. The GENI query processor is invariant to application changes.

Secondly, the GPLAN query processor is designed to utilize network data bases. And although networks furnish the most flexibility (and the most powerful abstract knowledge representation) of all the varieties of data base structures, they do have certain shortcomings from the artificial intelligence viewpoint [86]. There is, for instance, the problem of using the same notation to represent both types and tokens (i.e., occurrences). The GENI query processor is designed to utilize a multi-level network (i.e., the information base) which does not suffer from such deficiencies. Of course, the GPLAN processor could be modified to do the same.
Another, perhaps more basic, difference between the two processors is that the GENI processor contains certain problem solving elements. This stems from its use of programs and abstract programming knowledge which have been stored in the information base. Unlike the GPLAN query processor, the GENI processor is capable of suggesting alternative algorithms for problem solution. (GPLAN can do this only to the extent that special applications programs are devised to do so.) A related distinction between the two processors is the program-data base interface described in the previous chapter.

Rationale for Study of a Generalized Processor

It must be emphasized that, when referring to the GENI query processor, we are not considering query processing in the sense of mere data retrieval. We shall take the broader view that query processing includes the activity of problem solving. The query processor must be able to 'understand' the meaning or semantics of queries that ask for more than retrieval. The test for this capability to 'understand' is the query processor's ability to perform the intended problem solving activities. Recall that we are not here considering the ability to solve problems of a specific application area, but rather a single, invariant device for handling any kind of programmable model. The query processor's property of invariance refers to the absence of a need for any alteration in its code, even though the models which it utilizes and the types of analyses which are to be performed with them undergo modification. It is almost needless to point out that the processor is independent of the type and structure of data that are subject to retrieval and to being interfaced
with models. This issue has been treated at length elsewhere \([11],[15]\). It will not receive detailed attention here. There is yet another sort of independence which the processor allows, namely the independence of each model from the manner in which data is organized in the information base. This implies that structural changes in the information base do not affect the code of any models.

It is appropriate at this juncture to pause for a consideration of the advantages of such a generalized query processor vis-a-vis application-specific processors. Perhaps the most significant attributes of the generalized approach is the conceptual framework which it affords. This is particularly important from the pedagogical standpoint. Given a knowledge of the general system, the student is in a much better position to comprehend and derive specialized systems, than would be the case in the absence of such knowledge. Observe that possession of this knowledge furnishes a basis for systematic comparison, contrast and discourse regarding special cases. Thus even before we examine the issue of implementing a generalized intelligent query processor, we can reasonably contend that the development and exposition of the conceptual framework (which must inevitably underlie implementations) is a valuable contribution to the as yet rudimentary field of decision support systems.

Of course this is not meant to suggest that specialists be deterred from their lines of investigation and development. Indeed, the specialist is quick to point out that with respect to execution and storage, a specialized system is almost invariably more efficient than a general system. It must be remembered however, that the price for greater opera-
tional efficiency is the forfeit of some degree of flexibility. Where efficiency is paramount it may be simpler or less costly (more efficient?) to constrain an existing general system to conform with some standard of operational efficiency than to devise a special system from scratch. As a topic for research (and as a managerial strategy) there is a good deal to be said for concentrating on how to improve responses to needs over time, under changing conditions. In the long run this may very well be more fruitful than devotion to augmenting one's capabilities for finding solutions that are optimal at a given moment of time. This would imply a need for flexible systems that can easily adapt to what has been learned and to current needs. One is faced with two distinct courses of action: 1) investment deterioration or incapacity; or 2) investment in a system that is not optimal, but is capable of adapting (and perhaps improving) when conditions change.

In an environment where numerous, diverse specialized systems are derived from the general, there is an advantage of relative ease in understanding each since all have been based on the same general principles. This is contrasted with situations wherein one must learn several entirely distinct systems which are not derived from a common root. A further asset acquired from the study of a general system is its potential for engendering insights that allow what is known in one system to be translated to others.

Framework of Decision Making

Since a major objective is to emulate (in some degree) human cognitive processes, it is imperative to identify the fundamental functional
facets of decision making. Following this, we proceed to examine the extent to which (and methods by which) each of these functional abilities can be incorporated into an intelligent query processor for decision support. The framework of decision making is given in the form of postulates which identify seven functional facets of decision making (i.e. seven abilities needed in decision making).

The central postulate is that decision making has three facets which are basic in the sense that no one of the three can be expressed in terms of the other two. We shall call these facets I) Power, II) Perception, and III) Design. In other words, the ability to make a decision necessarily involves I) the exercise of some power, authority or directive force, II) the ability to collect information, and III) the ability to formulate models and plans. It is submitted that no one of these three is describable in terms of the other two.

The second postulate states that the existence of the three basic facets implies the existence of four additional facets, each of which may be described by a unique subset of the three basic facets. These four additional facets will be called IV) Adaptation, V) Valuation, and VII) Organization. Analysis may be described as a continuing adjustment between perceptions and formulations and it results in beliefs, expectations or 'facts'. Valuation is describable as a continuing adjustment between perceptions and powers. Values (ideals, standards, utilities, etc.) are based on available information and available powers. If there is no information about X, it is meaningless to speak of the value of X; similarly if there is no power with respect to X, then it is irrelevant.
to speak of the value of $X$. The facet of organization is expressible as a continuing adjustment between design and power, resulting in the imposition or execution of plans. Finally, the facet of adaptation may be described as a continuing adjustment among the other six facets. As such, it involves the activity of problem recognition which is constrained by the natures of the other six facets. We can consider adaptation to be a sequence of problem recognitions which, as a result of the activity of the other facets, tends toward a problem that is minimal; that is, the minimal problem consists of a single alternative.

The third postulate indicates that these seven facets are complete and minimal, in that all activity of a decision maker may be described in terms of them. If these postulates are accepted as accurately portraying the functional aspects of decision making, then they furnish a framework for an intelligent query processor for decision support. We say that the processor supports (rather than makes) decisions because some of the facets are not accounted for in the processor. For instance, the processor has no intrinsic power or authority; it has authority only in proportion to the weight which the decision maker attaches to its activities. For all facets, the processor can participate in the joint human-computer decision activity only to the extent that they are capable of formal expression. For example, the facet of valuation may involve some non-formalizable subjective processes.

An overview of those functions which have been incorporated into the processor design is given in Figure 14. The arrows roughly indicate flows of control (and information). Rectangles within the query processor do
Figure 14. Functional Facets of Decision Support incorporated into the GENI Query Processor
not indicate distinct components, but rather distinct functional activities which are operationally related to one another. Throughout this paper we shall discuss the query processor more on this functional level of detail, than on the level of structural components. Each function may be carried out through several components; but some components may participate in more than one functional activity \([58]\) and in so doing they establish the operational relationships among the various functions.

Observe that the query processor has two sources of information: the user and the information base. The human information processor also has two sources: external discourses and memory. The method for obtaining information from one source is different from that of the other. Information is obtained from the user by decoding the user's English-like queries into an internally comprehensible form which we shall call an intermediate structure. Information is located in the information base by finding appropriate paths through the semantic network. Problem recognition, lying at the core of the query processor, transforms the intermediate structure of a query into an explicit problem statement(s). If there is more than one problem statement several courses of action are possible; the one which we shall use involves interrogating the user for clarifying information. In the course of deriving an explicit statement of the problem a model is formulated. This model is a high-level representation of executable code. Analysis involves the fitting of data to the formulated code, in terms of handling the execution of that code with the proper data inputs.
Within the present scope of decision support the facets of power, valuation and organization remain within the domain of the decision maker. This is because the query processor, by itself, is essentially without power or authority; and as previously postulated, valuation and organization are closely related to the nature of available powers. This omission means that the processor's capability of problem recognition is not complete, for it involves only the interrelationships (or interactions) among the facets of perception, design and analysis. The extent to which the three omitted facets could be accounted for in a \textit{generalized} query processor remains an open question.

There is however an application specific system \textsuperscript{[68]} which addresses all of the facets. This system plays Go and may be characterized as a decision making system, in contrast to a decision support system. The Go system may therefore be considered to have power over decisions which are made. It has extensive perceptual abilities, in conjunction with a knowledge representation scheme. The system has model formulation capabilities in the sense of devising short and long range playing strategies from a set of available playing options. The authors indicate the utilization of heuristic evaluation procedures to resolve tradeoff questions about the moves to be made. The analysis facet involves fitting existing strategies to data about the current game situation in order to produce some expectations about the outcomes of the strategies. Decisions are implemented by updating the current game situation which is stored according to the knowledge representation scheme. Finally, the function of problem recognition involves the coordination of these other functions; it is activated
by the perception of a current game situation and proceeds to identify a set of semi-independent problems to be resolved by utilizing the other functions. It must be pointed out that this description of the GO system is very brief and does not mention many important issues and objectives relating to its development. The purpose here is merely to illustrate how it may be described in terms of the seven postulated facets.

Requirements for Decision Support

Although there are several facets involved in reaching decisions, we investigate three in particular: information access, model formulation, and analysis. The efficacy of a decision support system may be evaluated in terms of its flexibility, facility, scope, timeliness and cost in supporting these three facets. Further comment on a fourth facet, problem recognition, is deferred until Chapter 6.

With respect to information access there must be a mechanism for the systematic, integrated storage of all pertinent information. The information base outlined in the next chapter provides just such a mechanism, through both horizontal and vertical integration and through its capacity to relate operators with each other and with arguments. Given such a storage mechanism there must be a technique for interrogating (and modifying) it that can be used by decision makers who are not computer experts or programmers. The query language presented in the previous chapters is used for this purpose. Extensions to this language to take advantage of the full expressive power permitted by the information base are topics for future research.
The second facet which must be supported is the activity of model formulation. This facet refers both to models that are subsequently used for purposes of analysis and to models in the sense of plans to be implemented. This is a crucial aspect for resolving unstructured problems and for supporting the exploratory aspects of decision making. In short, the decision support system must have a component for the generation (and possible evaluation) of alternatives for achieving a stated goal. It will be shown that the information base contributes to such an end.

The decision support system must also provide for the activity of analysis; i.e. the fitting of data with models and models with data, thereby resulting in some expectation, beliefs or knowledge. Implicit in the very nature of the planning activity is the dynamic quality of the interface between model and data; for even though a collection of data may be comparatively stable over some time period, both the problems and the models used for problem solving may be subject to frequent alteration. Notice that a model operates on a particular subset of the entire collection of operands available, and it requires a certain configuration of this data as input. We contend that the tedious, cumbersome task of interfacing data and models for purposes of analysis should be automatically handled by the decision support system in response to the commands of a non-programming user. The method for accomplishing this is discussed in Chapter 6.
CHAPTER 5

THE INFORMATION BASE

Introduction

Within the scope of the remaining chapters a distinction is drawn between the terms information and data. Observe, first of all, that information is an abstraction; it is not something which can be pointed to or seen. However it may be conveyed by patterns of 'matter-energy', i.e. by configurations of symbols, by data. Data and information invariably accompany one another. The words on this piece of paper are not information, but rather a pattern of matter-energy which as a consequence of certain activities (e.g. inputting, transmitting, decoding, associating, storing, deciding, etc.) conveys information. The important point is the patterning of data; the 'richness' of a notation in terms of the kinds of data relationships which it can represent has obvious implications for its power in conveying information. With this in mind, we can note a pronounced trend in the history of information systems from the relatively impoverished linear data structure to the tree and network data structures capable of a greater variety of data configuration; correspondingly the ease with which comparatively complex information can be conveyed has also grown. Summarizing, "... we can say that data is an objective notation which has no significance in itself, versus information as a subjective concept which relates a datum to a context." [1].
In order to understand the varieties of contexts or configurations in which data must appear if there is to be a comprehensive conveyance of information, we examine the field of semantics. Of special interest is the notion of a semantic net. The results of this examination constitute a basis for the specification of information base features which permit the unambiguous representation of all types of information pertinent to decision support applications. This representation must configure data such that all significant relationships among parcels of information (e.g., among fact, procedures, empirical information, etc.) are accommodated. Furthermore, these objectives for information base features must be set in a manner that is amenable to processing for the purposes of inference and deduction.

Since semantics deals with the relationships between symbols and what they denote or mean [83], what we call the information base may be viewed as a semantic mechanism capable of representing meanings in terms of data configurations. Its storage technique must be general enough to handle the basic kinds of information involved in decision making regardless of the specific decision application. These types of information are: directive information, conceptual information, empirical information, stimulatory information, information about expectations, information concerning valuations, and procedural information. In addition, the information base must be flexible enough to represent the often intricate interrelationships among information parcels, relating them so as to capture their full meaning and impact with respect to other parcels of information. This latter point is particularly significant in that it
furnishes a basis for the synthesis of separate parcels of information that are all related to the same object, concept, observation, etc.

Woods [83] defines a semantic network to be an attempt to combine into a single mechanism both the ability to store factual knowledge and the ability to model associative connections which render certain parcels of information accessible from certain others. Moreover he indicates three criteria which must be satisfied by a notation used for semantic representation:

1. Logical adequacy. The notation must provide an exact, formal and unambiguous representation of any particular interpretation that may be given to a sentence.

2. There must be an algorithm for translating an initial sentence into this notation.

3. There must be algorithms capable of using the semantic representation in order to perform needed inferences and deductions.

The information base detailed in the subsequent discussion will be shown to satisfy the definition of a semantic network. The previously described query language, in conjunction with the information base, will be shown to satisfy the three requirements of notations for semantic representation.

**Information Base Features**

A specific design and implementation of the information base is described in a later section; this design and implementation is based in part upon the idea of a network data base advanced in the CODASYL DBTG Report of 1971 [27] and subject to extensions and modifications outlined in [20]. The term information base, rather than data base, is used to
emphasize its incorporation of two fundamental features which do not appear in the general data base management literature. Both features concern ways of patterning data that can convey information not commonly treated in the guise of data base management, but of value to decision makers; they furnish methods for introducing two novel kinds of context into data structures.

In observing the progression from linear structures to trees, to networks, we note increased facility for relating a datum with other data; there is an increased capacity for specifying the context of a datum in terms of data structures. Though there is little context inherent in linear data structures, the data content of groups of such structures may be used to represent trees. This becomes complex and cumbersome as the tree to be represented grows in size. Similarly, though it is possible to twist tree structures to the task of representing large or complex networks by using collections of tree-like structures, this cannot be accomplished in a facile, straightforward manner. This analogy may be continued with respect to the two features being introduced in this section. That is, they can in some sense be represented within network data structures, but such an approach leads to certain asymmetries (with respect to processing) and difficulties akin to those encountered when representing trees in linear structures. Since the two features are not inherent in the common notion of a network data base, the information base is introduced as a mechanism which encompasses both while allowing full network capabilities.

The first feature involves the introduction of the concept of resolution levels within the mechanism for information organization. A simple
example of this is described by Winograd [83]. Consider data about cars in which specific weights and colors are related (linked) to each car; on a higher level of resolution, we may want to somehow store information about what the properties of cars are. So on one level of resolution we are interested in specific attributes of specific cars and on another level we are concerned with properties of cars. Thus two distinctive characteristics of the information base are links which integrate individual information parcels on a given level of resolution into a single network structure and secondly, the integration of information of varying levels of resolution into a single structure. We term the former characteristic 'horizontal integration' and the latter 'vertical integration.' So horizontal refers to linkage of entities on the same level; whereas vertical denotes linkage among different levels via information parcels that participate in both levels (though the nature of participation is different on each level). Subsequent sections of this paper describe resolution levels in greater detail, providing examples and formal definitions.

The second outstanding feature of the information base involves its ability to handle the integration of programs into its logical structure. Not only does this permit the linkage of a datum with a program that uses it; it allows the construction of networks (in both the horizontal and vertical sense) of program modules. This capacity has two primary effects. First, it provides the basis for model formulation. Second, it furnishes a more comprehensive mechanism for semantic representation.

The aspect of model formulation involves the action of relating certain modules into a desired configuration. This necessitates a know-
ledge of which configurations are meaningful and which are not. Such knowledge is stored in the information base's semantic network. This approach has much in common with the notion of structured programming. Programs devised according to the tenets of structured programming [30] are readily amenable to storage within the information base; indeed there is also the ability to store alternative modules (e.g., alternative functional forms) for performing a particular role within the context of either other modules or a higher resolution level. The advantages of structured programming in terms of maintainability and extensibility [31] are also apparent in the strategy of integrating program modules into the logical structure of an information base. That is, it is possible to add, replace or delete a module in the same manner that one would add, replace or delete an occurrence of data.

It is useful at this juncture to point out a distinction between program modularity and program resolution. The idea of a resolution level also goes under the name of level of abstraction. Dijkstra [30] indicates that each level of a system's software hierarchy constitutes an abstract resource which participates in the next higher level and which has available to it the resources of lower levels. So "... at one level the programming amounts to manipulation of the abstract resources supported by the next lower level of the heirarchy. The programs at that level manipulate abstractions—the abstractions of the resource, whatever it may be—and at the same time participate in generating a higher level of abstraction for the next level of the heirarchy to manipulate" [57]. Furthermore, Miller and Lindamood suggest that a "... highly modular
implementation is one in which specific functions are performed by specific modules (and nowhere else); on the other hand, a system which preserves a hierarchy of abstract resources would appear to require modularity as a minimum, and perhaps a great deal more 'structure' [57]. Such a structure is effectively treated by the information base feature of resolution levels which allows the arrangement of program resources into levels of abstraction.

The second effect of allowing the integration of programs into the structure of the information base is the more comprehensive semantic representation that is permitted. Much literature about semantic networks is concerned with the network representation of English sentences (e.g. [50], [86]). These sentences consist of patterns of verbs and arguments. The typical decision maker who queries the information base requests the execution of some model (i.e. operators, verbs) using certain data (i.e. operands, arguments) as inputs. The usual data base structures handle information about arguments only; the meaningful operator contexts in which such arguments may appear is not represented in standard types of data base structures. A more detailed discussion and practical application of this feature of representing programs in an information base is presented later.

**Formalization of the Information base**

This section presents a formal description of what is meant by the term 'information base.' We define a record occurrence to be a uniquely labeled aggregate of data (i.e. string of symbols). Where \( I^+ \) is the set of positive integers, let \( X_0 \) be the set of labels associated with a finite
set of record occurrences, such that $X_0 \subset I^+$. A record type, uniquely denoted by the label $p_i$, may be described by a function $r_1$ as follows. Define $R_k$ as the set of all $r_1 : X_k \rightarrow \{0, 1\}$ such that:

1. $\forall x \in X_k$, $r_1(x) = \begin{cases} 1 & \text{if } x \text{ is of the type labeled } p_i \\ 0 & \text{otherwise} \end{cases}$

2. $\forall x \in X_k$, $\sum_{i=1}^{k} r_i(x) \leq 1$

3. $\forall r_1 \in R_k$, $\sum_{j=1}^{k} r_1(x_j) > 0$ where $X_k = \{x_j\}$

Property (1) states that $r_1$ defines the collection of $x \in X_k$ of the type labeled $p_i$. Property (2) indicates that each $x \in X_k$ can belong to at most one $p_i$. Property (3) states that each $r_1$ is non-trivial.

Before defining $X_k$ for $k > 0$, note that $P_k = \{p_i\}$ is the set of all labels associated with the elements of $R_k$. Since $X_0$ is finite, we can define these labels such that $P_k \subset I^+$, $P_k \cap X_0 = \emptyset$; furthermore we can define each of these sets of labels such that it has no elements in common with any other $P_k$.

Define:

$$X_1 = \left\{ p_i \in P_0 \mid \exists x \in X_0 : r_1(x) \neq 0 \right\} \cup X_0$$

$$X_2 = \left\{ p_i \in P_1 \mid \exists x \in X_1 : r_1(x) \neq 0 \right\} \cup X_1$$

$$\vdots$$

$$X_N = \left\{ p_i \in P_{N-1} \mid \exists x \in X_{N-1} : r_1(x) \neq 0 \right\} \cup X_{N-1}$$
It follows from the definition of $X_0$ and $R$ that there must exist a $K$ such that $X_K = X_{K+1} = \ldots$; then let $X = X_K$. Observe that $X$ is the set of labels of all record occurrences within an information base; these labels are unique identifiers, thereby serving as information base keys. All occurrences of a record type denoted by the label $p$ can be determined by successive applications of the function $r$ to the set $X$. The magnitude of $K$ indicates the levels of resolution inherent in the information base. The reader will notice that $P$ is always a subset of $X$; if it were not desired to treat all record types as record occurrences, one could define $X = X_{K-1}$. There are advantages to defining $X = X_K$, especially for purposes of altering the logical structure of an information base after it has been loaded. This will be elaborated in a subsequent section.

Continuing, we can now formally define the information-set (in-set). Let $Q_i = \left\{ x \in X \mid r_1(x) \neq 0 \right\}$. If a function associates each element of its domain with no more than one element of its range it is said to be a functional relation. Then each functional relation $f_1: Q_j \rightarrow Q_1$ uniquely defines an in-set of which the record type $r_i$ is said to be the owner and the record type $r_j$ is called the member. It is important to make several observations about the in-sets of an information base. It is permissible, and sometimes useful (Chapter 2) to allow $i = j$. Second, an in-set may be used to associate record types of different levels of resolution. Third, the set $F$ of in-sets of an information base must be carefully defined so that its elements are consistent; e.g., one should exercise caution in defining both $f_1: Q_1 \rightarrow Q_j$ and $f_2: Q_j \rightarrow Q_1$ as elements of $F$. Finally if $f_1: Q_j \rightarrow Q_1$ and $f_2: Q_1 \rightarrow Q_K$, then we can form the composite in-set $f_1 \circ f_2: Q_j \rightarrow Q_K$ defined by
\((f_1 \circ f_2)(x) = f_1(f_2(x)) \forall x \in Q_1\).

This is sometimes desirable from the standpoint of access efficiency; it also allows us to attach special significance or meaning to certain groups of sets.

The foregoing is a formal description of the major features of the information base. It accounts for both the horizontal integration (via in-sets) and vertical integration (via resolution levels) of information into a single mechanism. In order to illustrate the use of resolution levels, the above formalisms are applied to the problem (see Winograd [83]) of representing information about cars. In this problem cars are to be described in terms of color and weight; in addition suppose that we would like to denote that color and weight are properties. Assume that we have record occurrences as shown in Figure 15a; these are identified by the respective labels in \(X_0\). The set \(R_0\) is also shown; by inspection we see that \(R_0\) satisfies the needed conditions as given at the beginning of this section. The function \(r_1\) determines whether or not an element of \(X_0\) is of the type color. Similarly \(r_2\) is associated with the type weight and \(r_3\) is associated with the type car. In the proposed implementation each \(r_1\) defines (and is defined by) a linked list of occurrences of its type. Given \(X_0\), \(R_0\), and \(P_0\) we apply the rule for defining \(X_1\) to obtain the result shown in Figure 15b. \(R_1\) is also given and clearly satisfies the necessary conditions for its definition. Application of \(r_4\) to elements of \(X_1\) can be used to determine which elements are vehicle properties. Figure 15c gives the \(X_2\) that follows from the definition. If we take \(R_2 = \emptyset\), then \(X = X_2\).
$X_0 = \{1, 2, 4, 5, 15, 16\}$

$R_0 = \{r_1, r_2, r_3\}$ with labels $P_0 = \{8, 9, 11\}$

\[
\begin{align*}
    r_1(x) &= \begin{cases} 1 & x < 4 \\ 0 & \text{otherwise} \end{cases} \\
    r_2(x) &= \begin{cases} 1 & 4 \leq x \leq 7 \\ 0 & \text{otherwise} \end{cases} \\
    r_3(x) &= \begin{cases} 1 & x \geq 15 \\ 0 & \text{otherwise} \end{cases}
\end{align*}
\]

a. Initial Record Occurrences

$X_1 = \{1, 2, 4, 5, 15, 16, 8, 9, 11\}$

$R_1 = \{r_4\}$ with label $P_1 = \{14\}$

\[
\begin{align*}
    r_4 &= \begin{cases} 1 & 8 \leq x \leq 10 \\ 0 & \text{otherwise} \end{cases}
\end{align*}
\]

b. Generation of $X_1$

$X_2 = \{1, 2, 4, 5, 15, 16, 8, 9, 11, 14\}$

c. Generation of $X_2$

Figure 15. Resolution Levels for Representing Information about Cars
The occurrences and their 'vertical' relations with each other are diagrammed in Figure 16. Also depicted are two in-sets: \( f_1 \) and \( f_2 \).

Using the definitions of \( Q_1 \), \( Q_2 \), and \( Q_3 \) given in Figure 16, \( f_1;Q_3 \rightarrow Q_1 \) and \( f_2;Q_3 \rightarrow Q_2 \). The arrows in the diagram point from the owner of the in-set to the member; i.e., each arrow points in the direction opposite to that in the notation of its corresponding functional relation. Using the formalisms introduced here it is a simple matter to represent an extended problem including other kinds of vehicles [11], more properties, subclassifications of properties (e.g., structural, functional, etc.) and even properties of properties.

**Levels of Resolution**

Before proceeding to a water quality illustration, there are a few more remarks to be made about levels of resolution. A cursory survey of existing data management systems indicates that there are several types of logical data structures (Chapter 2) which are typically used to store empirical information (e.g., measurements, identifiers, etc.). Recall that of these, network data structures have the most far-reaching capability in terms of representing intricate interrelationships among information parcels where associative connections are combined into a single structure; i.e. the attributes of network data structure most nearly fulfill the definition of a semantic network. In passing, note that the recently popularized notion of 'relational' data structure [29] is directly analogous to the network data structure [15]. As an exercise, networks may be disguised with various cosmetics to give the 'relational' appearance, if one is more comfortable with such a veneer. Of course at
Let:

\[ Q_1 = \{ x \in X \mid r_1(x) \neq 0 \} \]
\[ Q_2 = \{ x \in X \mid r_2(x) \neq 0 \} \]
\[ Q_3 = \{ x \in X \mid r_3(x) \neq 0 \} \]

Figure 16. Occurrences in Car Information Base
the query level, neither the operators for network traversal nor the relational operators should be apparent.

Returning to the car example, weight and color exemplify (i.e., are instances of) that type of information which is called property. This is a semantic distinction inherent in the linguistic usage of these terms. They are therefore treated as record occurrences of the record type property. But observe also that red and blue exemplify the type of information which we call color. So they are treated as record occurrences of the record type color. This implies that color is treated as both a record type and a record occurrence. (But no provision is made for such a contingency within the standard CODASYL framework.) Figure 17 shows an example of record occurrences (ovals) organized according to the structure in Figure 16. Observe that the type of each occurrence is given in the margin to the right of that occurrence. The broken lines show the connections between levels. Color and weight are occurrences of PROPERTY on level 2; but they are record types on level 1.

As a footnote it must be pointed out that no great ingenuity is required to represent an information base structure (including resolution levels) in the notation of a network data base structure (the same may be said for representation of a network in terms of trees). Figure 18 presents a data base for cars in a strictly network notation. But this requires a utilization of the CODASYL "set" feature, so that in the data base structure two distinct methods are required to represent a single kind of relationship between two parcels of information (namely the relationship of one information parcel being exemplified by the other).
Figure 17. Car Information Base - Structure of Occurrences

Level 1

- Blue
- Red

Level 2

- Car-1
- Car-2
- Car-3

- 1-Ton
- 2-Ton

Set Record Type

Property

Weight

Color
Figure 16. Network Structure for the Car Data Base
This requirement is not inherent in the information to be stored; to the contrary it is forced upon the data base designer by limitations in the network notation. One must certainly question the desirability, reasonableness, and practicality of such a practice. It can cause certain problems for the type of query processing that ventures beyond mere retrieval.

Recall that the central concern of this thesis is a generalized intelligent query processor. This processor's value depends upon its interpretive ability, its capacity to systematically discern meanings of what it 'perceives'. Semantically, there is an important distinction between defining or exemplifying an entity (e.g. an object, a concept, etc.) on the one hand, and specifying a functional or structural relationship between that entity and other entities on the other hand. Examples of functional or structural relationships include possessing, containing, requiring, using, etc. It will become apparent later that the above distinction is particularly significant with respect to the formulation of models, where the entities may be thought of as concepts or models. Given some parcel of information and some concept \( X \) the query processor must be able to ascertain whether that parcel is an exemplification (i.e. one of a number of alternative definitions) of concept \( X \), or whether it is itself a distinct concept which bears some functional relationship with concept \( X \). The latter case is represented by the CODASYL 'set'. However, utilizing network data base notation one may be forced to represent the former case in two ways. This implies that a single construct, the 'set', is utilized to represent two seman-
tically different kinds of relationship. This presents a query processor with a dilemma of how to interpret a 'set'. The interpretation has implications for the nature of the processing that must follow it. This difficulty does not appear if we use the information base notation, in which the former case is treated only as occurrences of a record type and the latter case is represented with an in-set. This is illustrated in a subsequent section.

The information base feature of resolution levels has another advantage over the strictly network notation. This may be visualized by contrasting Figure 17 with Figure 19. The latter Figure depicts occurrences organized according to the logical network structure of Figure 18. Observe that both methods use four record types, and correspondingly they necessitate identical numbers of record occurrences. However the network method uses four sets, whereas the information base needs only two. These additional sets require the storage of additional pointers which are not needed in the information base, for resolution levels are not implemented with pointers; rather, record type and record occurrence information about an entity are stored in essentially the same location. Not only does this result in storage savings relative to the network method, but it also results in access that is no less efficient (by virtue of the smaller number of 'sets' to be traversed). These advantages become more important as the number of resolution levels increase.

An Information Base for Water Quality Management

More detailed discussions of the water quality management problem may be found in [21]. The objective of the example presented in this
Figure 19. Structure of Occurrences for Network Car Data Base
section is to demonstrate the applicability of the information base as a device for capturing the semantics used to support practical decision problems. At this point, it is presumed that the reader has a sufficient concept of what are information base entails to obviate the need for complete formalistic description. So for the sake of economy, the following example is presented in a less formal manner than the previous one. It will be used to depict certain implementational details (e.g., languages in which the information base is specified and with which it is utilized).

Consider the record type POLLUTER, displayed in Figure 20a. This aggregate of data item types represents measures of types of polluter activity for a given date. So occurrences of this record type correspond to measurements taken on various dates. In order to build a semantic network, we must indicate how this concept of POLLUTER fits into the pattern of knowledge concerning water quality management. A polluter is properly characterized as being a property of a river reach. Other properties of a reach include reach parameters, headwater, incremental flow, and treatment plan. So a reach is characterized in terms of these properties as follows: a reach is a portion of a river in which certain water quality parameters are relatively invariant; which has no more than one (point-source) polluter, one incremental flow or one headwater; and which must possess treatment plans. This could be represented in the information base by occurrences of the REACH PROPERTY record type displayed in Figure 20b. However, observe that each occurrence of the data item NAME (e.g. 'POLLUTER', 'HEADWATER', 'PARAM-
### a. Polluter Record Type

<table>
<thead>
<tr>
<th>POLLUTER</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATE</td>
</tr>
<tr>
<td>NCOD</td>
</tr>
<tr>
<td>CBOD</td>
</tr>
<tr>
<td>AMMN</td>
</tr>
<tr>
<td>DO</td>
</tr>
</tbody>
</table>

#### b. Reach Property Record Type

<table>
<thead>
<tr>
<th>REACH PROPERTY</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

#### c. Information Base Representation

<table>
<thead>
<tr>
<th>NAME</th>
<th>POLLUTER</th>
<th>PARAMETERS</th>
<th>HEADWATER</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DATE</td>
<td>DATE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NCOD</td>
<td>PDATE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CBOD</td>
<td>FLOW</td>
<td></td>
</tr>
<tr>
<td></td>
<td>AMMN</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DO</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 20. Logical Structure Example (REACH)**
ETER', etc.) is also the label of a record type which is itself an aggregate of item types and which may have numerous occurrences. So, for instance, 'POLLUTER' denotes an occurrence of REACH PROPERTY; but it also denotes a record type (shown in Fig. 20a). The same circumstance holds for the other reach properties, though their record types are not depicted here. The resultant logical structure is illustrated in Figure 20c.

Continuing the example, general water quality modeling characteristics are examined. In order to simulate water quality information is needed about the following: the rivers involved, the reaches which are in each river, each reaches' properties, junctions, piping plans, and model parameters. This is shown in the structure of Figure 21. Note that the record type GMC has two item types: CHAR (characteristic) and IMPT (a measure of the relative importance of each characteristic). Five occurrences of GMC are shown: RIVER, REACH, JUNCTION, MODEL and PIPE PLAN. General Modeling Characteristic is not the only property of a segment that needs to be represented; Local Modeling Characteristics (LMC) are also needed. (The term segment is used to indicate a particular area of a river basin.) The details of the high level record type LMC are not shown here, but they describe information about non-point sources of pollution, permits for point-source pollution, treatment plant construction status, permit violation data, etc. As shown in Figure 22, GMC and LMC are occurrences of SEGMENT PROPERTY which is itself an occurrence of the record type WQMA; BASIN and SEGMENT are also occurrences of this record type. The information base could be further extended to in-
Figure 21. Logical Structure Example (GMC)
Figure 22. Logical Structure Example (MQA)
corporate aspects of land use planning since they influence and are influenced by water quality management.

The foregoing logical structures are initially defined in terms of an Information Description Language (IDL). Use of the IDL to define the logical structure of Figure 21 is presented in Figure 23. (In-sets S2 and S3 are not shown.) The specification shown is largely self-explanatory. For simplicity, details of the type and size of items are not shown; also the ordering criterion of each set is not shown.

Language for Information Base Access

The reader will observe from the preceding discussions that the decision support system has two basic components: an information base and a query language (interpreted by a query processor). Clearly the usability of a semantic network depends upon implementation of a language with which one can extract (insert) meanings that are held in the semantic net. Not only are semantics conveyed by a particular language, they are limited by it as well. The language is used to express meanings, but it also delineates the kinds of meanings which are expressed. One can devise arbitrarily complex semantic networks, but their usability is (from the practical standpoint) constrained by the languages (and language processors) which can be interfaced with them. Observe then that there is a fundamental duality of 1) the language in which ideas are expressed, and 2) the structural representation of ideas in an information base. On the other hand the semantic mechanism must be capable of taking full advantage of the language's power. In the case of the implementation described in this paper, the query language is the
Figure 23. Example of IDL for Figure 21
constraining factor since it is intended primarily for the practical support of decision activities of managers in both the public and private sectors.

Implementation of a natural language (e.g. English) processor is certainly a noble objective. It is not unusual that the typical decision maker neither uses, nor needs, a complete facility for conversing in a natural language. It often happens that phrases or clauses are sufficient to convey an idea; there are grammatical constructs (e.g. reflexive, passive) which are not particularly germane to the decision activities of information access, model formulation, and analysis. In addition the decision maker is more prone to desire information conveyed in a tabular or graphical fashion than in a narrative mode. The user sitting at a computer terminal has a tendency to use abbreviations and concise mathematical notation.

With these factors in mind, the query language used by the GENI query processor has been designed to meet the needs of decision makers for flexibility and brevity of expression, while at the same time being easy to learn and utilize. In particular, the query language used is the same as that of the GPLAN query system. However, the manner in which that language is processed is considerably different. Chapter 6 investigates the GENI query processor, in detail.

Upon receipt of a query, the GENI query processor generates appropriate commands for traversal of a multi-level network. These commands are operators in an Information Manipulation Language (IML). The term IML is used to distinguish from the Data Manipulation Language (DML) pro-
posed in the CODASYL DMTG Report [27]. Recall that the DML is intended to permit access, modification and retrieval for a single level network database. The IML has the more extensive function of furnishing tools for manipulation of the information base. Thus the IML contains operators for handling traditional DML functions (see Chapter 3) and operators for processing higher level record types. The latter, originally suggested in [11], are discussed here.

In the DML, routines exist for creating a record occurrence at a unique location denoted by its key. The IML includes analogous routines for specifying that an existing record occurrence be treated as a record type as well. There are four such commands:

- **CRTK**, Create Record Type based on a given Key
- **CRTT**, Create Record Type based on the current occurrence of another Record type
- **CRTO**, Create Record Type based on the current Owner of a given in-set
- **CRTM**, Create Record Type based on the current Member of a given in-set

Another traditional DML operator (AMI) adds a specified record occurrence as a Member of a given In-set. A similar IML operator is used for adding an existing occurrence of one record type as an occurrence of another record type. Note that utilization of this operator must be preceded by a generalization of the definition of a record type which was introduced above (i.e. the definition is generalized by removing the restriction that \( \sum_{i} r_i(x) \leq 1 \), \( \forall x \in X_k \), where \( r_i \in R_k \)). This operator is **AORT**, Add Occurrence to Record Type, and it uses the key of the occurrence to be added. In conjunction with commands for the logical restructuring of a network database, AORT provides the ability to add and delete higher
level record types and add existing occurrences to higher level record types; and this is accomplished without dumping and reloading data.

Finally operators are needed for determining the key of a record type, given an occurrence of the record type. These commands are:

GKRR, Get Key of the Record type for the current Record occurrence of that type

GKRO, Get Key of the Record type whose occurrence is the current Owner of some in-set

GKRiM, Get Key of the Record type whose occurrence is the current Member of some in-set

These operators provide the capacity to proceed from a lower level occurrence to a higher level occurrence, when used in conjunction with traditional DML operators.

It must be emphasized that the typical user of the query system needs to have no knowledge of the INL operators, for they are automatically set up and executed by the GENI query processor in response to a user query.

Model Representation within the Information Base

It is important to observe that network, tree, and linear data structures have traditionally been used for storage of empirical information, but they are not commonly used for the representation of conceptual information (e.g. models). It is imperative to have a systematic means for the representation and management of conceptual information, if the goal of a generalized, intelligent (GENI) query processor is to be realized. This is particularly pertinent to the activity of formulating models by relating certain parcels of conceptual information into desired
configurations. In order to perform analyses it is necessary to know the meaningful ways in which the empirical information may be related to the conceptual. With the information base such knowledge is represented in a single information structure, along with knowledge about interrelationships among data values and interrelationships among models. These interrelationships may be horizontal or vertical in nature.

We are now prepared to examine some particulars of the inclusion of conceptual information in the information base. This examination does not pretend to be exhaustive, but rather is offered in a suggestive, stimulatory spirit. Examples of kinds of conceptual information being considered here range from such relatively simple ideas as the costs of a water treatment facility to the more complex notions or water quality within a river basin. These ideas, notions, or concepts are operationally defined by computer programs and they function as models which, when fitted with compatible empirical information, result in certain expectations, beliefs, or facts. For instance, the familiar concept of linear regression is a model (i.e., a system of ideas) which may be formalized as a computer program. This program effectively defines the concept of regression. The activity of fitting the model with compatible inputs to perform needed analysis is a primary function of the query processor and will be discussed in the next chapter.

One method for integrating programs into the logical information structure is to allow each program to be indicated by a record occurrence. These are formed into groups (i.e. into record types) according the types of analyses which may be performed with them; i.e., according
to the concept of which they are alternative definitions or exemplifications. Suppose for example that there are several methods for determining the cost of a treatment facility; that is, there are several distinct ways in which the concept of cost may be defined. Then each of these programs may be identified as an occurrence of the same record type. Each program record occurrence contains the program name, location of the compiled program code, etc. The discussion of a program's input and output requirements is deferred to the next chapter.

Given this method of representing models as occurrences in an information base, we have a basis for constructing a large model via the in-set relationships between stored models. That is, the outputs of a given model may reasonably be used as inputs to other models. As will be seen, this feature of model formulation is an important aspect of the GENI query processor. Suppose a query requests that some very complex analysis be performed, but it does not explicitly indicate the configuration of models to be used. It is the task of the GENI to determine which formulations are appropriate for the requested analysis. In addition it may determine the relative values of each such formulation with respect to some standard of model evaluation.

Observe that this capacity for model formulation necessitates a knowledge of which configurations are meaningful and which are not. In order to maintain the query processor's generality such knowledge is stored in the information base's semantic network. An example of this is given in the specification of GENI itself, which appears in Chapter 6. It is of interest to note that the storage of inter-model relationships
has much in common with structured programming. Programs devised according to the principles of structured programming are readily amenable to storage within the information base. In effect, the GENI provides a means for managing modules of the structured variety.

**Advantages of the Resolution Level Facility**

The concept of resolution levels effectively adds a new dimension to the field of information storage. The preceding discussion has suggested a means for operationalizing this concept as an extension to the traditional single-level network approach. One advantage is that multi-level semantic networks may be stored without introducing asymmetry in the interpretation and processing of in-sets and record types. Since a record type may also be defined to be an occurrence of a higher level record type, the addition of a record type is treated by creating a new record occurrence at the next higher level. That is, we remove the distinction between data values and the structural pattern according to which data is organized. In other words, the terms 'attribute' and 'value' are recognized as being relative, so that what is a value on one level is an attribute on another and vice versa.

From one viewpoint this abolishes the special status of an IDL specification by permitting record type definition to be a dynamic process. That is, the creation of a new record type is synonymous with the creation of a new record occurrence of a higher level record type. Thus the IDL specification of the highest level of resolution is effectively reduced to the definition of three record types (one describing information about record types, another relating to information about sets, and
one with various system information [11]) and some in-sets between them. This definition is always the same regardless of the content and structure of lower resolution levels.

A second advantage, already mentioned in connection with integration of programs into the information base, concerns a mechanism for handling levels of abstraction in software. A third advantage is that higher level record types may be used to characterize areas of an information base by assigning a particular area to be occurrences of a higher level record type; these areas may be defined for a variety of reasons (e.g., for information security, to denote scenarios, to delimit functional areas - which may overlap, etc.). As the information base becomes large and varied in content, this technique may also be used to realize efficiencies in path determination processing by limiting the scope of network traversal to a particular information base area.

The Information Base as a Device for Semantic Representation

With the foregoing background, we can now address the three criteria proposed by Woods [86], which must be satisfied by a notation used for semantic representation. First observe that the information base is a tool for the representation of a semantic network (i.e., a single mechanism with both the ability to store factual knowledge and the ability to model associative connections which render certain parcels of information accessible from certain others).

The first criterion of a notation for semantic representation is logical adequacy. The notation must provide an exact, formal and
unambiguous representation of any particular interpretation that may be
given to a sentence. Recall that the sentences with which we are concerned
are those allowed in the query language for decision makers. The informa-
tion base allows a given query to have a multitude of interpretations.
The query specifies a group of data items which may be related to each
other in many ways via vertical and horizontal linkages in the informa-
tion base. Each path of linkages on which these items lie corresponds to
a particular interpretation of the query. Upon receiving a query which
is subject to multiple interpretations the query processor prompts the
system's user in order to ascertain which interpretation (i.e., path) is
intended. Details of the manner in which this has been implemented may
be found in [47],[13].

The second criterion is that there must be an algorithm for trans-
lating an initial query into the notation of the information base. This
is the central function of the query processor whose operation has already
been described; implementational details appear in [11]. The third
criterion, concerning algorithms capable of using the semantic represe-
tation, has also been addressed in the discussion of the query language.
Observe that the IML provides the means for interfacing algorithms with the
semantic representation. Algorithms which have been used (by the GPLAN
system) range from relatively commonplace report generators to large scale
water quality simulation models [45].
Summary

In the beginning it was observed that it is the patterning of symbols which can convey information; a datum's meaning derives from its context, from its relationships with other data. Thus when considering the design and implementation of systems for decision support, a crucial point is the power of available tools for representing contexts. The value of such systems is constrained by the 'richness' of patterning allowed by their data structure mechanisms. Observing the progression from relatively impoverished linear structures to trees and networks, we note that each stage has provided a more powerful and flexible tool for semantic representation. In this chapter the notion of an information base was introduced as a natural step forward in the continuing evolution of data structures. An outstanding feature of the information base is its accommodation of both the horizontal and vertical integration of information parcels into a single mechanism. An information base implementation which builds upon network concepts was discussed. A second distinctive feature of the information base, namely the integration of program modules into its structure, was also described. The information base is utilized by a non-procedural, English-like query language, that has been designed for decision support applications. This language, in conjunction with the information base, satisfies the requirements for a notation for semantic representation.
CHAPTER 6
THE GENERALIZED INTELLIGENT QUERY PROCESSOR FOR DECISION SUPPORT

Introductory Observations

Research in the field of artificial intelligence aims at discovering ways in which computers may be used to emulate human cognitive (i.e., perceptual and judgemental) processes. The specific topic considered in this chapter is that of an intelligent query processor which is general in the sense of its capacity to treat a broad spectrum of formal decision making applications. That is, the query processor itself is invariant to both changes in application and modification of the types of formal analyses pertinent to a given application area. For example, the same query processor may be used in such diverse decision support applications as water quality management and material requirements planning. Moreover, within an area such as water quality management, alterations in the types of formal analysis to be performed do not affect the query processor. This feature of generality is examined in more detail in ensuing sections of this chapter.

Now with regard to the notion of an intelligent query processor, one definition holds that a system is intelligent to the extent that it is able to successfully cope with new situations. In this sense the query processor's feature of generality contributes to its 'intelligence'. How-
ever within the present scope, usage of the phrase "intelligent query processor" should be considered to indicate intelligence in the sense of artificial intelligence. Thus the objective of the investigation is a generalized query processor which emulates human perceptual and judgmental processes. No claim is made with respect to the maintenance of structural analogies between components of the processor and components involved in human cognitive activities. The objective is merely to endeavor to capture within the processor, some of the important functional activities involved in perception and judgement. These include 1) the translation of incoming messages into an internally comprehensible language; 2) the retention and integration of various types of information in a single mechanism (an information base) which serves as a semantic representation of extant knowledge about an application area under investigation; 3) the activity of associating outward perceptions emanating from the system's users, with inward perceptions of the information base; and 4) the activity of making judgements or decisions. The first three concern the facet of information collection (perception). As suggested in Chapter 4, the fourth point involves the facets of model formulation, analysis and problem recognition. Figure 14 presents an overview of how these various facets are related to each other within the GENI query processor. The following sections examine the design and implementation of each facet within this query processor context. It is assumed throughout that facilities (e.g. terminal system) are available for transmitting queries to the processor and receiving messages from it.
Information Collection

Recall that this facet of perception has two basic aspects. The first is information collection from sources external to the system (i.e., from the user) and this necessitates a decoding process, transforming incoming messages into an internally comprehensible form. Decoding in the GENI query processor is based upon the GPLAN query processor's implementation, which has already been discussed in some detail.

The second basic aspect of information collection involves acquisition of information from sources internal to the system (i.e., from the information base). This involves finding paths in the information base. Its implementation is also based upon the GPLAN automatic path determination.

Decoding

Remarks about decoding are prefaced with a brief review of the query language (see Chapter 3). Recall that design of this language was undertaken with the following criteria in mind:

1) Convenience to non-programming users in terms of non-procedural queries and a simple query syntax;

2) Generality in terms of the capacity to handle the lexicons of diverse problem areas;

3) Flexibility in terms of the ability to support requests for both traditional retrieval activities and the activities of model formulation and execution. Retrieval is not limited to the production of a few standard reports; on the contrary, specification for retrieval of any configuration of data is allowed.
Recall the basic query syntax:

<COMMAND> <FIND CLAUSE> <CONDITIONAL CLAUSE>

The command denotes a type of model that is to be executed. The retrieval clause is used to specify what data are to be used in that analysis. Any conditions for restricting the retrieval or the operation of the command are specified in the conditional clause. So the conditional clause consists of a group of predicates, each of which specifies some relation between a query item and a value for that item (e.g., AMOUNT > 25, CITY = 'LAFAYETTE'). A query item is simply an element of its query language's lexicon for a particular problem area. It is important to stress that a query to the GENI processor might not explicitly state the problem to be solved. An example of this is a situation where items expressed in the retrieval clause do not explicitly exist in an information base, but must be derived from appropriate analyses. Another example of this occurs when there are several alternative methods for executing the type of model indicated by the command; this requires that inferences be made, based upon contents of the other query clauses. The facet of problem recognition is the ability to make implicit requests explicit. However before such activity can take place, the original query must be decoded into an internally comprehensible form.

Formally speaking, the query language has a context-sensitive grammar [2]. Using Chomsky's notion of transformational grammars [26] the decoder applies inverse transformations to a statement in the query language in order to produce a corresponding expression in a language derived from a context-free grammar [2]. It is this context-free expression which is
used by the problem recognizer in order to arrive at an explicit problem statement, which is also an expression in the context-free language. The advantage of a context-free grammar is that it yields expressions which can be compiled by using well-known methods of syntax-directed analysis [2]. Details of the manner in which this has been implemented (including the grammar and precedence parsing tables) may be found in [16]. As a final note, all lexical information is retained in the information base to preserve the generality of the decoder with respect to problem areas that can be handled (see record types QUIT and MDL in Figure 24).

Path Finding

Remember that a path is a sequence of in-sets which are related by intervening record types, such that any two record types on the path are linked by a unique sub-sequence of the path's in-sets. In Figure 18, two examples of paths are the sequences S1, S3 and S2, S4, S3; but S1, S3, S4, S2 is not a path because the record types PROPERTY and CAR are linked by two sub-sequences of S1, S3, S4, S2 (namely S1, S3, and S4, S2). Be careful to note that the path definition contains no notion of 'direction'. In the diagram of a logical structure the direction of the arrows is immaterial for purposes of path specification.

Suppose that we are interested in understanding the relationship among some data item types that do not belong to the same record type. It is the task of the path finder to 'look' at the information base to locate every path which passes through all of these data item types (i.e., through all record types which have any of these data item types). The algorithm for accomplishing this is discussed in [13]. Each path corre-
responds to a different interpretation of the functional (or structural) relationship among the items. Thus if some group of data item types appears in a query, the information base gives a means for formal and unambiguous representations of various interpretations which may be given to the query.

Thus we see once again that the information base satisfies the first condition for a semantic notation: logical adequacy. Recall that the second condition requires that there exists a means for translating an initial query into the information base notation. The GENI query processor fulfills this condition since the decoding results in a group of items and the path finding determines the possible interpretations in terms of paths in the information base. Recall that the third criterion for a notation for semantic representation is the existence of algorithms that can use the information base to perform needed inferences and deductions. Subsequent sections address this criterion.

**Problem Recognition**

The activity of problem recognition is a function not only of an incoming query, but of some knowledge about the problem area as well. One approach to handling such knowledge is to incorporate it directly into a query processor and the application models which the processor can use. This is a highly specialized approach and it does have some practical viability, especially in situations where knowledge about the problem area is static and the types of analysis required are not subject to change. But the approach introduced here is not restricted to these kinds of situations. It necessitates no changes in the code of the problem recognizer
(nor in the code of application models), regardless of the nature of the
problem area(s), the acquisition of new knowledge about the problem
area(s), or the desire to perform new types of analysis. Moreover, this
generalized method of treating problem recognition is indispensable to
the overall objective of designing a generalized intelligent query proc-
essor.

In short, the approach consists of recording conceptual knowledge
about a problem area in the system's information base. By representing
conceptual knowledge in this manner, it is just as amenable to modifica-
tion as any empirical information. The conceptual information needed for
successful problem recognition must indicate what type of analysis is to
be performed for a given query; it must be sufficient to allow the recogni-
zer to determine the implications of a literal request and to produce
a precise statement of the problem to be solved. In inter-human discourse
there are frequently implications which are attached to messages by both
recipients and senders (i.e. implications that are not explicitly stated). The human-computer discourse should also address these economies of ex-
pression.

Types of Analysis

A query is recognized as requesting a particular type of analysis
on the basis of the command(s), query item names and associated data value
ranges which are present in the query. A simple information structure for
representing knowledge about types of analysis is shown in Figure 4. As
that structure indicates, a given model may be used for several types of
analysis, each of which is denoted by a set of (query item name, data
value range) pairs. Associated with each of these pairs there may be zero, one or more deep structure expressions. Once a query is recognized as requesting a particular type of analysis, the deep structure expressions for that type of analysis are inserted into appropriate clauses of the decoded query. What this amounts to is the application of additional inverse transformations as a result of inferences made by the problem recognizer.

What follows is an algorithm for identifying the types of analysis associated with a given model. Let \( Q \) be the set of all query items \( q \) which are allowable in query expressions for some problem area. Then the cardinality of \( Q \) is equal to the number of occurrences of the record type QUIT (Figure 24). Some of these \( q \) can take on (i.e., be modified by or qualified by) certain values. Let \( V \) be the set of all possible values which can occur for the members of \( Q \). We let \( U \subseteq Q \times (V \cup \emptyset) \) be the set of all \((q,v)\) such that \((q,v)\) is meaningful with respect to the problem area. Referring to Figure 24, these pairs are stored in the information base as occurrences of QUIT and VAL which are associated via the in-set.

If \( \text{RIVER-NAME} \in Q, \text{POLLUTANT} \in Q, \text{"AMMONIA"} \in V, \text{and } \text{"WABASH"} \in V \) then the pairs (POLLUTANT, 'WABASH') and (RIVER-NAME, 'AMMONIA') are probably meaningless if the problem area is that of water pollution control. But the pairs (RIVER-NAME, 'WABASH') and (POLLUTANT, 'AMMONIA') do have meaning for this problem area and would be stored in the information base structure shown in Figure 24.

Now given a model, as indicated by the query's command clause, various subsets of \( U \) may appear with it. Each such subset implies that the
model is to be used in some type of analysis. Let $T_1, \ldots, T_N$ be the
types of analysis which may be requested with respect to a particular pro-
gram. Each $T_k$ ($k=1, \ldots, N$) is denoted by a unique subset of $U$. Let
$||T_k||$ signify the cardinality of that subset which denoted $T_k$. Let $S$
be the set of all members of $U$ appearing in some query; $||S||$ is its car-
dinality. Then the following algorithm may be used to locate every $T_k$
which has all of its members contained in $S$, where $s_i$ signifies the $i^{th}$
element of $S$:

1. $\text{IND}_k = \begin{cases} 1 & \text{if } ||T_k|| \leq ||S|| \\ 0 & \text{otherwise} \end{cases}$ \quad \forall k = 1, \ldots, N

2. $k = k + 1$; if $k > N$ then stop

3. If $\text{IND}_k \neq 1$ then go to 2

4. $i = 0$

5. $i = i + 1$; if $i > ||S||$ then go to 2

6. If $s_i \in T_k$ then go to 5

7. $\text{IND}_k = 0$

8. Go to 2

If $\sum_{k=1}^{N} \text{IND}_k > 1$ then more than one type of analysis has been found
and the user is prompted to determine which is actually desired. Alter-
natively, it may be possible to evaluate which of the types is 'best'.
Such a procedure would require that all evaluation information and pro-
cedures, that are problem area dependent, be stored in the information
base to preserve the query processor's generality. Furthermore, the log-
ical structure for storing dependent information needs to be invariant to
changes in the problem area. This last condition also holds for the
structure used to handle information concerning the types of analysis for the various models; the structure shown in Figure 24 is only one possibility for this. No claim is made for this particular structure with respect to being 'optimal' or being able to treat all contingencies. It is used for illustrative purposes.

For an example of how the problem recognizer works, consider the query RUN COST FOR PERCENT.REMOVAL = .3 AND PARAMETER = 'AMMONIA'. Figure 25 shows relevant knowledge about the problem domain, which is stored according to the logical structure of Figure 24. The query processor's perception activity identifies three query items: COST, PERCENT.REMOVAL AND PARAMETER. Problem recognition involves a search of the information base in order to find a type of analysis that is denoted by these three items and their associated values; in Figure 25, TOA-12 is an example of this. Observe that each of the query items may be associated with many other types of analysis. Also notice that data values in the original query fall within the ranges shown in the occurrences of VAL. Application of the occurrences of DSE to the original request results in RUN TC FOR C1=M1(.3), TC=TC5. So the value of PARAMETER implies the particular total cost model (TC5) which is to be executed. The value of PERCENT.REMOVAL indicates that an elementary cost (C1) is to be computed by a particular method (M1) with .3 as a parametric input. Note that greater flexibility in specification of the conditions, under which deep structure expressions are to be applied, is permitted if we allow a many-to-many relationship between the record types VAL and DSE.
Figure 24. Logical Structure for Support of Problem Recognition
The Problem Reduction View

The process of problem recognition as a whole, including its use of the model formulation ability, can be characterized in terms of problem reduction or in terms of linguistic transformations. In the former, a query is treated as a problem which is to be reduced to a group of directly solvable primitive problems. The reduction occurs through the application of rules stored in the information base. In the linguistic characterization of the recognition process, the query is viewed as an expression in a surface structure grammar. Inverse transformations are applied to the surface structure to arrive at an expression in a deep structure grammar which can be readily compiled. There are intermediate expressions generated during this process, each of which is 'deep' with respect to the original query.

Thus we can say that, in the problem reduction characterization, operators (which are inverse transformations) reduce problem descriptions (which are strings in a surface language) into subproblem descriptions (which are strings in a deep language). Given a problem description, (surface language string) it may occur that there are, several alternative sets of subproblem descriptions (alternative deep structure strings) which could result during the reduction process. Each set of subproblem descriptions is such that the solvability of all subproblems within the set implies a solution to their original problem.

Suppose that for some query the model formulation process has furnished several alternative subproblem sets (deep structure strings), each of which is solvable and therefore implies a solution to the query. Ob-
serve that these solutions are not necessarily identical. Just as a
statement in inter-human discourse may have several meanings, so may a
query submitted to the intelligent query processor be subject to several
interpretations. The number of deep structure strings reflects the num-
ber of interpretations which may be given to a query. Naturally, the
more explicit a query is, the fewer the number of interpretations which
may be given to it. In the event of multiple interpretations it is
the task of the problem recognition process to resolve the ambiguity by
interaction with the user and by consulting stored information about the
problem area. This gives what was previously called the deepest structure.

The reduction of a problem into alternative subproblem sets is
often represented in terms of an AND/OR graph [63]. An example is shown
in Figure 26. The total cost problem (TC) can be solved by solving either
TC1 or TC2 or TC3. Note that C1 and C2 and C3 must be solved in order
to solve TC1; they therefore constitute TC1’s subproblem set. The sub-
problem set of TC3 consists of C2 and C5. Continuing, we see that C1
has two subproblem sets, each having a single member. The rule to be
followed in constructing AND/OR graphs is that if a node has any suc-
cessors at all they are either all OR nodes, or all AND nodes [63].

From the AND/OR graph, it is clear that TC has four possible solutions.
Suppose that the graph is in some way recovered from the information base
(in the next section we shall examine how this is accomplished) during the
course of attempting to recognize what the user intends by "SOLVE TC."
The four possible solutions correspond to four alternative models. The
crucial issue for the problem recognizer is determination of which is to
Figure 25. Occurrence Structure for Types of Analysis
be utilized. One approach consists of describing the alternatives to the user, who then makes a choice. This can be done at every node that has OR nodes as its successors. Referring to Figure 26, the user would be presented with descriptions of the salient aspects of TCl, Tc2, and Tc3. If TCl is chosen descriptions of M1 and M2 are provided and another choice is made. Alternatively, the user could be asked to choose among the four overall possibilities after being given a complete description of each.

Presumably the rationale for being able to produce alternative reductions is to allow for different meanings of a concept (e.g., "total cost"); the intended meaning of a given reference to the concept may sometimes be deduced from the context in which that reference is made. For example, reference to COST accompanied by references to certain parameters (e.g., SOLVE COST FOR PERCENT-REMOVAL = .3 AND POLLUTANT = 'AMMONIA') might imply that TCl is intended and that M1 is to be chosen for Cl. In other words user-specified conditions on the use of TC may be used to determine the type of analysis which is to be performed. Treatment of types of analysis was discussed in the previous portion of this section about problem recognition.

This technique of storing information about interpretations, which are to be given to certain contexts, reduces the need for interrogation of the user. The occurrences of DSE (see Figure 25, for example) are solvable primitive problems. As already indicated, a matching procedure is used to obtain the appropriate set of primitive subproblems. Observe that Figure 24 is a schema with which one can define an arbitrarily large number of
Figure 26. Illustration of an AND/OR Graph
primitive problems. But as this number becomes large, the speed of problem recognition is degraded. Therefore it may not be feasible to store interpretive information about all possible types of analysis; hence the need for the previously mentioned interactive facility. In the interactive mode of problem recognition the user is coaxed into specifying the subproblem set to be used.

**Model Formulation**

Construction and subsequent execution of a desired model is based on the modification and combining of various known modules. Rules for the allowable modifications and combinations are stored in the structure of the information base. If we think of a program as defining (or exemplifying) some concept, then a modification of that concept consists of a redefinition (i.e., the selection of an alternate program to define the concept). For instance, there may be many alternative ways to define the concept of total cost; the concept of total cost is modified according to the context in which it is to be used. In the information base, alternative definitions of a concept are treated as occurrences of a record type for that concept. Of course there may be various levels of conceptual resolution. So there may be N ways to define total cost, but there may (on a different level) be alternative ways to define each of these N. In other words, a concept may be defined by other concepts, which are themselves defined by still other concepts. It is important to be able to represent all of these various meanings in a semantic net. This is accomplished in the information base by means of the single technique already indicated; use of this single structural technique to represent alternate meanings facilitates the query processor's interpretive ability.
We can now examine relationships between information base structures and AND/OR graphs with respect to model formulation. As previously mentioned, generation of the AND/OR graph from a given query and a given logical information structure aids the problem recognition ability of the query processor. On the other hand, design of a logical information structure is partially based upon an implicit knowledge of AND/OR graphs pertaining to the problem area under consideration. An avenue of future research will investigate the utilization of explicit AND/OR graphs as aids to information structure design.

The Graph - Information Structure Correspondence

Our consideration of the correspondence between AND/OR graphs and information structures commences with the OR case (see Figure 27a). Where there is an OR relationship between a parent node and its successors, all successor nodes correspond to distinct record occurrences of the record type indicated by their parent node. Thus node a corresponds to a record type with \( b_1, b_2, \ldots, b_n \) as its occurrences. In the AND case (see Figure 27b) the parent node a corresponds to an occurrence of the record type A. Successor nodes are of two sorts: record occurrences or record types. If the successor nodes \( b_1, b_2, \ldots, b_n \) correspond to record occurrences, then the AND relationship means that they are associated with the record occurrence node a by means of \( m \) in-sets \( m \in \{1,2,\ldots,n\} \) each of which is owned by record type A. There can therefore be up to \( m \) distinct records types (\( B_1, B_2, \ldots, B_m \) where \( \Lambda m \)) which are members of the \( m \) in-sets. Each \( b_i \in \{b_1, b_2, \ldots, b_n\} \) is an occurrence of at least one of these distinct member re-
a. The OR Case

b. The AND Case

Figure 27. Two Problem Reduction Cases
cord types and each of these distinct member record types has at least one element of \( \{b_1, b_2, \ldots, b_n\} \) as an occurrence. Observe that it is permissible for \( A = B_i \) for some \( i \in \{1, 2, \ldots, I\} \). The above description can be generalized to the case where \( a \) is an occurrence of more than one record type \( (A_1, A_2, \ldots, A_p) \). In such a case the in-sets which associate \( a \) with the \( b_1, b_2, \ldots, b_n \) may be owned by various elements of \( \{A_1, A_2, \ldots, A_p\} \); of course, each in-set has only one owner record type. Another generalization consists of allowing direct \( N \) to \( M \) relationships in addition to in-sets.

This paragraph is a short digression to elucidate the notion of directed \( N \) to \( M \) relationships with respect to program modules. Each program module is represented by a record type. If some group of modules are considered to be alternative definitions of some concept (e.g., alternative methods for defining the concept of 'total cost'), then in the information base they are treated as occurrences of a record type describing that concept. In-sets are used to represent the ways in which program modules can be combined into models. It is in this connection that the 'direction' of an in-set is especially important. The convention is adopted that member occurrences owned by an owner occurrence are required for the solution of the program module represented by that owner; specifically, the member modules are called by the owner module. This convention must be extended to handle the case of \( N \) to \( M \) (i.e., many-to-many) relationships between program record types. In general \( N \) to \( M \) relationships between occurrences of two record types are easily handled by creation of a mediating third record type which each of the two
original record types owns by means of an in-set \((N \geq 1, M \geq 0)\). Following the same procedure to treat \(N\) to \(M\) relationships, between occurrences of two programs record types, gives no immediate means for ascertaining which of the two types of program modules is subsidiary to the other. A solution is to designate an item type in the third, or mediating, record type for the purpose of indicating the 'direction' of the subsidiary program record type. Notice that it is the value of occurrences of this item which indicate the direction, or possibly the absence of direction. The latter contingency is indicative of recursion among program modules. A mediating record type which contains a directional indicator is hereafter referred to as a 'gate'. Record occurrences not pointed to by the gate are said to be owners of the gate; record occurrences which are pointed to are said to be members. In a sense the gate is just a generalization of the in-set notion.

The second sort of successor node which may appear in an AND graph is a node corresponding to a record type. Suppose \(b_i\) is a record type node which is related to its parent node \((a)\) by means of an AND relationship. Then in the information structure, the occurrence corresponding to node \(a\) must be an occurrence of the record type corresponding to node \(b_i\). Since the information base allows an occurrence of one record type to also be an occurrence of other record types, there may be more than one record type successor node in an AND graph. Thus Figure 5(b) several of the successor nodes may correspond record types while the remainder of the nodes correspond to record occurrences.
It should be clear that there can be more than one logical information structure which is descriptive of a given AND graph. This implies that a logical information structure is able to capture more information (than an AND graph) about the nature of the possible relationships between \( \{b_1, b_2, \ldots, b_n\} \) and \( a \). Figure 28 shows three possible logical information structures which capture the information represented in the AND graph of Figure 27b.

We now examine what happens when AND cases and OR cases are combined to obtain an AND/OR graph. Observe that the status of a node may be of eight possible types as enumerated in Figure 29. Nodes of types 1 and 5 must describe record types. Nodes of types 2, 4, 6, and 7 must necessarily describe record occurrences. A type 8 node may correspond to either a record type or record occurrence. Types 3 describe the situation of a record occurrence which must also be a record type; such a node must correspond to a record occurrence due to its relation to its parent, but the same node must also describe a record type by virtue of its OR successor. Thus in developing a logical information structure to describe an AND/OR graph, a type 3 node must be represented by a record occurrence that is also a record type. Although a node of type 1 or 5 must describe a record type, it is also allowed to describe a record occurrence. Similarly a node of type 2, 4, 6, or 7 must correspond to a record occurrence, but it is also allowed to correspond to a record type. Therefore, given an AND/OR graph, there are many information structures to represent it. As the number of AND/OR graphs to be represented in a single structure increases, the number of possible structures becomes more constrained.
a. $-a$ is an occurrence of $A$
   $-b_1, b_2, \ldots, b_n$ are occurrences of $B_1$

b. $-a, b_1, b_2$ are occurrences of $A$
   $-b_4, b_5, \ldots, b_n$ are occurrences of $B_1$

c. $-a$ is an occurrence of $A$
   $-b_1 = A$
   $-b_2, b_3$ are occurrences of $B_1$
   $-b_4, b_5, \ldots, b_n$ are occurrences of $B_2$

Figure 28. Possible Information Base Structures for Representation of the AND Case
Graph Generation from an Information Structure

We now examine an algorithm that generates a unique AND/OR graph from a given information structure and a given record type (or record occurrence) to serve as the start node. The algorithm to accomplish this makes use of three operations which we now define: the OR expansion operator is applied to a record type node and consists of creating a node for each occurrence of that type. Each of these occurrence nodes is linked to its parent record type node, such that an OR relationship exists between the parent and its successors.

The AND expansion operator is applied to a record occurrence node, where that occurrence is of a record type which owns an in-set (or which 'owns' a directed N to M relationship, i.e. a gate). The result is that a node is created (if it does not already exist) for every occurrence of the in-sets' (or gate's) member record type, which is associated with the (owner) record occurrence described by the node in question. If this parent node has no OR successors, then all of its associated member occurrence nodes (both new and already extant) are linked to the parent by an AND relationship. If the parent node already has OR successors, this linkage is not made to the parent. The linkage is made to the node which lies at the end of every strictly OR path emanating from the parent.

The generalization operator is applied to a record occurrence node (say, \( N_o \)) whose record type is not already represented by a node in the graph. Recalling that \( N_o \) may be the occurrence of more than a single record type, a node is created for every such unrepresented record type. If \( N_o \) has no OR successors, then the nodes describing its record types are
linked to $N_0$ by means of an AND relationship. However if the record occurrence node has OR successors, the AND linkage is made to the node at the end of every strictly OR path emanating from $N_0$.

The three operations are now defined in a more formal manner and then demonstrated pictorially. The notation OR ($a;b_1, b_2, \ldots, b_n$) denotes an OR graph like that shown in Figure 27a. Similarly, an AND graph (Figure 27b) is denoted by AND ($a;b_1, b_2, \ldots, b_n$). The successor operator $\Gamma (\text{NODE}) \rightarrow \{S_1, S_2, \ldots, S_7\}$ yields the set $S$ of the end nodes of the $\ell$ strictly OR paths emanating from NODE. In Figure 30a, the application of $\Gamma$ to NODE gives the set $S = \{S_1, S_2, S_3, S_4\}$. If NODE has no OR successors then $\Gamma (\text{NODE}) = \{\text{NODE}\}$.

The OR expansion (indicated by $\phi$) of NODE is defined as follows:

$\phi (\text{NODE}) \rightarrow$ OR (NODE; $0_1, 0_2, \ldots, 0_k$) if NODE describes a record type having the $k$ occurrences $0_1, 0_2, \ldots, 0_k$. Application of $\phi$ to a node, that corresponds to a record occurrence only, results in a null operation.

The AND expansion (indicated by $\Theta$) has as arguments a node and an in-set, such that the node corresponds to a record occurrence whose type owns the in-set. Define $\Theta (\text{NODE}, \text{INSET}) \rightarrow \text{AND} (S_1; 0_1, 0_2, \ldots, 0_k)$. The $0_1, 0_2, \ldots, 0_k$ are the $k$ occurrences of INSET's member record type that are associated with NODE (which is an occurrence of INSET's owner record type). The $S_i$ ($i=1, 2, \ldots, \ell$) are determined by $\Gamma (\text{NODE})$.

The generalization operator (denoted by $\wedge$) is defined by $\wedge (\text{NODE}) \rightarrow \text{AND} \ i=1, \ldots, \ell \ (S_i; T_1, T_2, \ldots, T_k)$ where NODE describes a record occurrence which is an occurrence of each of the $k$ record types $T_1, T_2, \ldots, T_k$. The $S_i$ ($i=1, 2, \ldots, \ell$) are obtained by applying $\Gamma$ to NODE.
1) no parent, OR successors (e.g., A)
2) no parent, AND successors (not shown)
3) OR parent, OR successors (e.g., b)
4) OR parent, AND successors (e.g., b)
5) AND parent, OR successors (e.g., d)
6) AND parent, AND successors (e.g., d)
7) OR parent, no successor (e.g., d)
8) AND parent, no successor (e.g., f)

Figure 29. Types of Nodes in an AND/OR Graph
The effects of these operations may be visualized as shown in Figures 30 and 31. Suppose that our information base consists of two record types A and B; such that A owns the in-set II and B is the member. Furthermore, assume that A has three occurrences \( (a_1, a_2, a_3) \) and that \( a_1 \) is related to \( b_1 \) and \( b_2 \) via \( II \). Figure 30b shows the graph that results from applying \( \Phi \) to an already existent node corresponding to record type A. In Figure 30c we begin with a node corresponding to record occurrence \( a_1 \) and apply the AND expansion to it. In Figure 31a we begin with the graph containing the record occurrence node \( b_1 \); application of \( \Gamma \) results in the linkage of a new node \( B \) to \( b_1 \) by means of an AND relationship. Figure 32b demonstrates the effect of applying the generalization operation to NODE in the graph of Figure 30a; it is assumed that, in the information base, NODE is an occurrence of record types C and D. Notice that the graph implies that NODE is also a record type in the information base, since it is related to its two successors in an OR fashion.

The above operations may be used to specify an algorithm that derives an AND/OR graph from a starting point (i.e., either a record occurrence or record type) and an information structure. In problem solving by problem reduction, Nilsson [63] observes that explicit graphs to search are rarely available; instead the graph is implicitly defined by an initial problem description and by reduction operators. He states that the problem solving process is accomplished by generating enough of the implied AND/OR graph to demonstrate that the start node is solved. The same observations can be made concerning the query processor's problem recognition activities. The query processor does not have explicit graphs to search.
Figure 30. Graph Generation Operations:

a. $\Gamma(\{C\}) = \{S_1, S_2, S_3, S_4\}$

b. $\delta(A)$

c. $\theta(a_1, II)$
a. $\Lambda(b_1)$

b. $\Lambda(\text{NODE})$ where NODE is an occurrence of record types C and D

Figure 31. Graph Generation Operators: II
Figure 32. An Information Structure Defining a Set of Problem Reduction Operators
Instead, a graph is implicitly defined by a decoded query (i.e., an initial problem description) and by an information structure (which effectively constitutes the set of permissible reduction operators of rules). The query processor utilizes an algorithm (which makes use of Λ, Θ and ¯φ) to generate the graph implied by the decoded query and the information structure.

The decoded query's command is taken as the starting point or root node of the graph. The information structure describes the permissible reductions which can be made for a given problem, be it a subsidiary problem or the initial problem description. The graph generation algorithm typically produces several reductions of an initial problem description. These reductions are generally not equivalent in terms of the solutions which they would produce. Each reduction corresponds to an alternative interpretation which could be given to the original problem description. As indicated in a previous section, selection or the interpretation which corresponds to the user's intention is based on interaction and/or analysis of the context in which the user's command is embedded. Once the interpretive ambiguities have been eliminated, the problem is said to be solvable. Information base contents, which are stored according to the information structure of the resultant reduction, are used to formally construct and execute the model implied by that reduction.

The basic algorithm for generating an AND/OR graph is as follows:

1. Let the starting point (the decoded query's command) be the root node of the graph.

2. Locate all record type nodes (i.e., nodes corresponding to record
types) \( N_1, N_2, \ldots, N_p \). \( N_1 \) does not have successors and \( N_1 \) was not created by generalization \((i=1, 2, \ldots, p)\). Examples of the method of location are depth first and breadth first searches.

3. If \( p=0 \), go to 5.

4. Perform \( \Phi(N_1) \) for \( i=1, 2, \ldots, p \). Go to 2.

5. Locate all nodes \( N_1, N_2, \ldots, N_q \) eligible for AND expansion.

6. If \( q = 0 \), go to 8.

7. For each \( N_i \) (\( i=1, 2, \ldots, q \)), perform \( \theta(N_i, \text{INSET}_j) \) \( \forall j = 1, 2, \ldots, J \). Go to 2. Here \( J \) is the total number of in-sets owned by the record type(s) of the occurrence corresponding to \( N_i \). Thus \( J \) is a function of \( N_i \) (e.g., if \( N_i \) describes an occurrence of two record types each of which owns two in-sets, then \( J=4 \)).

8. Locate all nodes \( N_1, N_2, \ldots, N_r \) eligible for generalization.

9. If \( r = 0 \), stop. The initial problem description has been reduced as far as possible, given the permissible reduction rules inherent in the information structure.

10. For each \( N_i \) (\( i=1, 2, \ldots, r \)), perform \( \land(N_i) \). Go to 2.

Consider the information structure shown in Figure 8 where lower case letters represent occurrences of the record types enclosing them. Upper case letters represent record types which are also occurrences of record types enclosing them. The arrows indicate logical in-sets; specific in-set occurrences are not depicted. Figures 33 and 34 illustrate three AND/OR graphs which can be derived from the information structure of Figure 32 by using the above algorithm. Notice that in each of the three cases the same set of reduction operators (defined by the information
Figure 33. AND/OR Graph Generation (I)
a. Root $Y$

b. Root $x_1$

Figure 34. AND/OR Graph Generation (II)
structure) was available. The difference among the cases results from different initial problem descriptions, each of which requires the use of different aspects of the information structure. A decoded query which requests the solution of problem X1 gives the graph of Figure 33. Similarly requests for solutions to problems Y and x₁ given the graphs of Figures 34a and 34b respectively.

These examples are useful for highlighting several features of the graph generation algorithm. First of all, it should be clear why the term "generalization" is used to describe the $\wedge$ operation. The application of $\wedge$ to Y1 creates a node for Y, which is AND-linked to the node at the end of every strictly OR path emanating from Y1. Recall that the semantics of an information structure state that every occurrence of a record type is an exemplification of instantiation of the concept represented by that type. It follows that all which is true of that concept is also true of its instantiations. In order to fully understand or resolve an instantiation, we must take cognizance of the general concept of which it is a special case. We do not, however, necessarily need to examine the siblings of that instantiation; it is for this reason that step two of the algorithm is concerned only with record type nodes that have not been introduced by generalization.

Referring again to the examples, it is readily apparent that the AND/OR graph could be drastically pared down, if the decoded query contains conditional information in addition to a command (e.g., SOLVE X1 USING $y_1$). Another feature of the algorithm is the uniqueness of the graph which it generates. Observe, however, that the information structure
cannot be recovered from any of the AND/OR graphs without additional information. Finally, we point out that it is entirely permissible, and in many cases quite desirable [63], for a node to have multiple parents.

Extensions to the Graph Generation Process

Finally we investigate extensions to the graph generation algorithm. These were omitted from the previous discussion for simplicity of presentation. Specifically, the extensions to be considered concern elaborations of the generalization and AND expansion operations.

In Figure 35a, D is a record occurrence of two record types: A and B. According to the present notion of generalization, all that is true of B in general is also true of its instantiation D. Similarly, all of the general features of A are also descriptive of its instantiations, including D. Application of ∧ to the graph node D results in the AND linkage of D to successor nodes A and B. The question arises as to whether the AND linkage is always appropriate in this situation of 'overlapping' record types. It is entirely conceivable that in the solution of a particular problem, D should be considered as an instantiation of B only (or A only); then again, the reduction of other problems may depend upon the fact that D is an instantiation of both types. In order to accommodate all of these possibilities, the ∧ operation should establish a logical OR relationship between node D and its successor record type nodes: A and B. However, the traditional AND/OR graph notation does not include a primitive construct to represent a logical OR. Figure 35b shows how the logical OR can be included in a graph by creating the artificial nodes \( \alpha \).
a. Overlapping Record Types

b. Representation of a Logical OR

Figure 35. The Overlap Problem
and $\alpha_1$. Note that this treatment requires a redefinition of $\land$ so that the AND connection is not made directly to the record type nodes. It is made instead to a single artificial node (e.g., $\alpha_o$) which has as OR successors both the record type nodes and a number of artificial nodes. If the number of pertinent record type nodes is $r>1$, then this number of artificial nodes is $q=\sum_{i=2}^{r} \binom{r}{i}=2^r-r-1$. Each of these $\alpha_1', \alpha_2', \ldots, \alpha_q'$ is the parent of an AND relationship to each of the $q$ possible (non-trivial) combinations of the $r$ record type nodes. If $r=1$, then $q=0$ and no artificial nodes are needed (not even $\alpha_o$). Redefinition of the $\land$ operation in this manner is still compatible with the graph generation algorithm. A more simple, though less flexible, approach is to disallow overlapping record types in the information structure and utilize the generalization operation as originally defined.

The technique described thus far permit the representation of recursion both in the sense of a program calling itself and in the sense of program X calling program Y, which in turn calls program X. The former is accomplished by means of defining an in-set whose owner and member record type is the same. The latter is accomplished by defining a pair of directed gates between two record types, such that each record type is the owner of one gate and the member of the other. Although it is outside of the present scope to examine recursion in detail, it should be noted that the use of two gates for this purpose is redundant. It is however forced upon us because of the nature of the AND expansion as previously described. That is, the AND expansion is a directed operation in that it proceeds only from an occurrence of an owner record type to occurrences of a member
record type. If the AND expansion were embellished (or a new operation defined) such that it also proceeded from an occurrence of a member rec-type to the associated occurrence of the owner record type, the need for redundancy is eliminated. The full implication of such an elaboration, for the semantics of an information structure goes beyond the issue of recursion and is outside of the present scope. Even though we may have the ability to define recursive models, it is of no consequence unless there is also a mechanism for the loading and execution of such models. Such a mechanism is not discussed in this paper. Hence the issues relating to the representation of recursion in the information base are pointed out here only as a possible topic for future detailed investigation.

Final Comments and an Example

The important point regarding the model formulation facility is that the knowledge of an expert, concerning ways in which models can be combined and utilized, can be captured in the information base. This knowledge is then available for use by users of the system who are not experts in modeling details. There are basically two methods for entering modeling knowledge into the information base. The first method is to perform this loading during the initialization of an information base; procedurally this corresponds to typical data loading with MML commands. The second method involves tracking and recording direct formulation as they are proposed (and verified). Those which are used repeatedly are candidates for preservation in the structure of Figure 24. It is a simple matter to use the DML to add or delete models; specification of new configurations
of models would require a dynamic restructuring of the information base logical structure.

The following example serves as a means for recapitulating the ideas introduced in this section on model formulation. It will later be used to illustrate the loading and execution mechanisms of the analysis facet. Figure 36 displays an example of the way in which total cost can be combined with elementary cost. In this logical structure we see that total cost (TC) has several alternative definitions (TC1, TC1, ... ) each of which is a record type in its own right. Alternative definitions of TC1, TC2, ... are not shown here. With respect to elementary cost (EC) there are several exemplifications (C1, C2, ... ). On a different level we see that C1 may be defined in either of two methods (M1 or M2). So the record type-record occurrence technique is used to represent alternative meanings. The two in-sets shown provide a different technique for representation of the ways in which total cost can be combined with elementary cost. This may be visualized at the occurrence level in Figure 37.

The necessity of the resolution level feature for the representation of conceptual knowledge should now be clear. Since model formulation depends on both the modification (i.e., choosing an alternative definition) and the combining of known modular models, the query processor must have some way to distinguish between the two. A straightforward method for permitting this distinction is to represent each by a unique technique (i.e., by a unique construct within the semantic net). Just as in the car example of a previous section, it would be possible to devise a strictly network logical structure to represent the same information that is in-
Figure 36. Example of Structural Techniques for Representing Knowledge about Modification and Combination of Models
Figure 37. Occurrence Example of the Technique for Representing Model Combinations
herent in the resolution structure of Figure 36; such a structure is shown in Figure 38. However, from the perspective of a query processor this structure is ambiguous, since the same construct (the in-set) is used to represent information about both the modification and the combination of modules. The relationship between TC and EC is functional, in that it describes how their occurrences can be meaningfully combined. The relationship between EC and M (although it uses an identical notational construct) is definitional, in that occurrences of M are various alternatives for defining the occurrences of EC. Such a structural ambiguity leads to asymmetric processing. The same notational construct must be processed differently and additional data must be stored to indicate which way each instance of that construct is to be processed. Conversely, since we are forced into using two notational constructs for a single semantic attribute (i.e., that of definition), the processor must possess two methods for arriving at semantically analogous interpretations of the relationships among pairs of modules. Finally we recall the previously mentioned inefficiencies of the network implementation relative to the information base implementation.

Analysis

Recall that this facet of decision support involves fitting the appropriate data to a formulated model, with subsequent execution of the model. Just as the decision facets of model formulation and perception are interrelated with the problem recognition facet, analysis and problem recognition are interdependent. The analysis is directed by the results of problem recognition. Conversely, the available facilities for per-
Figure 38. Example of Strictly Network Structure for Representing Conceptual Information
forming analysis restrict the practical utility of the problem recognizer; that is, an explicit problem statement, describing a model to be used and the data to be used, is useless without a facility for interfacing the model with the data such that execution may ensue. The query processor being considered here performs this interfacing automatically. Moreover, the specification of how the data requirements of a particular model are related to elements in a database, is not made in terms of DML extraction routines (as in the current implementation of the CPLAN system). The specification is made in terms of a comparatively high-level mapping language, discussed in Chapter 3.

The output of the analysis facet is the set of facts or expectations which the user intended to produce. It is up to the user to evaluate the facts and expectations, produced in a series of queries, and subsequently make a decision. Automatic performance of the data-model interface is based on a query's deepest structure (as produced by problem recognition) and on information stored in occurrences of program record types appearing in the formulated model. The two primary activities of this analysis process are the loading of program modules and the construction of a command to initiate execution of the model. The composition of program record types and occurrences is considered first. Next comes an examination of the requirements for storing and utilizing information maps. Finally there is a description of the loading and execution activities.
Program Record Types and Occurrences

Figure 39 displays the make-up of a hypothetical record type named PROGRAM-X. The data item types NAME and DESC are self-explanatory. Present design of the analysis activity requires that all program modules be stored in the information base in the form of relocatable code. But this code is not stored in the record occurrences per se, because of certain buffering inefficiencies which eventuate as a record occurrence becomes large. The location of the code for a particular record occurrence is indicated by the value of the corresponding data item occurrence of PTR.

Each program module is treated as a callable subprogram, which has all information about its external return jumps and all information about I/O areas in a parametric form. Suppose that program occurrences of the type PROGRAM-X have a maximum of N external calls; then for a particular module, values of the item occurrences of EXT 1 through EXT N specify the ordering of externals in that module's parameter list. There are other methods for storing this information; the above method is chosen for simplicity of presentation. An approach that is more storage-efficient, but less flexible, utilizes in-set's logical ordering facility; it can be used only for simple structures like that of Figure 28a, but not for structures akin to Figures 28b and 28b. Another approach, which is more flexible, entails storing the ordering in occurrences of gates which mediate program record types.

An occurrence of the SCHEMA-KEY item type gives the information base key (i.e., address) for locating the module's schema (i.e., the information structure used by that module). This is consistent with the CODASYL notion
<table>
<thead>
<tr>
<th>RECORD</th>
<th>PROGRAM -X</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>ITEM</td>
<td>NAME</td>
<td>name of the record type</td>
</tr>
<tr>
<td>ITEM</td>
<td>DESC</td>
<td>item type denoting name of an instantiation of PROGRAM-X</td>
</tr>
<tr>
<td>ITEM</td>
<td>PTR</td>
<td>description of the instantiation (used for interaction with user during problem recognition)</td>
</tr>
<tr>
<td>ITEM</td>
<td>EXT 1</td>
<td>pointer to relocatable code</td>
</tr>
<tr>
<td>ITEM</td>
<td></td>
<td>ordering of externals in a program module's parameter list</td>
</tr>
<tr>
<td>ITEM</td>
<td>EXT N</td>
<td></td>
</tr>
<tr>
<td>ITEM</td>
<td>SHEMA-KEY</td>
<td>information base key of the program module's schema</td>
</tr>
<tr>
<td>ITEM</td>
<td>IMP 1</td>
<td>the K input maps for a program module (containing target specifications only)</td>
</tr>
<tr>
<td>ITEM</td>
<td>IMP K</td>
<td></td>
</tr>
<tr>
<td>ITEM</td>
<td>OMP K+1</td>
<td>the M output maps for a program module (containing source specifications only)</td>
</tr>
<tr>
<td>ITEM</td>
<td>OMP K+M</td>
<td></td>
</tr>
</tbody>
</table>

Figure 39. Structural Composition of a Program Record Type
of a subschema capability. A very compact information structure for storing these schemas is presented in Figure 13. The usage of input maps (IMP) and output maps (OMP) is described later. In the event that it is not already evident, we note that program record occurrences always refer (via PTR) to a module of relocatable code. But a program record type does not refer to a module of code unless that record type is also a record occurrence of some other record type.

Information Mapping

Information mapping refers to the transferral of information organized according to one structure (the source) into another structure (the target). The mapping language and processor to perform the actual data transferral were documented in Chapter 3. A command (i.e., a map) in the mapping language specifies the source, the target, and the correspondence to be made between elements (e.g., data item types, record types, in-sets) of a substructure of the source and elements of a substructure of the target. All program module input (other than that in a parameter list) and output is handled by maps. A given module's schema forms the target of the module's input maps and the source of its output maps.

An input map for a program module defines the mapping of data from some source into that program's schema. In general a module may have many maps (i.e., many data sources), just as the usual Fortran program may have several tapes of input data. However, when storing an input (e.g., an IMP in Figure 39) we do not know the source specification, since this varies according to the type of analysis which the user desires. Thus, source portions of all input maps are null in the information-base. It is in
conjunction with problem recognition that source definitions for all of the program's input maps are made.

The possible sources are of two types. The first consists of substructures of the information base. The second type of source consists of temporary structures which may actually be used to hold data or which merely serve as templates for a virtual mapping process. With respect to the first type of source it must be remembered that the source for an input map can vary, according to the type of analysis which the program is intended to perform. If the program were always used in the same way, the source would of course be static. We shall consider the more general (and realistic) case here. Remember that problem recognition results in the identification of a particular type of analysis. Thus if we store source map definitions with each type of analysis, these can be directly inserted into the program's input maps (i.e., the input maps with substructures of the information base as inputs). There is still a difficulty however. How can we handle the type 'one' sources for all programs which are subsidiary to (either directly or indirectly called by) the primary program? These sources are also subject to change depending on the type of analysis to be performed. This information can also be stored with the type of analysis.

The second type of source consists of data generated by a subsidiary program; this data must be organized according to some structure. This structure serves as both the source for input maps of the calling program and as a target for output maps of the called program. Stored in the gate (that mediates the two modules), this structure is used to fill null areas
of these input and output (OMP of Figure 39) maps prior to loading the relocatable code. Observe that just as a program's input maps have null source specifications, their output maps have null target specifications. And just as there are two types of sources, there are two types of targets: substructures of the information base and temporary structures to pass data among programs (or to the user). Targets of the first type may be filled in a manner analogous to sources of the first type.

Having described the mapping process for passing information from a subsidiary program to the program that calls it, now consider the reverse process (which is handled in a similar fashion). That is, in the gate we store source specifications for input maps of the called programs to be used independently (i.e., as a main program, not a subsidiary); in this event the program's input maps, which would have otherwise related it to a calling program, must have their source specifications associated with the type of analysis which is being requested.

Loading and Execution

In order to carry out an analysis there must exist capabilities for the transfer of information and for the transfer of control. There are three kinds of information transferral. The simplest involves the transfer of (typically low volume) data values, which have been specified in a user's query (or the follow-up interaction), to the program module(s) which make use of them. This transfer is accomplished via parameters in the execution command. An execution command may be thought of as a calling statement which transfers control to the loaded code for execution.
The preceding portion of this section dealt with the other two varieties of information transferral: the transferral between two modules and the transferral between a module and the information base. Both involve source and target structures. Some of these structures may be used to define 'scratch' areas which are assigned such that there is consistency in the naming of sources and targets among the mapping statements of the various modules. The sources and targets to be used by a particular module are passed to that module when it is called. An I/O operation for a module is handled by a call to the mapping processor, using the parameterically designated source and target as well as the subschema correspondences between the two.

With respect to transfer of control, the externals to be used by a particular module are passed to that module when it is called. After the loading of modules, the address of each is known and can therefore be inserted into appropriate positions of the execution command's argument list. Recall that these positions are implied by the relevant occurrences of the item types EXT1, ..., EXTN. Thus the handling of control transfers is much less complex (assuming that recursion is disallowed) than the treatment of information transferral.

The analysis activities described to this point (let us designate them by \( \Psi \)) are illustrated in the following example. Recall the query which results in a deep structure corresponding to \texttt{RUN TC FOR CL = M1(.,3)}, \( TC = TC5 \). This may be expressed as \texttt{TC(TC5,M1(.,3),.)} which is called the deep* structure. The elements within a pair of parentheses denote resources utilized by the program whose name immediately precedes the
parentheses pair. The meaning of each element is determined by its position in the argument list. In this example the first position refers to the abstract resource is to be used. The succeeding positions identify the resources to be used by this version of TC. From the information structure (see Figure 37) we know that there must be three and that their order matches the ordering the member occurrences the set SI. Thus the second position indicates that we are to use version MI of the abstract resource "Cl". In this example C2 and C3 have only one version apiece; the blanks for positions three and four indicate defaults to the only possible versions of these two resources. No data inputs from the query are used by TC5 itself, although it may be tied to certain data inputs from the information base via a map command. But MI does accept a data input from the query. Note that if MI were itself an abstract resource, then the first position within its parenthetical expression would identify the version to be used.

The Ψ process may be indicated by TC(TC5,MI(3),)→addr_{TC5}(W,X,Y,Z) where W,X,Y,Z, are parameter lists. W is a set of control parameters which define the sizes of X,Y and Z. X describes the external jumps in terms of addresses and parameter lists to be passed; thus X has the form:

addr_{1}(W_{1},X_{1},Y_{1},Z_{1}), addr_{2}(W_{2},X_{2},Y_{2},Z_{2}), ... Each X_{i} (i=1,...,k) can have a form corresponding to X, where k is the number of externals called by the subprogram with X in its parameter list. Alternatively X_{i} is null in the case where no externals are called by the subprogram with X_{i} in its parameter list. Parameter list Y is simply an assignment of I/O devices to be used by the subprogram. Each subprogram has a (possibly null)
parameter list Z. The list contains all data parameters which have been specified in the query for use in this subprogram. In the above example, Z is null for the subprogram TC5, but Z has one element (i.e., .3) for the subprogram ML. So a more explicit expression of $\text{addr}_{TC5}(W, X, Y, Z)$ is:

$\text{addr}_{TC5}(W_{TC5}, \text{addr}_{ML}(W_{ML}, Y_{ML}, .3), \text{addr}_{C2}(W_{C2}, Y_{C2}), \text{addr}_{C3}(W_{C3}, Y_{C3}), Y_{TC5})$. In this expression details of the control and I/O parameters are not shown. Execution output is displayed at a terminal or routed to a printer. The user also has the option of saving output in the information base.
CHAPTER 7

CONCLUSION

The foregoing observations on a generalized intelligent query processor for decision support have involved the introduction of some novel concepts, as well as suggestions for methods of design and implementation of such a system. Presently the system as described in this paper, is partially implemented. The use of this partial implementation for decision support in several application areas has been investigated (e.g., water quality management [45], picture processing [23], mathematical programming [19], inventory management [17], nuclear power plant evaluation [18]). Full implementation is not presently feasible, given the constraints of academic computer resources.

In the face of the rapidly increasing complexity of societal problems, the study and research of methods for aiding decision makers who must deal with these problems is of paramount concern. It is this concern which motivated the foregoing dissertation. The approach taken for investigation of this issue was of a dual nature, identifying and integrating elements from the fields of artificial intelligence and data base management with the result of a framework for generalized and intelligent decision support systems.

All studies concerned with information processing should be cognizant of the extensive work that is being (and has been) undertaken in the discipline of data base management [47]. The notion of an information
base involves extensions to the CODASYL approach to database management. This approach is the basis for several prominent commercial systems (e.g., TOTAL, IDMS) and has received consideration as a basis for a database management standard [34],[32]. In addition the information base serves as a semantic net.

Research was previously cited whose aim is to develop "...more intelligent computer software systems for complex dynamic activities that require integrated perception, analysis, planning, and problem solving" [68]. In a sense, this goes a step beyond the current ambitions of the GENI query processor, in that it involves actually making decisions that have continuity in an ongoing, dynamic situation. The GENI processor is intended to support a user's decision activities according to that user's request; it is left to the user to furnish (or depart from) continuity in the series of queries which are posed. The GO study endeavors to understand human cognitive activities in a specific (but complex) task domain, using skilled human GO players as models. The GENI study also attempts to understand and emulate human cognitive activities, though from a perhaps more general perspective.

In summary, this presentation has examined the subject of a generalized intelligent query processor for decision support. This processor is invariant to changes in the decision making context within which it operates. It utilizes both empirical and conceptual information (which are retained in a single mechanism) in order to understand the implications of a query, to formulate a model for the query's solution, and to execute this model with the pertinent data. The user's non-procedural,
English-like query may be viewed as the specification of a program to be executed; the query processor may be viewed as a high-level compiler which automatically generates code in a procedural, programming language.

Avenues for future research run in the directions of 1) extending the query language to take full advantage of the semantics supported by the information base; 2) refinement and elaboration of the underlying conceptual framework of decision support; 3) utilization of the information base to support computer graphics and model data base machines; 4) investigation of the use of the GENI decision support system in a logically distributed environment; 5) incorporation of the evaluation facet into the GENI query processor; 6) examination of the meaningfulness of optimality with respect to information base design; and 7) study of the relationship between predicate calculus quantifiers and the information base constructs.
BIBLIOGRAPHY


15. Bonczek, R. H., Haseman, W. D. and Whinston, A. B., "Relational vs. Network Data Base Implementation," Krannert Graduate School of Management, Purdue University, August 1975.


44. Haseman, W. D., Holsapple, C. W. and Whinston, A. B., "Implementation


