CHARTMAKER: A "True Consultant" Expert System for Designing Charts

by

Thomas Aaron Shulok

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Computer Science and Applications

APPROVED:

John W. Roach, Chairman

Sallie M. Henry

Harold A. Kurstedt

June, 1988

Blacksburg, Virginia

CHARTMAKER: A "True Consultant" Expert System for Designing Charts

by

Thomas Aaron Shulok

John W. Roach, Chairman

Computer Science and Applications

(ABSTRACT)

Expert system technology has produced systems that perform heuristic classification. These systems solve problems of a type determined by the knowledge engineer and the expert at system design time. A "true consultant," on the contrary, applies domain knowledge to solve a problem not previously seen. For example, a graphic design consultant must accept the statement of almost any problem from a client and turn it into a visual design. This thesis reports the successful construction of the first such true consultant for a well-understood domain: the visual design task of chart construction. The system leads a client in a dialogue to define a problem in the client's terms and then maps the problem representation into a knowledge base for constructing charts. Extensions of the technology reported in this thesis may aid the creation of a new class of expert systems.

Acknowledgements

This work is the culmination of 18 years of academic study. Such a long trek would never have been possible with out the support of my parents, . To them I owe both my education and my continuing desire to learn. This thesis would never have been possible without the guidance and support of John Roach, my advisor. His insight and direction provided the help I needed to persevere when the road became rough or disappeared altogether. Special thanks to and who served as our graphical experts. Thanks also to for getting the whole thing started and to for reasonably unlimited use of his Macintosh.

Table of Contents

List of Illustrations

1.0 Introduction

A "true consultant," in the sense used in this thesis, allows a client to define the problem of interest and then uses general knowledge about the problem domain to solve the client's problem. For example, when a student brings a problem to a programming consultant to get help with a bug, the consultant already knows a great deal about programming and the language used by the student but may never have seen the particular program in question. Before the consultant can provide any help, the programmer must first explain what the program is supposed to do, how the bug manifests itself, and perhaps the status of certain variables at the time that the bug appears. The client's problem is not completely defined before the actual consultation begins. By contrast, in typical expert system applications, all the problems that the system can handle are defined during the knowledge-engineering phase, and the system produces advice by following a decision tree guided by a user's answers to a set of pre-defined questions.

Without an adequate problem definition phase which allows the client a reasonable latitude of expression, it will be impossible to construct a programming consultant or any other consultant that relies heavily upon the user's conception and description of a problem. The cause for this failure centers upon a weakness in a fundamental theoretical aspect of expert system design: knowledge about problems worth solving (including problem definition) and background knowledge of the domain itself have been combined in the primary representation technology of expert systems, rules.

[Aikins, 1983] suggests that rules combine knowledge and control in a detrimental fashion since rules contain both the knowledge about the domain as well as the underlying sequence of the consultation. The discovery that rules (and rule/frame systems, too) combine different forms of an expert's knowledge is of much greater interest. Without a solid representation framework upon which to accurately build a definition of the client's problem, no expert system will ever attain the status of a "true consultant".

While conventional expert systems rely exclusively on if/then rules, this thesis explores and develops an alternative approach to building expert systems: building a user-directed problem model and then mapping that model into background domain knowledge. By creating a problem model and avoiding reliance on some variation of tree structures typical of current expert systems, more of the user's view and intent can be captured, thus expanding the role of expert systems into a much broader range of application.

More generally, this thesis hypothesizes that the expert consultation problem will be solvable not by some unifying theory of representation, but by classifying problems into broad categorizations such as design problems, diagnosis problems, discovery problems, etc. Instead of attempting to fit all problems into some generic representation, our theory suggests that the model of expertise should reflect the intrinsic nature of the problem domain. For example, a theory of diagnostic problems might include a well-known set of problems (bugs) and fixes, or a technique to compare intended versus actual behavior for problems not part of the problem set (classification scheme). A natural choice for diagnostic problems is some hierarchical structure traversed to find a solution and a model-based component for harder problems. Other problem areas require a different theory. Design problems, for example, are too "open-ended" to permit codification in a decision tree. Moreover, design problems are usually more "human-oriented" in that individual preferences play an important part in the consultation; the same situation may be perceived and thus defined differently by different people. Any effective model in this domain must possess flexibility sufficient to account for these perceptual differences, which -- by their nature -- prohibit an exhaustive or reasonably complete codification of knowledge prior to a consultation. To achieve this end, we

explore the construction of a problem definition framework for a particular design problem area; in the area of graphic arts we have built a chart-construction system for numerical data called CHARTMAKER. The charting domain provides useful insight into the design domain and serves as an adequate vehicle for demonstrating the capabilities of the automated true consultant. Moreover, the charting domain for numerical data is well-understood [Spear, 1969] [Tufte, 1983] [Enrick, 1972] and less subjective than other design areas such as architecture, so it avoids the detailed complexity which could obscure the basic theoretical issues that are the focus of this thesis.

1.1 Simple Example

A brief example will help illustrate the basic principles of the automated design consultant. Consider a user who is making a presentation to the salespeople in his division. He wants to increase his division's sales by implementing a new bonus incentive program; however, he is not sure how to best graphically convey the message that the salespeople can increase their income if they increase their sales.

The first step in the consultation is to create the problem model. At the outset of a consultation with CHARTMAKER, the user is presented with a menu that allows him to construct a logical relationship diagram showing the basic concepts of the problem model and how they relate to one another:

- ADD a concept to the model
- DROP a concept from the model
- LINK two concepts (causal)
- SNIP (delete) a link
- DISPLAY current problem model

- FINISH building the model

After some consideration, the user decides his situation consists of four basic issues: profit, bonus, sales, and sales force. He then ADDs these concepts to the model. After further consideration, he determines exactly how these concepts affect one another. In his estimation, the sales force has a direct impact on sales, and sales in turn have a direct impact on company profit. Moreover, company profit has a direct influence on the amount of the bonus paid to -- and having an obvious effect on -- the salespeople. He adds these causal influences to the model by LINKing the concepts together. If he wishes to see what the model looks like, he can DISPLAY it. The relationships are listed by the concepts they affect. For a given concept, both the "influences" and the "is influenced by" links are given. The diagram of the situation model is shown in Figure 1 on page 7.

Once the user is satisfied with the model, he selects the FINISH option, thus completing the first phase of the consultation. He is then prompted to identify the key concepts of the model. A key concept is one that the user really wishes to emphasize in the presentation. For the example problem, the user decides the key issues in the presentation are both the sales and the bonus, and so chooses these as the key concepts. Once the key concepts are identified, the automated consultant investigates how the key concepts affect one another by analyzing the transitive closure of influence in the original model and producing a reduced model consisting of the original situation model with highlighted key concepts and their derived interrelationships (Figure 2 on page 8). From the reduced model, the system then determines exactly what the user is trying to demonstrate. Since the concepts have mutual causation links (after transitive reduction of the initial model), the implied goal is the "relationship of two variables". This is intuitively satisfying since a relationship chart will illustrate the mutual causal effect of two entities. For example, a chart showing the relationship between boiling point and pressure shows the impact a change in one entity has on the other. If the causation effect transpires in only one direction, a relationship situation is nonetheless implied. For example, a chart showing the impact of years of education on annual salary is still

an instance of the relationship of variables, but the causality is unidirectional. In this case, number of years of education is the independent variable, so it would be automatically labelled on the x axis.

Before rendering a chart, CHARTMAKER needs to elicit the raw data from the user. CHARTMAKER uses the key concepts of the model to form a data pattern or template, and the user enters the data one line at a time. For the sales example, the data pattern is:

bonus	sales

and the user enters:

100	30000
150	30500
200	31000
230	31500
250	32000
260	32500

From the goal model and the raw data the automated consultant creates a rendering model. The rendering model (Figure 3 on page 9) contains chart-specific information, such as axis labels, interval spacing, and even the particular type of chart to be constructed. In constructing the rendering model, the system relies on background charting knowledge. For example, even when the system knows that it is plotting a trend-type chart, it must still decide between a line chart and a column chart since the nature of the raw data often indicates a particular chart type. In this case, the decision depends on the number of sampling intervals, and if only a few intervals are available, the automated consultant will select a column chart.

The primary issue in this thesis is the application of background knowledge to a freely-defined problem. To accomplish this, the "true consultant" approach focuses on the separation of different types of knowledge and the structure of the expert's domain. While a conventional expert system has all its knowledge contained in if/then rules, the true consultant has knowledge about interviewing clients, focusing on the appropriate areas of concern for a given problem, and finding solutions by classifying problems. By enhancing the expert system's knowledge, the client can describe his problem with greater independence from the preprogrammed approaches that dominate the field today and communicate with the expert system much as if it were a human expert. The design domain helps illustrate the necessity of a situation model while the charting domain limits the complexity of underlying design issues which are not well-understood. By designing the system to follow supporting domain methodology (as determined by actual experts), the expert system behaves more like a human expert since they both share the same type of information about the domain. Incorporating this expertise into an expert system requires an understanding of the human expert's expertise -- how he approaches a problem situation, how he defines the client's objectives, and how he synthesizes a solution. Once these principles are derived, the expert system can be built based on these domain principles, in effect "building" a new expert. Success in a simpler facet of the design domain lays the framework for the entire class of design problems. The approach presented in this thesis may be expanded to incorporate more complex areas of design based on more complex situational models and more detailed background knowledge.

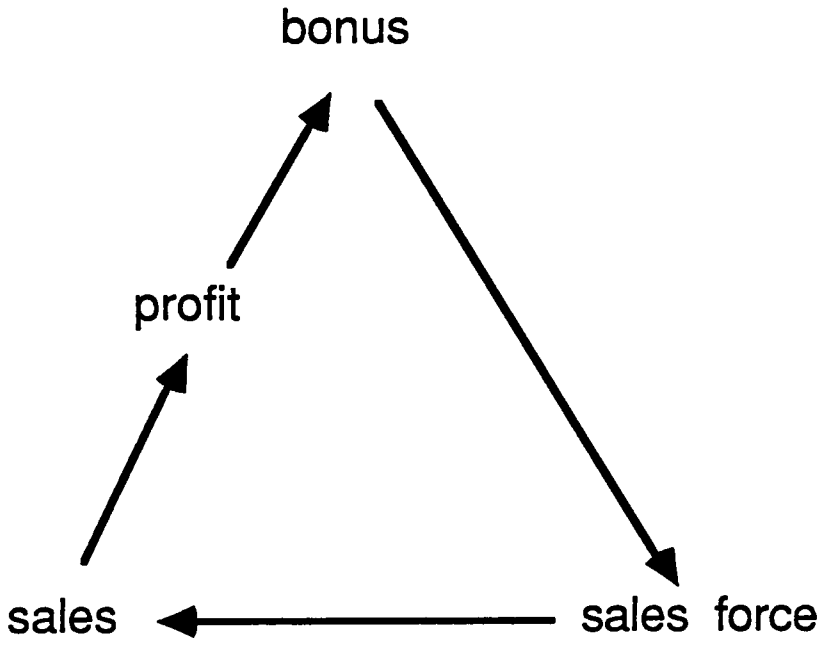

Figure 1. The Problem Model for Sales Example

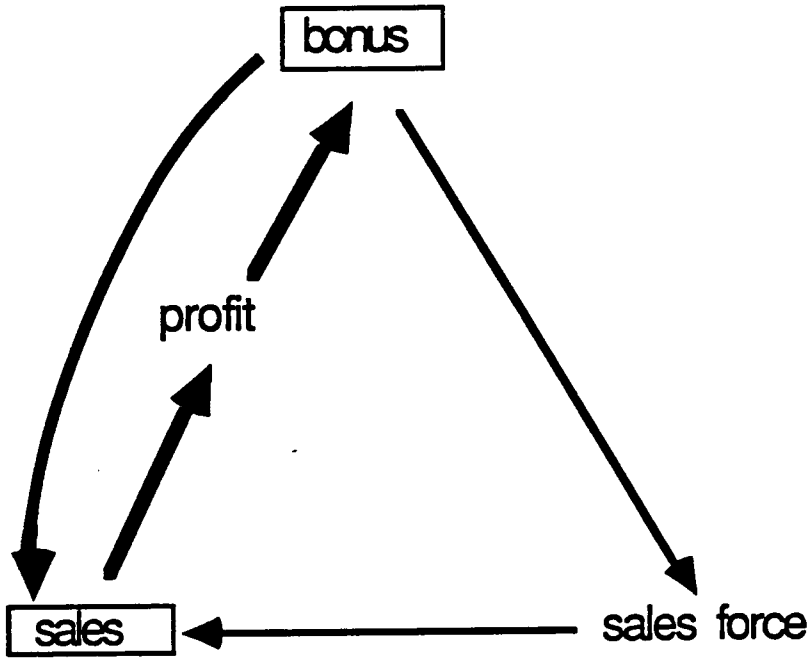

Figure 2. The Goal Model for Sales Example

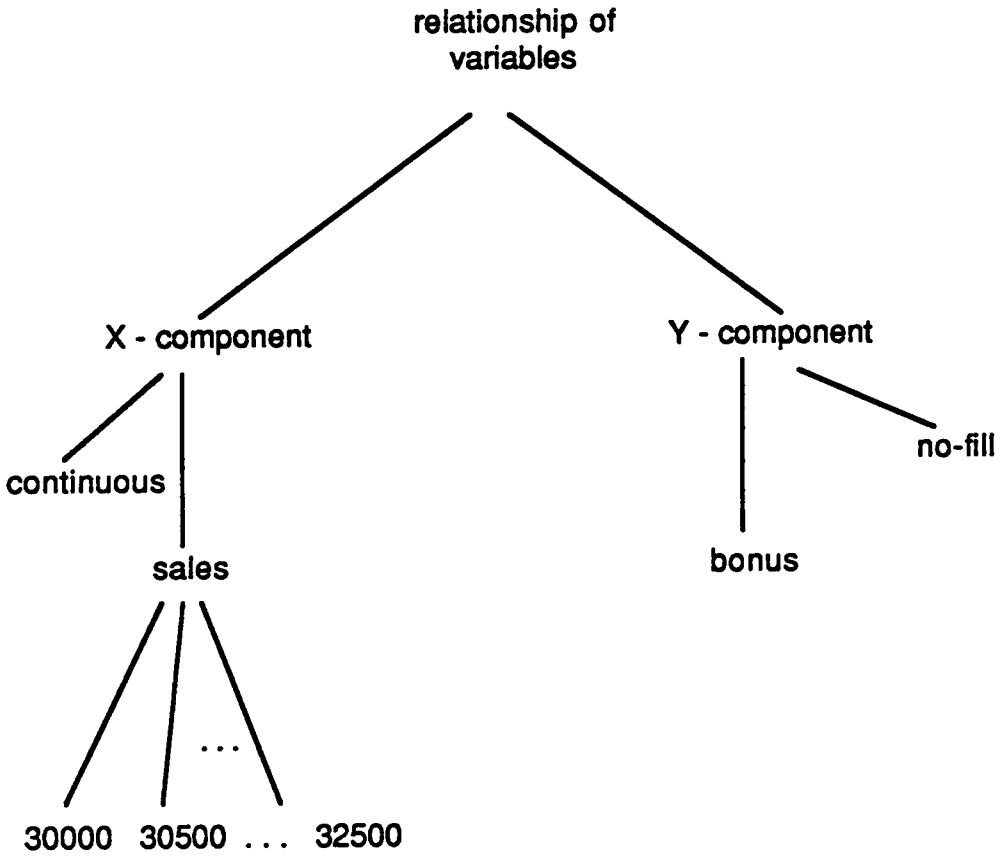

Figure 3. The Rendering Model for Sales Example

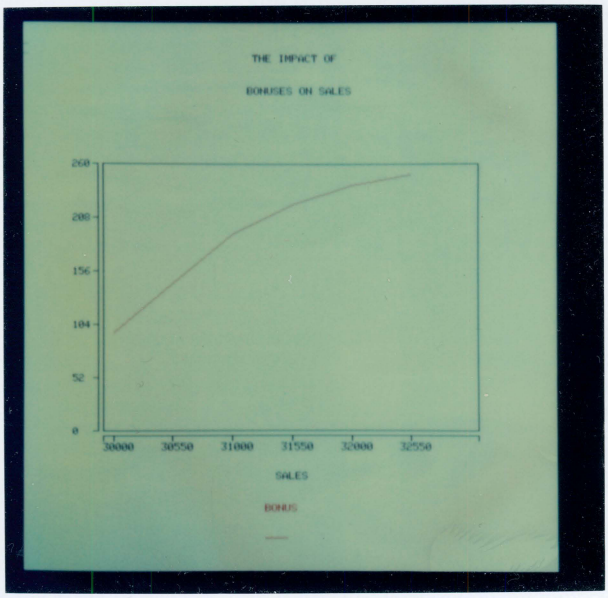

Figure 4. The Rendered Chart for Sales Example

2.0 Literature Review

The foundation of this thesis draws from several diverse areas. The fundamental contribution of this work is a new approach to computer-based problem definition and solution. The motivation for this work derives from the ongoing difficulty in defining a human problem or situation in a manner that a computer can both understand and manipulate, yet retain all the information vital to the actual problem. Already, a tremendous amount of theoretical and practical work has been devoted to this particular facet of artificial intelligence (AI), and any new contribution must be measured against previous work -- both theoretical and practical. Beyond basic AI theory, one must consider generic design theory. Since the true consultant paradigm discussed in this thesis is intended for the design domain, it should reflect basic design considerations as well. Finally, the software implementation of this approach operates in the charting domain, so some consideration is given to more specific design issues concerning chart types and their suitability for given contexts.

2.1 *Problem Definition in Artificial Intelligence*

At this juncture, one might wonder why more "human-like" computer systems are not currently in use. The primary reason centers on the theoretical basis of knowledge representation -- that is, if the system is to behave like a human, it must store its knowledge in some analogous manner. The knowledge representation problem is not easily overcome. By definition, knowledge representation involves the codification of information to correspond to some state of the world. The goal of AI is to have an intelligent machine reason with and draw new conclusions from these symbolic representations. So far, all current schemes fail to truly fulfill this deceptively simple goal. The earliest attempts at formalization of knowledge can be traced back to the time of Leibniz who used predicate calculus to represent and manipulate ideas. The explicit approach has endured and is best seen in today's knowledge-based systems which employ explicit knowledge bases and logical rules of inference to reach conclusions. The popularity of explicit formulation is best expressed by Brian Smith's Knowledge Representation Hypothesis:

> Any mechanically embodied intelligent process will be comprised of structural ingredients that a) we as external observers naturally take to represent a *propositional* (my emphasis) account of the knowledge that the actual overall process exhibits, and b) independent of such external semantical attribution, play a formal but causal and essential role in engendering the behaviour that manifests that knowledge [Smith, 1982].

If a system is to have an explicit knowledge base, it must have some well-defined language to express that knowledge as well as meta-statements, or rules of inference, to combine and transform statements into other statements. Moreover, a successful system should also be able to derive implicit knowledge from the representation -- automatically drawing inferences from the *structure* of the knowledge. An intelligent system's power rests in its knowledge representation scheme which details the knowledge of its domain. The representation problem can be reduced to three primary areas: the representation language, the inference regime, and the domain knowledge. This delineation, however, does not obviate the problem. While the areas may be well defined, the approach to take in any of them is not. Consider, for example, the role of predicate calculus in knowledge representation. While predicate calculus may work well in the realm of symbolic logic and math-

ematics, its application in knowledge representation is not as straightforward. The aim of knowledge representation is to capture the semantics of knowledge -- natural language concepts and plausible psychological models expressed in the language. The core of the belief management problem lies in the derivation of implicit beliefs from explicitly stated knowledge. The question of how the knowledge should be represented is related to where the knowledge comes from and how it is acquired. There are three fundamental reasons why these areas are related:

1. because the chosen representation may affect the acquisition process

2. because the acquisition process can suggest useful representations (tools exist that build up knowledge structures from dialogs with human experts), and

3. because it is possible that some of the knowledge that a system is to use should stay in the form in which it is available. [Tanimoto, 1987]

Since knowledge acquisition is an integral part of knowledge representation, an effective representation strategy should exhibit some attention to these concerns. While the ability to store and manipulate knowledge represents a fundamental human ability, it remains a major point of difficulty in knowledge representation, and thus, effective expert system design.

This difficulty manifests itself as several areas of concern. Expressive adequacy focuses on the semantic content conveyed by the representation. Reasoning efficiency deals with the system's ability to draw inferences from its explicitly represented knowledge. Typically, the more expressive the language is, the lower its efficiency. Incompleteness involves the system's ability to cope with underspecified situations. While validation is an important issue in knowledge representation, some systems employ non-deductive reasoning techniques, and it can be difficult to validate the system's reasoning behavior since it cannot be formally verified. Finally, when representing certain agents, it is also necessary to reconcile non-concrete issues like their beliefs, attitudes, and dispositions.

There are, of course, other issues, but these are particularly relevant to this thesis [Levesque, Brachman 1985].

Since there are many diverse concerns surrounding the knowledge representation problem, it is not surprising to find a variety of proposed solutions which follow Smith's hypothesis in that the underlying representational structure defines a problem in propositional terms. Methods that employ formal logic are obviously propositional in nature, but even problem representation structures such as semantic nets that appear to be fundamentally different, are still based on a propositional foundation.

2.1.1 Logical Calculi

Formal logic has formed the cornerstone of knowledge representation in contemporary artificial intelligence. The majority of knowledge representation methodologies employ some form of logical calculus. The mathematical nature of formal logic permits formal verification of the knowledge base. Moreover, formal logic has an extensive theoretical framework and has been used for centuries to represent concepts and relationships between them [Tanimoto, 1987].

Unfortunately, formal logic is no panacea for the knowledge representation problem. Formal logic is an excellent classification system, but provides no facility for exceptions or defaults. While classification is vital to knowledge representation, exceptions and defaults are commonplace in the real world, and an effective representation scheme must have some facility to express them. Associated with the problems of exceptions and defaults is the necessity to explicitly express all the system's knowledge -- a tedious, often difficult process. Beyond this, there are also problems in determining what the system actually knows. The closed-world assumption underscores a major deficiency of formal logic in knowledge representation: if a concept is not explicitly coded into a system's knowledge base, does the system assume the concept is false or does it recognize that it *doesn't know*

if the concept is true or false? If the system does the latter, then it must have explicit knowledge of everything it knows to be false, and if it does the former it risks inconsistency since currently unknown facts may be derived later. Additionally, formal logic can represent only propositions that are either true or false, so concepts such as "possibly" or "maybe" that are common in the real world cannot be expressed as logical statements. In spite of its shortcomings, formal logic remains the principal basis for knowledge representation, primarily because no other formal system exists.

Formal logic is the foundation for most contemporary knowledge representation techniques: predicate calculus, rule bases, frames, semantic nets, and modal logics. Predicate calculus represents information as variables, quanitifiers, and predicates, and it is an attractive choice because well-known inference techniques can be applied to the representation to produce formally verifiable results [Charniak, 1986] [Nilsson, 1980]. Rule bases are composed of if/then production rules that are derived from the implication connective in predicate calculus. In general, a rule consists of an antecedent, a set of conditions that must be satisfied, and a consequent, a set of actions that are performed when the antecedent is satisfied. The knowledge of most conventional expert systems is contained in rule bases [Tanimoto, 1987]. Frames are a way to group and organize predicate calculus statements. By grouping related statements, a frame actually represents a partial context. For example, the frame for "living room" could contain information about the objects in the room, such as a television and a couch, as well as information about the purpose of the room, its location, and its size [Charniak, 1986] [Tanimoto, 1987]. Semantic networks were originally developed to represent the meanings of sentences in terms of objects and their interrelationships. In a semantic net, the concepts are represented as nodes, and the relationships are represented as links between the related nodes. Semantic nets are based on logical formalism, and they share the same representational deficiencies associated with formal logic [Sowa, 1984] [Charniak, 1986]. Modal logic is an extension of predicate calculus that permits concepts such as necessity and possibility. In modal logic a concept can be possibly true, necessarily true, true, or false. The rationale for this extension is to reduce the "black or white" rigidity of standard formal logic. Modal logic still suffers from the other limitations of formal logic, as well as a more complex inference regime [McCarthy, 1985].

A variety of representational techniques exist for knowledge representation, and these form the theoretical basis for conventional expert systems. The basic expert system paradigm, if/then rules, is derived from predicate calculus, so the limitations associated with predicate calculus are limitations of the expert systems. These limitations are manifested as restrictions in potential domains for expert systems. Systems that rely on predicate calculus require well-defined domains since all the knowledge they have must be explicitly coded into the knowledge base. One domain where predicate calculus-based formulations have not succeeded is the design domain, and the reason for this failure stems from the basic nature of design activity.

2.2 The Design Domain

In its most general form, the process of design represents a progression from the abstract to the concrete. For example, a newly constructed house is the concrete embodiment of the preferences and constraints of the architect, builder, and owner of the home. The design process in this example, starts with the owner's initial concepts as to what the house should look like and what features it should have, to the architect's constraints regarding the feasibility and integrity of the structure, and finally to the builder's concerns about appropriate building materials and techniques. Through the design process, what started as some vague ideas and preferences has become an actual entity which embodies those original ideas as well as the preferences and constraints of others involved in the design process. While there are differing opinions about the details of design theory, it can be reduced to basic ideas that reflect the transition from the abstract concepts to the concrete objects.

2.2.1 Design Methods

Attempts at the systematization of the design process began in the early 1960s. The Conference on Design Methods held in 1962 is considered the seminal event in the field, and two fundamental ideas arose from the conference [Cross, 1977]. The first was a comment made by J. Page [Page, 1963], that "there only seems to be one common point of agreement, and that is that systematic design is a three stage process, demanding analysis, synthesis and evaluation." The second contribution was Jones's "Method of Systematic Design" [Jones, 1963] that was the first attempt at a unified system of design as a combination of intuition and experience with rigorous logical treatment. The goal of his methodology is to externalize all the logical activities into charts, lists, and diagrams to leave the designer's mind free to produce solutions based on ideas, hunches and guesswork -- the creative component of design.

The methodology is delineated as:

1. Analysis:
    a. Random list of factors,
    b. Classification of factors,
    c. Sources of information,
    d. Interactions between factors,
    e. Performance specifications,
    f. Obtaining agreement;
2. Synthesis:

a. Creative thinking,

b. Partial Solutions,

c. Limits,

d. Combined solutions,

e. Solution plotting;

3. Evaluation:

a. Methods of evaluation

b. Evaluation for operation, manufacture, and sales.

Jones's methodology was accompanied by a warning that design is in practice, an "iterative muddle", and that the systematic approach is best used as a guideline.

About the same time, Asimow published a detailed procedure for engineering design that he referred to as "the morphology of design" [Asimow, 1962]. The morphology "refers to the chronological structure of design projects," and is defined by the phases and constituent steps of the design process. The phases of the process are analysis, synthesis and evaluation. The constituent steps of each phase are far more detailed and complex. Asimow further delineated the design life-cycle into seven phases:

- Primary Design Phases:

1. feasibility study,

2. preliminary design,

3. detailed design.

- Phases related to the production consumption cycle:

1. planning for production,

2. planning for distribution,

3. planning for consumption,

4. planning for retirement.

A few years later, a more rigorous method was developed by Archer in a series of articles in *Design* magazine. Archer's method consists of seven basic phases: briefing, programming, data collection, analysis, synthesis, development, and communication. The complete methodology consisted of over 200 activities which, although they were described in a sparse logical format, are often quite difficult [Archer, 1965]. At the same time, the organizers of a design conference in Birmingham, Alabama, made a concerted attempt to establish a common basis of agreement about design methods. Unfortunately, not only was no consensus reached, but no common view could be found among the thirty-five papers presented at the conference, in spite of the fact that almost all the papers focused on engineering design [Gregory, 1966]. The dissent among systematic designers endures today.

The systematizers of design may have met with more success if they had devoted more time to understanding the design process rather than formulating a particular design method. A successful method of design should be the logical result of the intense study of the actual design process -- the theory which describes the process of design.

2.2.2 Design Theory

Unfortunately, scant research has been done in the field of design theory. Most published work in this area is author speculation and not reliable research. Perhaps it isn't that surprising considering the fact that design is very much a human-oriented process, including viewpoints and expectations that vary with the individual, and not necessarily amenable to the scientific process. The scientific research that has been done confirms the ill-defined nature of design problems.

By observing town planners, one researcher decided that the design process is actually a learning process [Levin, 1966]. The designer learns about the problem through a trial-and-error process, and each error reveals an aspect of the problem that the designer had not previously considered. Design is actually more exploration than conventional learning since there is no real predefined body of knowledge to aid the designer with the problem he is currently facing. The situation is perhaps best described by two of the prominent design methodologists. Jones has stated that design is not necessarily problem solving, but problem finding [Jones, 1966]. His comment underscores the vague nature of design while Asimow's belief that "the designer is presented not with a problem, but with a problem situation, (and) it is out of this milieu of perplexity that clear definitions of the relevant problems must be drawn" [Asimow, 1962] exemplifies the difficulty in applying some systematic approach to the field of design.

Some researchers have deduced a tree-like structure underlying the design process. A major difference between "design trees" and "decision trees" is that the branches of design trees often form loops with other branches in the tree. This looping is the major stumbling block of design methodology which attempts to serialize the process of design. Under serialization, a designer can get caught in an "infinite loop" of design consideration. For example, an architect may be considering the placement of a bedroom, dining room, and a bathroom within a blueprint. He may decide to place the bedroom first because it is the decision with the greatest freedom of choice, and then do likewise with the dining room, and thus by default, determine the placement of the bathroom. In

this situation, it is possible that the default location is not suitable for placement of the bathroom, and the architect would be forced to reconsider his decisions about the bedroom and the dining room. The three rooms are connected in a decision loop that the serialized designer can get caught in since he doesn't recognize the interrelationships between the concepts. This conclusion is supported by a study of interdependent decision-making in the architectural design process. The researchers noted that the design team was "attempting to make their decisions sequentially, when in fact almost every decision was affected both by those that had gone (before) and those that were yet to come" [Levin, 1966].

Other researchers perceive a hierarchical structure underlying the design process. Essentially, when a designer made a decision at a particular level of abstraction, it leads to a well-defined set of options at the next-lower level [Marples, 1960]. Although design loops are present in this approach, they are resolved not by analyzing interconnected decisions simultaneously, but by making decisions within one of the single decision areas that was part of the loop [Gregory, 1964] which likens this approach to the earlier and generally unsuccessful design methodologies that focused on individual concepts and not the overall structure of the situation.

Perhaps the most comprehensive treatment of design theory is given by John Wade's person-object spectrum in which the transition from initial concept to final result can be described as a chain of relationships [Wade, 1985]. In the definition of a design problem, both the abstract and concrete contexts and the relationships connecting them must be considered to properly understand the problem. The relationships that form this "bridge" are purpose, behavior, and function. The problem description can be interpreted as a ends-means chain in which:

> for a person to exist, his purposes must provide for that existence; to accomplish his purposes, his behavior must serve those purposes; for his behavior to be carried out, certain function capabilities must often be supplied; for those function capabilities to be provided, some arranged physical object must exist to provide that function. [Wade, 1985]

For example, consider the relationships between a person and a chair. Typically, the function of the chair is to provide support, and the behavior of the person is to sit in the chair, to achieve the purpose of resting his legs. This exhibits a complete chain of relations: the object, the chair, serves

a function of support which permits the behavior of the person to sit down to achieve his purpose of rest. Another dimension to the spectrum hierarchically arranges the objects relative to an overall perspective. For example, the chair, can be a component of a room, which is a component of a house, which is a component of a subdivision, etc, each designed for a set of particular purposes, beliefs, and functions.

Design is an activity closely anchored to individual human perceptions and attitudes. There may be "good" design and "bad" design, but there is no "ideal" design that can be arrived at by ascribing to some methodology or theory. This belief is certainly supported by the complete lack of agreement on design methodology and theory, and at best, the bare fundamentals of design -- analysis, synthesis, and evaluation -- represent the only universally accepted design theory. Not only are the basic aspects of design ill-defined, but the underlying theory of design is ill-defined as well. It is already apparent that explicit formulation cannot work effectively in such a non-explicit domain, but it's possible that no problem definition method can capture the generic and ill-defined fundamentals of design. If a representation methodology is to succeed in this domain, something must be made more concrete. Once a representational basis is constructed, further enhancements can be made to the model as the domain becomes more generic.

2.2.3 Design Theory for Charts

Charts convey information in a concise and easily understood manner. Meaningful charts reflect a real-world situation or a conceptual outline based on the salient features of a given set of data. Charts can enliven a presentation, but more importantly, they can highlight important issues and ease the reader's task. For example, a line chart can quickly relate the overall trend of the data, and save the reader the added labor of reviewing a column of figures to determine whether or not the numbers are increasing or decreasing overall. Charting is essentially a method of data compression

that converts raw data into visual images and invites the audience to form their own conclusions about the data [Enrick, 1972].

Although charts may described as a method of data compression, they actually have a multitude of important aspects. Effective graphical displays should:

- show the data
- induce the viewer to think more about substance than methodology, graphic design, graphic production technology, or something else not directly relevant to the emphasis of the presentation
- avoid distorting what the data actually indicates
- present a large amount of data in a small space
- make large data sets coherent and understandable at a glance
- encourage the viewer to make visual comparisons of the information
- reveal the data at several levels of detail, from a broad overview to the fine details
- serve a distinct purpose such as description, exploration, or comparison
- be closely integrated with the statistical and verbal integrations of the data set

[Tufte, 1983]

Beyond serving these basic functions, charts provide a variety of benefits not found in other techniques of data compression. First, the immediate appeal of a good chart invites the attention of the reader and thus creates interest in the material being portrayed. Second, the visual comparison and

contrast of data permits relationships to be more clearly grasped and more easily remembered. Since they allow relationships to be quickly and easily discerned, charts are also useful in revealing previously unknown relationships in the data [Enrick, 1972]. Finally, charts can provide an often unrecognized benefit. Charts allow the viewer to reach a conclusion about the data without ever being told explicitly what the conclusion is. For example, if the audience is shown a line chart with an ascending curve, they automatically infer that the data is increasing. The audience is making conclusions about the data instead of the presenter attempting to convince the audience that his conclusion is valid. In essence, a viewer will believe his own conclusion with far greater readiness than he will believe someone elses.

Naturally, since charting possesses a number of facets, there are a number of places to make mistakes in charting design. A variety of charting methodologies have been proposed, but they can all be reduced to three fundamental steps:

1. Analyze the data.
2. Determine the message that the user wishes to convey.
3. Select the most effective type of chart.

[Spear, 1969] [Schmid, 1952]

The first two steps are the determining factors for the third step, chart selection. Data analysis consists of determining what type of charts can be rendered from the available data. For example, if the client wants to show a trend in sales for his company, he must have sales data for several years. Hence, an effective chart is a combination of both the client's objective and the data's ability to support that objective. Eliciting the client's objective can be considerably more difficult than analyzing raw data. Not only may the client not know which type of chart will best convey his message, but he may also be unaware of his motive on a functional level. If he did, there would be no need for graphical consultants since computer programs could present him with a choice between "trend," "comparison," and "display". The graphical consultant engages the client in a dialogue to learn more about his *situation*. As noted earlier, charts are much more than just "pretty

pictures". To create an effective graphical display, the graphic consultant needs to know details about presentation, the audience, as well as the interrelationships of the information to be displayed to the overall situation.

In the context of generic design theory, charting is certainly not as complex as architecture, but it does embody the same basic concerns endemic to the design domain: the need to construct a concrete item from vague, typically ill-defined specifications. Moreover, it requires a strong interaction with the client, since his perspective of the situation is of paramount importance. Chart design provides a reasonable compromise since it still operates on ill-defined problem descriptions, but has a much better defined methodology. This domain allows the automated consultant to focus on the key aspects of problem definition and analysis (the subject of this this thesis) and not on some methodology for design.

2.3 Eliciting Information from the Client

An entire spectrum of possibilities exist for human-computer interaction ranging from assembly language to natural language. In assembly language, the human makes the greatest concession to the machine in terms of "speaking its language", and the reverse is true in natural language where the human "talks" to the machine as if he was talking to another human. Not surprisingly, an assembly language interface is relatively easy to implement, and the natural language interface is prohibitively difficult to implement. The goal of an effective interface is to allow the human to communicate his ideas to the machine without entailing excessive development and implementation cost for the interface. Nowhere is this communication problem more evident than in artificial intelligence where the human communicates and the computer attempts to understand complex and abstract concepts. A variety of approaches have been made to resolve this communication gap

between man and machine, and they all focus on extracting the maximum semantic content without requiring the difficulties and complexities of a complete natural language system.

2.3.1 Natural Language

Natural language is by far the most desirable method of man-machine communication. Unfortunately, natural language is extremely difficult to implement effectively. On the surface, natural language processing does not seem to be a major difficulty. Even small children can grasp the basic concepts of linguistics and communicate effectively, and sentence parsing is even taught in the classroom as an algorithmic task. The source of the problem is that no one understands the complex cognitive processes involved in both linguistic expression and interpretation. Even deceptively simple sentences require considerable amounts of background knowledge, or common sense, to be properly interpreted. The meaning of the sentence extends far beyond its pure denotative content to encompass issues such as situational context, tone, and causality. Interpretation of these issues relies heavily on common sense, an achilles heel of artificial intelligence. Moreover, natural language is fraught with ambiguity: lexical and syntactic. Lexical ambiguity refers to the variety of meanings or "senses" a given word may have. For example, the word "soft" has over thirty definitions as an adjective, and it can also be used as a noun and an adverb. Syntactic ambiguity refers to the ambiguity of phrases and sentences. For example, no one would have difficulty interpreting "electric pencil sharpener" as a device powered by electricity that sharpens pencils; however, there are no set rules that dictate this. In fact, basic English grammar states that modifiers should immediately precede the object they modify. Humans can properly interpret the phrase because they know that there are no such things as electric pencils. Computers have no such knowledge unless it is explicitly coded. Beyond these basic issues are problems like referential ambiguity and metaphor. Referential ambiguity concerns the use of pronouns in normal discourse. The listener must

consistently be aware of who or what the pronoun represents. Consider the following sentences:

Each child put on a uniform.

They had been donated by a local manufacturing concern.

An individual would have little difficulty understanding that "they" referred to the uniforms and not the children because manufacturing concerns are not typically in the practice of donating children [Brittain, 1987]. Again, computers are not aware of this because they lack the wealth of world knowledge that the individual possesses. Metaphor is even more complicated because it requires *sophistication* in addition to basic world knowledge. Metaphor is by no means restricted to poetry or even stylized expression. In fact, it seems to seep into every form of discourse, including this sentence [Charniak, 1986]. The source of these difficulties resides at as more fundamental level. Computers and people see the world from radically different perspectives. While people perceive subtle nuances, absorb information from five unique senses, and integrate these perceptions into a holistic perspective which is cultivated over an entire lifetime, computers basically know only what they are explicitly told. This is borne out by the fact that any conventional natural language systems actually operate with severely restricted subsets of the English language. Both the words and sentence structure must correspond to some *explicitly pre-programmed* language set [Samad, 1986]. Hence, before any system can understand true natural language, it must first perceive the world, or situational context, as a real person does. Providing a machine with this perspective remains an impasse of artificial intelligence [Brittain, 1987].

2.3.2 Personal Construct Theory

Psychologists use a tool known as personal construct theory [Kelly, 1955] to interview patients. Personal construct theory builds a model of the patient's world based on similarities and differences of selected concepts. This approach views a person as a "personal scientist" who classifies, categorizes, and theorizes about his world. In essence, an individual uses a system of organization

consisting of components and interrelationships between the components. The interaction of concepts within the structure produces interdependencies. If the client can become aware of the structure and the organization within the structure, he increases his ability to make adequate predictions and act according to them. He increases his own understanding of the situation, thus becoming a "personal scientist."

The basic tool of the personal scientist is the repertory grid, which is a method of extracting the conceptual system held by the individual. A repertory grid consists of constructs and elements. A construct is a bipolar dimension, which to some degree is a characteristic of each element. A construct is not the same as a concept in that a construct is a measure of a particular characteristic.

> A construct is a way in which some things are seen as being alike and yet different from others. ...The idea of a relevant contrast and limited range of applicability or convenience is not involved in the notion of a concept, but is essential in the definition of a construct... Sometimes concepts are also regarded as as ways in which certain things are naturally alike and really different from all other things. This use suggests that a concept is being considered as a feature of the nature of things, an inherent categorization of reality. The idea of a construct does not carry with it any such assumption, but rather is seen as an interpretation imposed upon events, not carried in the events themselves. The reality of a construct is in its use by a person as a device for making sense of the world and so anticipating it more fully. It must be stressed that all invented dichotomies, however widely agreed (large-small), specifically annotated (bass-treble), or scientifically approved (acid-alkali) are constructs -- useful inventions, not facts of nature [Bannister and Mair, 1968].

Elements in the grid are elements of the situation which are personally important to the individual. In scope, elements can be anything that represent something: events, experiences, objects, or people. For example, elements in a baby's view of the world could be "mother", "father", and "teddy bear." The mapping of the elements onto the constructs produces a two-dimensional grid which indicates relationships between the concepts, based on the client's constructs.

At the outset of the psychological consultation, the individual lists the elements relevant to the model. To elicit constructs, the individual is presented with a set of three elements taken from the total set of elements and is asked to choose two that possess a quality that the third does not. The common quality is called the emergent pole. The individual provides a name for that quality and the interview continues until all the concepts in the overall set are adequately defined. The implicit pole is the characteristic that differentiates the third element from the other two and may also be elicited directly, though it is not essential to the consultation.

Using an example from [Shaw, 1980] to illustrate, consider three school subjects: Mathematics, Literature, and Art. Now group these into the two which are similar and the one that is different.

Janet says: "Mathematics and Literature are alike because they are about a body of knowledge, and Art is about self-expression".

Philip says: "Literature and Art are alike because they are about life, and Mathematics is abstract".

John says: "Mathematics and Art are alike because they are communication by symbols and forms, whereas Literature is communication by words".

Mary says: "Mathematics and Literature are alike because they are useful in life, but Art is a waste of time".

Lynn says: "Mathematics and Art are fun and easy, but Literature is about writing essays which I don't like".

Obviously, each person has a different opinion based on a different value system. Each of the dimensions are personal constructs since it is expressed in personally meaningful terms such as abstract/concrete and like/dislike [Shaw, 1980].

The principal strength of personal construct theory is that it elicits the constructs without actually asking the patient what they are. This is valuable since most patients would be unable to even respond to a question like, "What fundamental measures do you use to distinguish between different areas of scholastic work?" Moreover, patients have an inclination to give the reply they feel is expected of them rather than the one that reflects what they truly believe (if they even know). Personal construct theory solves both of these problems as well as produces a substantial amount of information from a nominal amount of discourse. For example, the reader can easily construct a rough psychological sketch of the five respondents mentioned above on the basis of only one question.

The informative value of this type of discourse is not lost on computer scientists. This methodology allows a program to build a tailored model of the user's world. The model is tailored because it employs the user's terminology and personal beliefs to build the model. More importantly, the model can be constructed without the aid of natural language. Instead of having the client respond with an entire sentence, the system can have the user differentiate the concepts on the basis of a one or two-word characteristics. Personal construct theory has proven useful in eliciting knowledge

from experts to build expert systems since it constructs a model of the experts view of a situation. From the extracted repertory grids, knowledge engineers can determine the relationship of concepts to each other on the basis of characteristic commonalities [Boose, 1985].

2.3.3 Content Analysis

Prospective problem modelling tools, however, are not unique to psychology. Content analysis, a tool used by communication scientists, seeks to understand data not as a collection of events, but as symbolic phenomena. In essence, content analysis is a research technique for making replicable and valid inferences from data to their context [Krippendorff, 1980]. Content analysis consists of six basic components:

- the data as communicated to the analyst
- the context of the data
- how the analyst's knowledge partitions his reality
- the target of a content analysis
- inference as the basic intellectual task
- validity as the ultimate criteria of success

While this may sound like a logical solution to the interviewing problem, it remains prohibitively difficult to implement for several reasons. First, to be effective, content analysis requires an accurate description of a context to map inferences to. Unfortunately, the best way to represent contexts involves the use of pre-programmed scripts which define the contextual framework of a situation.

Scripts exhibit the same lack of flexibility inherent in conventional expert systems since they consist primarily of a pre-programmed component. Moreover, by the author's own admission, computerized content analysis will not be successful unless vocabulary, syntax, and semantics are restricted to a particular area of discourse, and the contexts are *explicitly* statable. Content analysis shares a striking similarity to natural language in these respects, and it also shares the basic fundamental problem: emphasis is placed on the *computer's* perspective which is deficient when compared to the human perspective which it attempts to mimic.

2.3.4 System Dynamics Modelling

Industrial engineering offers a useful design alternative: system dynamics modelling. Businesses, economies, and social organizations all exhibit dynamic behavior, and this dynamic behavior is typically the result of causal interactions among the components of the system. For example, a company holds an inventory of finished goods, which is depleted by sales, and replenished by production. Clearly, both sales and production have a definite influence on the inventory. However, sales and production are also influenced by other areas. Production depends on a labor force, available equipment, projected orders, etc., and sales depends on factors like the market condition, advertising, and the sales force. These factors are also influenced by other considerations including some that have already been mentioned. The primary aim of system dynamics is to find policies that will control the system in response to shocks from the outside world. Essentially, system dynamics is

> A method of analysing problems in which time is an important factor, and which involve the study of how a system can be defended against, or made to benefit from from, the shocks that will fall upon it from the outside world. Alternatively, System Dynamics is that branch of control theory which deals with socio-economic systems, and that branch of management science that deals with problems of controllability [Coyle, 1977].

To achieve this end, system dynamics constructs a scientific model of an objective reality through directed graphs that link relevant concepts in the system by arrows. The arrows in the graph indi-

cate a causal influence from one concept to another, and a link is usually accompanied by a plus (+) or minus (-) sign indicating direct and inverse influences respectively [Coyle, 1977]. A system dynamics model is usually implemented on a computer and the simulation shows what happens to the overall model when certain factors are changed. As the change propagates through the model, the client gets a better idea of the impact that certain areas have on the entire model which can help define strategies for controlling the model under certain situations such as responding to fluctuations in uncontrollable variables.

In system dynamics, the client specifies both the concepts of the model and their interrelationships which allows an individual to create not only a description of his situational context, but also a way to show how the context can change over a period of time. System dynamics provides a strong generic basis for representation, and this is evidenced by its range of actual applications: from modelling situational political influences to modelling a magazine publisher's business concerns (Figure 5 on page 33) [Eden, Jones, Sims, 1983]. Since system dynamics is a modelling tool, it is also a representational tool. The causal model used for simulation is a model of a particular situation, and the causal model can be effectively interpreted as a cognitive map of a situation. This forms an intriguing basis for problem definition, and the "concepts and influences" basis of system dynamics differs considerably from the "facts and rules" basis of propositional formulations.

2.4 Survey of Contemporary Expert Systems

Expert systems are currently the most marketable product of artificial intelligence research. The goal of AI scientists has always been to develop programs that could in some sense, think like people do, or at least solve problems in a way that would be considered intelligent if done by a human. While the initial hope was to develop a system that could solve general problems, it has proven far too difficult and has never produced any substantial results. In general, the more prob-

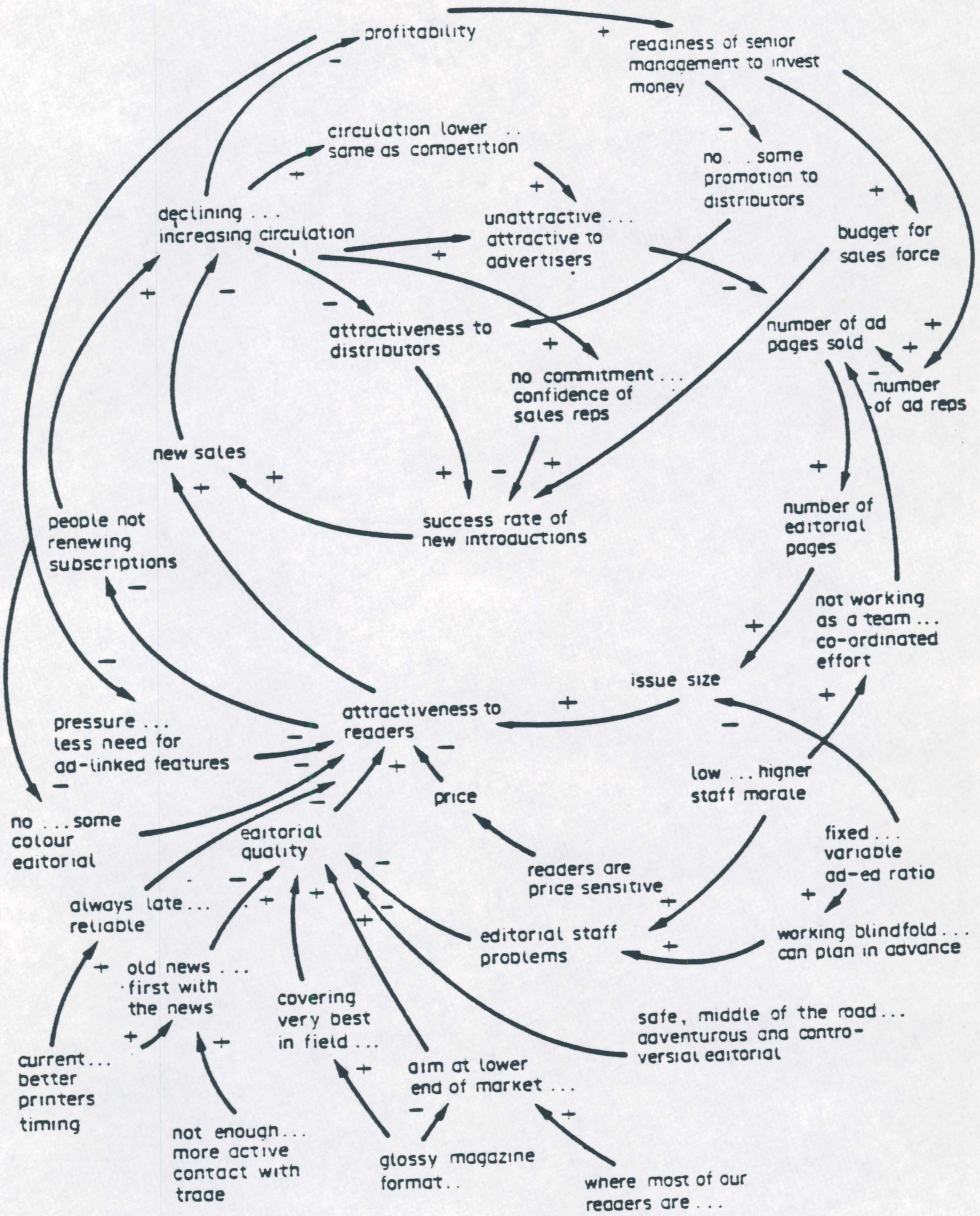

Figure 5. System Dynamics Model of Magazine Publishing: [Eden, Jones, and Sims, 1983]

lem classes a system was designed to handle, the more poorly it handled them. From this maxim arose the expert system. As the name implies, expert systems are highly specialized systems [Waterman, 1985]. While expert systems operate in areas like diesel engine failure diagnosis or molybdenum prospecting, they can be categorized into broad areas which characterize the systems fundamental activity such as diagnosis, interpretation or design [Chandrasekaran, 1986].

2.4.1 Diagnostic Systems

The diagnostic system is by far the most common type of expert system. The reason for this popularity stems from the basic nature of diagnostic problems. Diagnosis problems are easily expressed as tree structures. In general, each branching point in the tree structure corresponds to a particular question, and the leaves of the tree correspond to the final diagnosis. The diagnosis is effectively a combination of observable symptoms elicited from the user [Merritt, 1987]. A diagnostic expert system already has knowledge about all possible results of a consultation and arrives at a solution by following a particular path in the tree guided by user responses to pre-programmed questions. The basic diagnosis paradigm of query and search is well-suited to a computerized implementation.

Not surprisingly, expert systems in this domain have met with a significant measure of success. CADUCEUS, a system developed at Carnegie Mellon, relates symptoms to diseases in internal medicine. Although, CADUCEUS doesn't use a pure decision tree for knowledge representation, it does use a large semantic network. The inference principle is the same, except that a semantic network is traversed (instead of a tree) to reach a conclusion. CADUCEUS is one of the larger expert systems, containing profiles of over 500 diseases and 3500 manifestations of diseases. MYCIN is another diagnosis system that determines the nature and treatment of infectious blood diseases. MYCIN employs about 400 rules in its knowledge base and tries to draw inferences from the rules and available knowledge. Again, a tree structure is not explicitly used, but the flavor of

MYCIN and other diagnostic systems involves deduction based on information from the patient. Diagnosis is by no means restricted to the medical domain. MES is an expert system used by the air force to diagnosis problems with aircraft engines. MES's knowledge consists of information taken from repair manuals such as component weight and dimensions, troubleshooting procedures, and repair methods coupled with experiential knowledge or "rules of thumb" taken from expert mechanics [Waterman, 1985].

There are a number of other diagnostic expert systems in diverse areas ranging from network fault diagnosis to fault identification in large chemical plants. Although the diagnosis paradigm is easily implemented on a computer and a wide variety of systems have been developed, the overwhelming majority of systems have never gone beyond the stage of a research prototype.

2.4.2 Interpretation Systems

Interpretation is essentially a task that maps one context into another. The most obvious example is that of language interpretation which maps between linguistic contexts. Unfortunately, linguistic contexts are rife with issues from the human domain, which cause so many problems with natural language processing; and, because of this, the typical interpretation system operates in a far more restricted domain. Moreover, interpretation systems are somewhat more difficult to implement than diagnostic systems. Interpretation is not as concrete as diagnosis because interpretation requires a more humanistic perspective. Interpretation is an investigation of a particular fact or event in accordance with some context, and the investigation is usually concerned with the ramifications of the event under particular circumstances. For example, the interpretation of killing someone can be quite different when viewed in different contexts. If someone is killed because they are threatening someone else's life, it is perceived differently than if they are killed in a car accident. The event is the same, but it is perceived in a particular context which actually determines the meaning of the event.

One such interpretation system, SPE, distinguishes between various inflammatory conditions of a patient by interpreting waveforms from a scanning densitometer [Waterman, 1985]. The expert system interprets the waveforms in relation to a context derived from patient data to related disease categories. SPE has actually achieved commercial success, but it should be noted that it operates in a highly-restricted domain.

Another interpretation system is META-DENDRAL. META-DENDRAL helps chemists determine the dependence of mass spectrometric fragmentation on substructural features. It interprets the spectrometric information on the basis of known chemical substructures to predict fragmentation. META-DENDRAL accomplishes this by first generating a set of highly specific rules from the known substructures and then generalizes these on training examples [Waterman, 1985]. The generalized rules represent the interpretation relative to the particular context. Although the program has achieved some success, it remains a research prototype.

At this point, it is apparent that there is no clear delineation between the areas of diagnosis and interpretation. The areas are not necessarily independent, and may actually share some basic characteristics. As one progresses from diagnosis domain to the interpretation domain, however, the problem definition and resolution becomes less concrete and more subjective, and hence, more difficult to program. The progression from interpretation to design represents a greater movement along this continuum as well as entailing far greater programming difficulty.

2.4.3 Design and Configuration Systems

The design/configuration domain is not as easily addressed as the diagnostic domain. Part of the difficulty is caused by the nature of design problems: design problems are less structured and thus more difficult to adequately define in advance. Most systems in this domain are more configuration-oriented since configuration problems are better defined in advance and the final

solutions (configurations) are limited as well. Configuration is somewhat different from design since both the number and type of items as well as the number of possible arrangements of these items is severely restricted. Configuration systems are included in this section because configuration is generally considered a design activity and helps explain why true design expert systems *do not exist.*

2.4.3.1 R1

Perhaps the best-known configuration system is R1. R1 is an expert system developed and used by Digital Equipment Corporation to determine system configurations for the VAX 11/780 computer system. A successful system configurer (human or machine) must have two basic types of knowledge: knowledge about each of the individual system components including knowledge about voltages, ports, frequencies, etc. and knowledge about partial configurations that include these components within the context of a complete system. Typically, a configurer needs to know approximately eight pieces of information about a component and there are 420 components supported by the VAX system. Hence, a configurer must have knowledge of over 3300 pieces of information. Beyond this, the configurer must have rules for the relationships between the components. The initial expert system had 480 rules to represent these interrelationships, and to further restrict the domain and thus simplify the problem, additional configuring constraints were added to the system. Market considerations which limit the number of possible configurations and engineering considerations which eliminate sub-optimal designs are two examples of ways the configuration set is limited [Waterman, 1985]. Although the system is currently used by DEC to service customer requests, it is not a true design system. A true design system is less restricted, and the set of design components are varied and less-defined in the context of a larger system.

2.4.3.2 VT

Another design system, VT, designs elevators by interpreting a customer's functional description to produce an equipment and parts configuration that satisfies the customer's specifications as well as safety, installation and maintenance requirements. Although VT and R1 use domain-specific strategies to produce solutions, the basic paradigm that VT employs differs from that of R1 in that VT's problem solving strategy consists primarily of constructing an approximate solution and successively refining it [Marcus, 1987]. Again, VT is not really a true design expert system in that the domain of possible results and method of consultation is so severely restricted that VT is configuring, and not really designing. What is notable about VT is that it goes through design iterations. Iteration is a standard part of design methodology, and a system that can refine its design based on user input is closer to the "true consultant" paradigm.

2.4.3.3 Dominic

A more flexible design expert system is Dominic, a system which employs a domain-independent structure for solving mechanical engineering design problems. When the system is given a problem, it posits an initial design and iteratively improves on it by using knowledge about the relationship between design goals and design variables. The input to Dominic consists of a set of parameters indicating physical constraints on the design, performance goals, and an initial design. Dominic then evaluates the initial design to find its weaknesses and then proposes a change in one of the design variables. If the overall effect of the change is positive, the change is implemented. This process is repeated until the overall design is deemed acceptable. In general, Dominic utilizes a hill-climbing algorithm to solve design problems which are described as a combination of problem parameters, design variables, and goals. While dominic applies a general strategy to its domain, it still requires the problem to be stated in explicit, predefined terminology. Moreover, Dominic's problem-solving methodology requires the client to explicitly state an initial design as well as ex-

plicit goals. While this methodology may be acceptable in mechanical design, it is of little utility in domains such as architecture where the client may not know how to formally state his "goals" or provide an initial design to the consultant [Howe, 1986]. Although Dominic suffers from serious limitations, it also possesses some interesting improvements over the basic configuration systems such as R1. Again, Dominic performs iterative design, but it also uses an experimental design paradigm. A design is rendered by determining how certain changes in the design affect the overall model. This can be considered another technique available to the human designer, albeit not the only one.

2.5 *Conclusions*

The literature review illustrates some of the fundamental problems encountered in this research. The conventional problem definition approaches, though well-suited for some domains, are not effective in the design domain, primarily becuase of the generic, non-explicit nature of design problems. The problems in automating a design consultant are exemplified by the basic lack of design expert systems. The expert systems that have been implemented, still use conventional representational technology and therefore operate in severely restricted domains. Moreover, these systems are directly concerned with the problem in that the user still specifies issues about the problem (or even posits a basic design) instead of information about the situation. Finally, it is difficult for an automated consultant to extract and understand such a situational description without restricting the client's method of explanation. A viable solution must not only allow the client to adequately specify his situation, but create a problem model that the automated consultant can understand.

3.0 Problem Analysis

Expert systems purport to be the computer equivalent of a human expert. Through an interrogation process, the system determines the nature of the client's problem, analyzes it, and proposes what it believes to be the solution. In a broad sense, this is the same approach that a human consultant uses; however, the human consultant's actual *methods* of interrogation and analysis differ considerably from the contemporary expert system approach. A "true consultant" applies domain knowledge to solve a problem not previously seen. For example, a graphic design consultant must accept the statement of almost any problem and turn it into a visual design. By contrast, in typical expert systems applications, all the problems that the system can handle are defined during the knowledge-engineering phase, and the system produces advice by following a decision tree guided by the client's answers to a set of pre-defined questions. Without an adequate problem definition phase that allows the client a reasonable latitude of expression, current expert systems techniques cannot allow construction of a programming consultant, nor any other consultant that relies heavily on the client's conception and description of the problem.

3.1 The True Consultant

What exactly constitutes a "true consultant" and how does he differ from the usual expert system consultant? A true consultant embodies qualities that are uniquely human, and at present, there are substantial differences in human-human interaction and human-computer interaction. Some of the primary differences can be characterized by personifying the expert system consultant. In many ways, the conventional expert system can be likened to a lawyer who is cross-examining a witness in a trial. The classic (in drama, at least) lawyer's ploy is to allow only yes and no answers to often complicated questions. The answer the witness gives may serve the lawyer's purpose of making a point, but it does not really answer the given question because there are usually extenuating circumstances that should be explored. The lawyer behaves in this manner to build the type of situation that best suits *his* purposes. While expert systems are not attempting to prove a point in a trial, they do require restricted replies to what are often complex questions, and while this behavior may be acceptable in constructing a legal defense, it is completely unacceptable behavior for a successful consultant. Any human consultant who behaved like a contemptuous lawyer would have a difficult time finding employment; and yet, all conventional expert systems have, to a significant degree, an interrogation style analogous to the "contemptuous lawyer". In their problem definition phases, expert systems are seeking answers to the questions that best suit their own purposes. While true consultants will, at times, behave this way, they have an entire repertoire of techniques for problem definition, while the expert system uses this as its only method of eliciting information from the client. In fairness, this type of behavior is useful and even necessary in some domains. For example, an emergency management consultant needs precise and timely answers to particular questions to provide advice in an emergency situation. As these examples illustrate, different information acquisition methods are required in different domains, and the basic paradigm for acquisition in the design domain is quite different from that in the diagnosis domain because of the basic differences in the information and objectives in the domains. This helps explain the limited success of diagnostic expert systems. In the diagnosis domain, the consultant often needs

answers to particular questions about the system in question. Consider a computer repair technician on a service call. Typically, before he even looks at the faulty machine, he will ask questions about the symptoms of the machine in an effort to *classify* the nature of the failure. Diagnosis, however, is only one of a variety of possible domains. The incisive question and answer approach will not work nearly as well in the interpretation domain (explained in the literature review), and not at all in the design domain. Both interpretation and design depend heavily on the *client's* perspective, and there is currently no way to effectively incorporate his perspective into an automated consultation. This, in turn, helps to explain the general failure of design and interpretation expert systems.

If a computer could understand natural language, it would likely perform as a viable consultant since it would have an effective supporting representational structure for knowledge (if it can talk like a human, it must think like a human). Unfortunately, natural language remains a distant and possibly unrealizable goal. Natural language may not be necessary, because the way a consultant *directs* a consultation is actually more important than unlimited free expression of ideas. The solution lies somewhere between the two extremes of the contemptuous lawyer and the "rambling conversationalist". This is not meant to under-emphasize the role of a client in the consultation. The client must be allowed to express his beliefs and ideas freely, but one of the consultant's jobs is to provide some direction to the consultation based on his background knowledge of the domain which provides an intuition about what is "important". The primary difference between this approach and the contemptuous lawyer is that, in the former, the consultant fashions his mental model in terms of the client's description rather than forcing the client's situation into some predefined classification scheme. Experience indicates that an effective consultant should allow the client to talk as directly as he can about his concerns. Typically, things that are seen as objective, hard facts about a problem are often fairly trivial when compared with subjective feelings and beliefs that the client feels are central to the problem under consideration [Eden, Jones, Sims, 1983].

3.1.1 The Nature of the True Consultant's Problems

To clearly understand how a true consultant performs, one must first understand what problems are. Problems are what consultants are supposed to solve, but what constitutes a problem? Empirically, a problem is a situation or question raised for inquiry, consideration, or solution. A more useful definition can be found in *Messing About in Problems*: "problems are psychological entities which are often unclear and expressed as anxiety and concern about a situation as well as being expressed as a positive wish for the situation to be different in a particular way". The authors continue by suggesting "a process of assisting the definition and formulation of a problem is crucial and often neglected precursor to any attempts to solve it," and that "understanding a problem as someone else sees it needs consideration and original methods for recording what you hear when you listen -- to both verbal and non-verbal elements of a problem description" [Eden, Jones, Sims, 1983]. Therein lies one of the abilities of a true consultant: the ability not only to effectively define the problem as the client sees it, but also to define it in a manner that it is useful to the consultant. For example, many clients would be happy to define their problem through seemingly endless ramblings which, although yielding some valuable information, also yield tremendous amounts of useless information which may serve to obscure the real problem as well as waste the consultant's time. The consultant offers "help" to keep the problem definition focused on the relevant aspects of the problem, and the style of help can vary from coercive to empathetic. A consultant may be coercive in that effectively he tells the client what his problem is, perhaps using a simple problem interpretation to force a more complex (and complete) problem interpretation from the client. Unfortunately, a client may allow the consultant to completely define a problem which the client may not actually have. The empathetic approach is the diametric opposite of the coercive approach. The empathetic consultant attempts to fully understand the client's problem in the client's terminology while staying within the client's methods of understanding concepts and taking actions. One problem with this approach is that no one can completely understand another's problem without actually becoming that person, and then they would not only understand the problem, but

they would also have it. A third strategy, the negotiative approach, is a hybrid of the coercive and empathetic methods. As the name implies, the client and the consultant negotiate a problem which may not be the exact problem that the client perceives nor one that the consultant considers helpful to analyze, but one that falls somewhere between the two -- a compromise between two perceptions that embody different attitudes towards the situation under consideration.

One might wonder why the consultant does not simply ask, "What's the problem?" In general, the client's answer to that question does not really contain information that adequately or even accurately describe the actual situation. Moreover, the step between feeling some discomfort or dissatisfaction, feeling that a problem exists somewhere, and being able to say "The problem is such-and such" is a very big step [Eden, Jones, Sims, 1983]. Consider taking an automobile to a mechanic. The mechanic's first question typically *is* "What's the problem?" In response, the customer will give *symptoms* of the problem and not state the actual problem. Instead of saying, "The distributor cap is cracked", he will say "The car stalls frequently". The mechanic may pursue the issue by asking if the car stalls all the time, or in particular instances, such as rainy days or after car washes. The mechanic attempts to construct a situation relevant to the suspected problem, but in terminology that the car owner understands. The mechanic can then deduce the nature of the malfunction by applying his background knowledge in auto mechanics to the situation model that was defined by the customer under the mechanic's guidance. If the diagnosis domain often requires a model of the situation as a problem definition, more complex domains like design will almost certainly require a model of the situation to define the client's problem effectively.

In a broad sense, a primary function of a consultant is the elicitation of a goal from the client. While in some situations this may be as easy as asking the client "What are you trying to achieve?", in many other situations the goal is buried in a morass of situational complexity that requires an expert to discern what the actual goal is. For example, the programming consultant mentioned in the introduction could be faced with two different types of problems: heuristic classification and observed vs. intended behavior. Heuristic classification problems are problems that can be classified based on immediately observable symptoms. Errors such as incorrect syntax, infinite loops, and

uninitialized variables are possibilities for heuristic classification. The presence of an infinite loop, for example, can be detected almost instantly. The observed versus intended behavior type of diagnostic problem is far more difficult to identify and resolve. The intent of this type of diagnosis is to determine if the program is really doing what the specifications indicate. A programming consultant compares the actual program against a mental model (based on the program specification) to determine the validity of the student's program. While this approach could be applied to something as simple as an infinite loop, it would be overkill and certainly not mirror the approach a human consultant would take.

The design expert approaches a consultation somewhat differently than the diagnostic expert. The client may have a particular objective in mind, but the objective must be interpreted within its situational context. The consultant *derives* the goal(s) from the client's description of the situation and professed objective. To do this, the consultant must first build, from scratch, an effective model of the client's problem. From this model, he then extracts the relevant and vital ideas to determine what really must be done to resolve the problem. Part of the consultant's expertise lies in his ability to identify the important issues in the client's problem as well as knowing what types of problems are in the domain of expertise. If the client already knows the course of action, he really does not need a consultant. For example, when a client consults a human factors and graphic design specialist to design a computer screen display, the specialist does not ask the client "What type of screen layout do you want?" because such a question eliminates the usefulness of the specialist. The consultant typically asks questions like, "What information do you need to see most often?" or "What information do you look for first?" The consultant uses these questions to construct a mental model of the client's situation. From this mental model, the consultant applies his background knowledge in graphic arts and human factors to the basic structure of the problem to create a screen design. In essence, the client cannot explicitly describe the type of screen he wants, but he can describe the situation that dictates the design, and the consultant uses this situational description to determine the client's "goal". This underscores the belief that human problems are usually ill-structured and not easily classified. Furthermore, the more freedom of expression that

the client is allowed, the more difficult it becomes to classify his problem and discern his objectives. This is one of the major problems associated with implementing a true consultant as an expert system using conventional expert system technology.

As the above comparison of design and diagnosis domains suggests, the underlying theory of diagnosis (heuristic classification) is different from that for design. For the true consultant concept to work, the consultation theory for the domain of expertise must be used by the automated consultant. The theory should motivate the representation structures as well as the information elicitation methods. If the automated consultant is to behave like a human consultant, it must "think" like the human consultant which means it has some foundation in the theory of the domain in which it operates.

3.2 Technical Problems with a True Consultant

Since conventional expert system technology does not provide a viable foundation for a true consultant, and since the aim of any expert system is to emulate what an expert actually does, it is worthwhile to investigate how an expert approaches a consultation. The impetus for this investigation is not to find the optimal user interface, but, more importantly, to find a mechanism capable of capturing and utilizing knowledge in the same manner as the human consultant.

3.2.1 The Representation Pipeline

One of the most critical issues in expert system design centers on the structure used to store knowledge about the consultation and the domain. The structure should reflect the basic nature

of the information it represents. While the structure may be more concrete and more readily defined in the diagnosis domain, it becomes increasingly complicated as the problem domain becomes less concrete. For example, the design process starts with some vague concepts and ends with a concrete object. This transition in the basic nature of the information can be illustrated by several examples. Consider first, an architect consulting with another architect to design a house. In this case, the architect knows exactly what he wants. He describes the house to the other architect in the utmost detail. The client is actually describing the concrete object. One may wonder what the purpose of such a consultation would be, given that the architect already has a complete idea of what he wants. Although the architect may wish to verify his own ideas with another professional, the purpose of this example is not to consider the plausibility of such a consultation, but to explore the basic nature of the information communicated from the client to the consultant. For the second example, let's assume the client is not an architect, but someone who has considerable practical experience with architecture. This client's knowledge is not so extensive he can give a detailed description of the house, but he can give the consultant some concrete information about the structure. For example, he may tell the consultant he wants the southern face of the house to be mostly glass, and the northern face to be partially underground. While these are somewhat broad specifications, they do represent concrete descriptions. The client knows what his basic objectives are, but does not know the exact way to achieve them. In the third example, the client knows virtually nothing about architecture. He has some ideas about what he would like his house to be like, but has no idea how these ideas translate into design objectives. Drawing from the previous example, this client may know he would like an energy efficient home that still provides a considerable amount of natural lighting, but be totally unaware that *his* objectives translate into the *design* objectives of the previous example. The last example is representative of a typical design consultation in that the client has little or no knowledge of the design domain, and the consultant must transform a situational description into an object that satisfies the client's objectives by satisfying the design objectives identified by the consultant. The basic differences in the data make it difficult to find a single appropriate representation for the client's objectives, the design objectives, and the designed object.

As noted above, a consultation can involve different types of information, and a point of contention in AI is how such knowledge should be combined into an internal representation. In conventional expert systems, knowledge about problems worth solving (including problem definition) and background knowledge of the domain itself have been combined into the primary representation technology of expert systems, rules. [Aikins, 1983] suggests that rules combine knowledge and control in a detrimental fashion in that the knowledge about the domain as well as the sequence of applying that knowledge are both contained in the same structure, rules. This observation offers an insight into another problem with automating the true consultant: separation and combination of knowledge. While rule-based approaches combine all knowledge into rules, a more complex representation paradigm may foster more robust systems with a far greater range of application. First, one must determine what knowledge should be separated. A simple approach would be to separate the problem definition knowledge from the background, or domain, knowledge. Even this simple approach introduces an important complication. At some point in the consultation, the information from the interview must be combined with the background knowledge to arrive at a solution. Not only is the basic nature of the information different, but it will probably be represented with different structures, and the information will have to be translated from one structure to another.

To further illustrate, a situation description tool such as system dynamics modelling (see literature review) may be an excellent choice for representing the problem situation; it is not well-suited for representing a concrete object since the basic structure of an object consists of parts and attributes for that part and not concepts and their respective influences. Conversely, a tree structure may be useful to represent the object, but possess insufficient flexibility to provide a generic model of a problem situation. It may be possible to use one structure for both representations, but it will be inefficient and probably not reflect the way the true expert considers and solves a problem. For example, when an architect begins a consultation, he is more concerned with the client's thoughts and preferences. As the consultation progresses and the architect gets a better idea about the desired building, he shifts his mental model from one of interdependent and often conflicting client prefer-

ences to a concrete model of what the building will look like. While the second model is certainly derived from the first, it is indeed different in that the architect is now thinking in terms of a particular structure with particular attributes that meets the client's demands.

Granted, it is *possible* to use personal construct theory to represent an object or a tree structure to represent a problem, but such a move is tantamount to using a screwdriver to pound nails into a board. That is, the tool may actually "work" but it is not really suited to the task, and at best will produce marginal results. Instead of compromising reasoning efficiency in an expert system by forcing information into inappropriate representations, it may be worthwhile to *transform* a given representation into one that is more appropriate for that stage of the consultation. The transformation must preserve the basic nature of the knowledge, yet convert it to a form whereby it best suits the information that it represents.

3.2.2 Situation Definition

The basic tool of interviewing is, of course, questioning. Clearly, one must ask questions to gain information, and this is what both experts and conventional expert systems do. The fundamental difference lies in the type of questions asked. As stated before, the human consultant allows more freedom than do the standard "yes/no" or "select one from the following" questions. The human expert seeks richer content by requiring more detailed answers in an effort to build a mental model of the problem. While a complex situation model may not be necessary for a heuristic classification system, the model is vital to a system in the design domain because the expert needs a solid understanding of the client's viewpoint in order to design what the client wants. If the expert is to accomplish this, he must permit the client to describe the situation in his own terminology. To be successful in the design domain, the consultant must be able to "step into the client's shoes" and to see the problem from the client's perspective. Because conventional expert systems have avoided the design domain and have focused on domains that are well-defined prior to any consultation,

they require less information and construct a less complex model, typically a decision tree. The automated design consultant needs to build a basic influence model as a cognitive map of the user's beliefs. The model should contain the basic aspects of the situation and the interrelationships between these concepts. The system and the user should build a useful and informative model of the problem by incorporating both the user's terminology and perspective, just as a human expert does.

3.2.3 Goal Elicitation

Conventional expert systems reach their conclusions through pattern matching and data driven rules. They elicit information from the user until a clear choice exists. While this paradigm may be suitable for the some problems, it is not amenable to basic design methodology. For effective design, a consultant must view the structure of the relationships *simultaneously* to avoid getting caught in constraint satisfaction loops. This poses a difficult problem if the client is free to devise whatever problem model he wants. Using a cognitive map consisting of arrows and concepts (like System Dynamics Modelling), a map with five concepts has over 100 possible configurations which requires the system to have knowledge of over 100 different influence models and their meaning for this scenario alone. A more intelligent and perhaps more human-like way to resolve this "representational explosion" is to focus only on those concepts that the client perceives as vital to the situation. The logic behind this approach derives from top-down design methodology that starts with the highest level of interpretation and resolves the relational issues at this level before proceeding down to a more detailed level.

3.2.4 Application of Domain Knowledge

The final stage of a design consultation represents a more detailed level of analysis. In this stage the consultant applies his background knowledge about the domain to produce the problem solution. The problem solution is a fusion of the consultant's domain knowledge and the aspects of the problem that dictate the nature of the solution. The consultant applies his domain knowledge to his own model of the situation which is composed of design objectives and constraints derived from the client's situational description. Only at this advanced stage of the consultation does the expert apply a classification system to the problem. To the consultant, the design objectives and constraints suggest a particular course of action that indicates the actual design -- the result of the consultation.

3.3 Conclusions

Without defining a new knowledge acquisition and representation method, expert systems will not extend beyond their current, limited range of applications. While the methodology must significantly increase the client's role in the consultation, it must also retain some structural, and thus semantic, consistency so the expert system can properly interpret the client's description. More specifically, indirect goal elicitation is vital in the design domain since the client's objectives are usually different from the design objectives which map directly into the designed object. The more flexible the problem model is, the "farther" away the problem model is from the model of the rendered object. A situation model, and all the benefit it provides, is not possible without a representational pipeline and representational transformation. The transformational component permits a given representation to be tailored to the information it is meant to represent without loss of reasoning efficiency and allows the application of background knowledge to a generic situation

model. Background knowledge, however, is more than just rules about the domain. The *method* used by the automated consultant should mirror that of the human consultant. As shown in Figure 6 on page 53, CHARTMAKER's architecture is directly related to design methodology. Each of the basic aspects in a design consultation have a corresponding component in CHARTMAKER. The first phase of design methodology, definition, is often considered implicit, so it is enclosed in a dashed box. CHARTMAKER's background knowledge of the design domain consists of rules, a specific sequence of representations, the individual representations, and the transformations that map from one representation to another. The tailoring of models to the information they represent, the separation of different types of knowledge, and the transformation of problem models according to certain contexts has the intuitive feel of what a human consultant actually does, and a successful automated consultant could benefit from a similar approach to problem definition and solution.

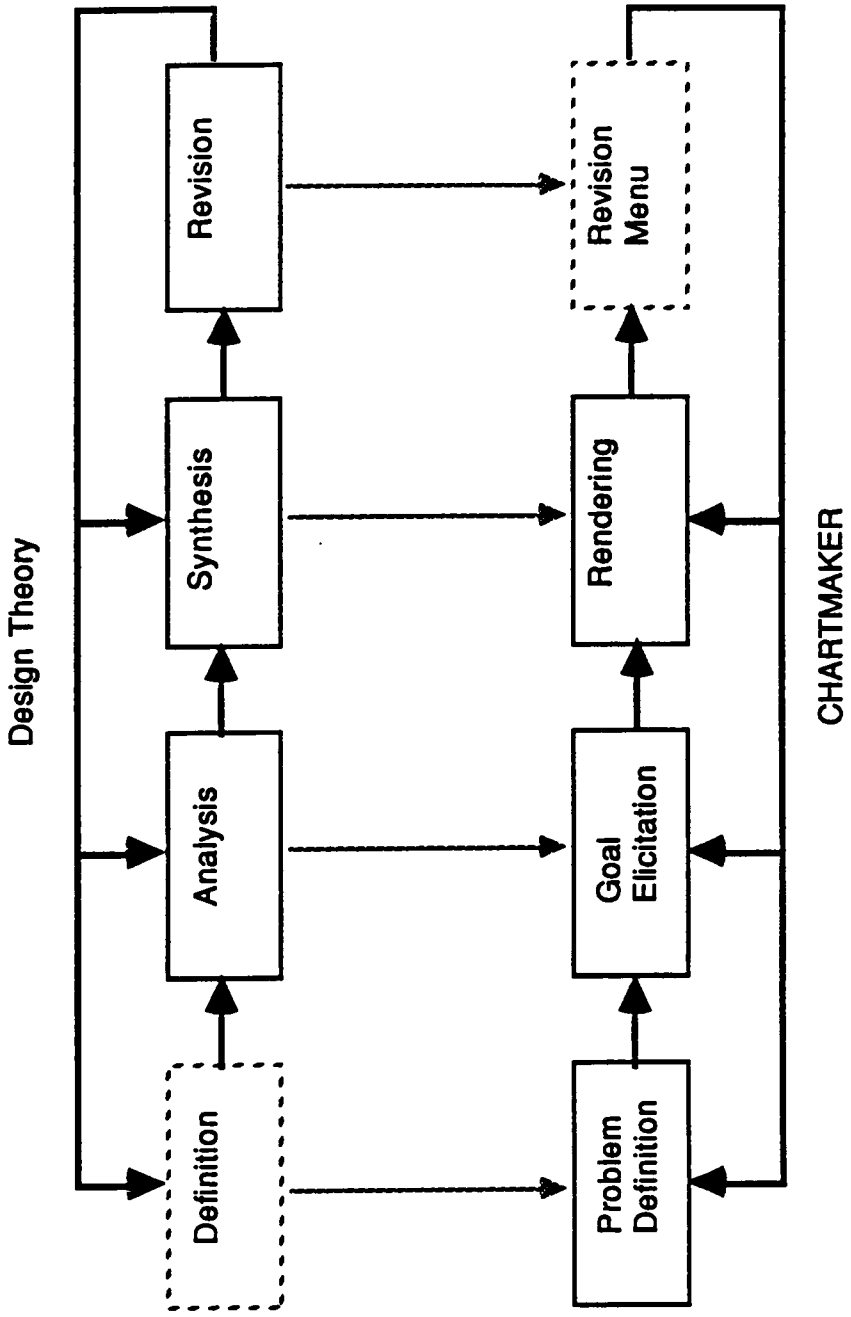

Figure 6. Design Methodology and CHARTMAKER's Architecture

4.0 Problem Solution: Problem Definition

A human consultant interviews a client using unrestricted natural language. We cannot rely on such a procedure and therefore must seek powerful yet well-defined tools for building problem models. Using an interviewing tool, the client must be able to adequately describe his situation as he sees it, preferably in his own terminology, and the automated consultant must then be able to interpret the client's situational model and render an effective design based on that model and its background knowledge about the domain. As noted before, it is difficult to achieve this balance between the client and the expert system, primarily because the human and the computer view the world from two radically different perspectives. This chapter focuses on the problem definition phase: the initial phase of the consultation where the client and the consultant work together to define the client's problem situation. From this situation model, the consultant determines the underlying goal (Chapter 5) and suggests an appropriate design (Chapter 6). Of the information acquisition strategies discussed in the literature review, two show promise for satisfying requirements of both the man and the machine: personal construct theory and system dynamics modelling. Both acquisition methods possess a high degree of flexibility as well as a methodology that can be readily programmed.

4.1 *Personal Construct Theory*

Personal construct theory (PCT) has been successfully implemented on a computer for the purpose of extracting and grouping the constructs of a client's belief system. As noted in the literature review, personal construct theory is useful for eliciting an individual's model of a situation, and if the interview is performed with respect to a particular problem, such a model could be construed as a problem model. Originally, personal construct theory arose from the concern about how a person categorizes and classifies his environment, and all the theories surrounding how a person actually does this suggest that the individual uses a system of organization together with interrelationships between the components of the system that interact with the structure to produce interdependencies. Kelly, the originator of PCT theory, states that each person constructs his own version of reality using a hierarchical system or lattice of personal constructs [Shaw, 1980]; and, if the client's "version of reality" can be extracted and understood by an automated consultant, PCT would be a useful problem definition tool.

The prospect of creating a psychological model of the client's world is, at first, appealing. For example, a computer model of an expert's view of the world (or at least, his domain of expertise) would be a useful knowledge acquisition tool. Not surprisingly, researchers at Boeing Laboratories have created a system based on PCT to interview experts about their domain of expertise [Boose, 1985]. An example knowledge acquisition session with an expert in computer languages will help illustrate how PCT can be used as an interviewing tool.

The automated knowledge acquisition system would first elicit from the expert the basic concepts of the domain, which for this example could be FORTRAN, PROLOG, COBOL, and assembly. The system would then present the expert with a "triad" of concepts and ask how any two of them are alike (and thus different from the third). If the system displayed FORTRAN, COBOL, and assembly, the expert might respond with "fast execution speed" for FORTRAN and assembly. This

associates "speed" with FORTRAN and assembly while implicitly disassociating speed with COBOL. Triads are presented and concepts elicited until all the concepts are adequately defined. If some concepts are too similar in their constituent constructs, they will be presented as part of a triad in an effort to enhance the differences between them. Once automated knowledge acquisition is complete, a consultation with a client could be carried out by effectively reversing the process and eliciting constructs from the client. The constructs could be characteristics or requirements of the problem. For example, a client may have a problem that requires mathematical formulation and high execution speed. The expert system could then match the characteristics of the problem to the characteristics of the languages that it knows about. As the example illustrates, PCT shows promise for heuristic classification problems, but it may not be as well-suited to defining a *situation* from which a problem can be derived.

4.1.1 Problems with Personal Construct Theory

The greatest strength of personal construct theory is, unfortunately, its greatest weakness in the context of this thesis. Personal construct theory constructs *personal* models of the world. The model is tailored to the client's perspective since it consists of both his concepts and his constructs. Since the entire structure of the model is in the client's format, there is no effective way for an automated consultant to apply background knowledge to it. While freedom of expression is vitally important in the design domain, the use of PCT in this context is tantamount to allowing the client to ramble freely, and perhaps incessantly, without any help or direction from the consultant. The use of PCT is analogous to the empathetic approach to consulting, and aside from the inherent disadvantages of the empathetic approach with a human consultant, there are far greater difficulties when the consultant is a computer. For an expert system to operate effectively with a problem model, there must be some facets of the model are generally assumed to be constant. With a human consultant, certain assumptions are made, such as the client will speak in English, may use certain mathematical tools, etc. With an expert system, the restrictions are necessarily greater. Some basic

assumptions about the problem model must be made, or the system will be unable to operate on the model since it will not know what to expect, even in a generic sense. With PCT, the model is defined in the client's words: the concepts are input by the client, and they are classified on the basis of constructs which are input by the client as well. The constructs supply the underlying structure of PCT, and unless the system has a natural language facility, it has no way to extract any semantic content from either the concepts or the manner in which they are grouped. The "empathetic" nature of PCT makes it quite useful in building a model of the client's world, but the model is so tailored to the client's perspective it is of little utility to the automated consultant. Beyond this fundamental flaw, PCT lacks other facilities useful in building a situational model. PCT is not a direct modelling technique. It does not allow the client to explicitly build a "nuts-and-bolts" model of the situation, consisting of parts and relationships between those parts. Because the model is indirectly constructed from the triads of concepts presented to the client, PCT lacks an intuitive "feel" since the client is not entirely sure how the model is being constructed or if it is really what he intended. Furthermore, a client cannot easily specify causality in a PCT model, and causality and interrelationship are primary issues in the design domain. Finally, PCT is a classification methodology, and while all consultations use some form of classification, most consultations in the design domain are not as amenable to the application of raw classification techniques. The programming language example using PCT is effective but operates as heuristic classification. That example is much more like a diagnosis problem in that it could be implemented with a decision tree. PCT theory is a viable alternative in the diagnosis domain for well-studied problems such as medical diagnosis, but it suffers from serious limitations when applied to the design domain because it lacks a generic but standard structural quality that an automated consultant can understand.

4.2 *System Dynamics Modelling*

To understand how the client perceives his problem or situation, a "cognitive map" of his beliefs may prove useful. Cognitive mapping is a technique that portrays beliefs, attitudes, ideas, and their relationship to one another in a form that is amenable to study and analysis. Cognitive maps differ from repertory grids in PCT in that they are *directly* constructed by the client and not by indirectly elicited constructs. The map consists of concepts and links that represent relationships between those concepts. As with PCT, the concepts are provided by the client, in his own terminology, but in System Dynamics Modelling (SDM) the relationships between these concepts are explicitly provided by the client as well. A relationship in the model is represented by an arrow that indicates the direction of the causation. Usually, if the concept at the point (head) of the arrow varies in the same direction as the concept at the base (tail) of the arrow, then a "+" sign is affixed to the link. If the head concept varies in the opposite direction, a "-" is used. From this basic structure, highly complex situational descriptions can be created. System dynamics modelling is a field of research devoted to the creation and study of cognitive maps and their application to dynamic systems. SDM provides a strong scientific basis for cognitive mapping, as evidenced by its wide variety of applications: modelling situational political influences (Figure 7 on page 59) [Richardson and Pugh, 1981], inventory tracking and control (Figure 8 on page 60) [Coyle, 1977], and a magazine publisher's business concerns [Eden, Jones, Sims, 1983] (Figure 5 on page 33).

The construction of an influence diagram involves making statements about how the system actually works. Hence, a link such as A = = > B is a diagrammatic representation of the statement that factor A causes factor B, or, more generally, factor A influences factor B, and variations in A will manifest themselves as variations in B. While "A influences B" is the most basic example of influence in a SDM model, relationships are often more complex, involving multiple influencing factors for a given factor, which in turn, impact on other factors in the model. Since the links are

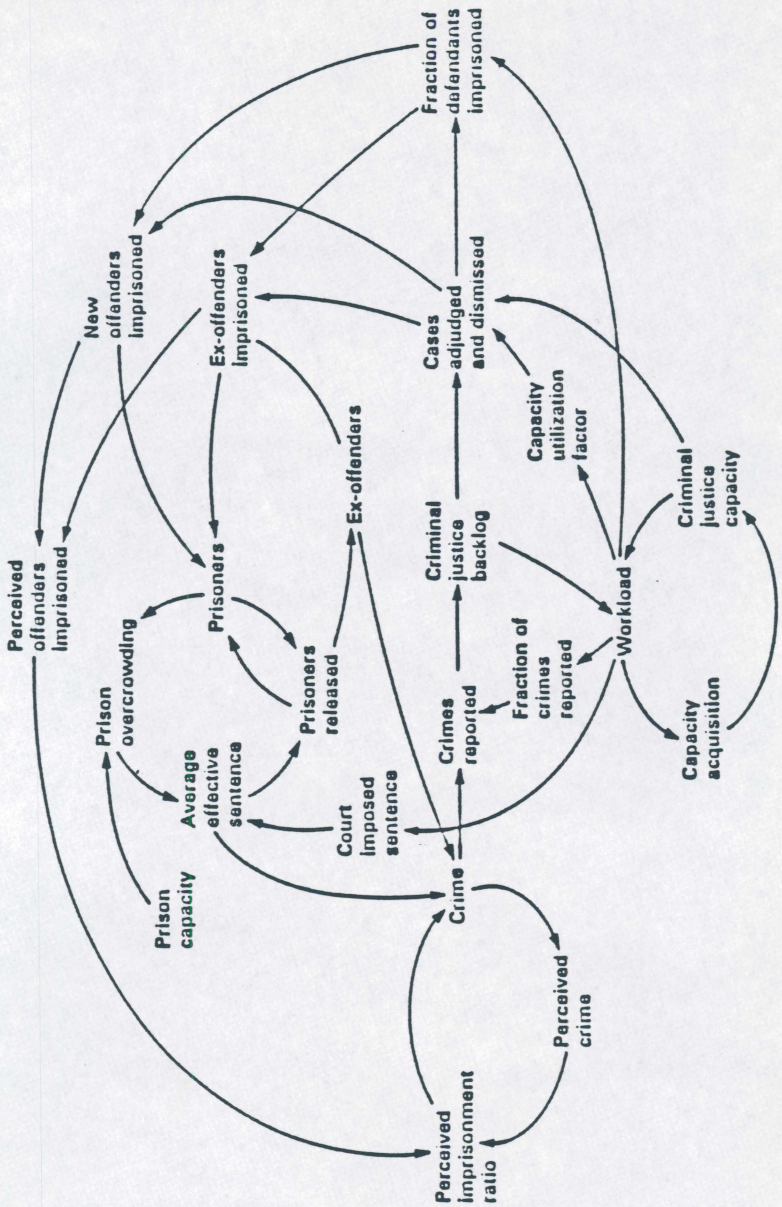

Figure 7. System Dynamics Model in Political Science: [Richardson and Pugh, 1981]

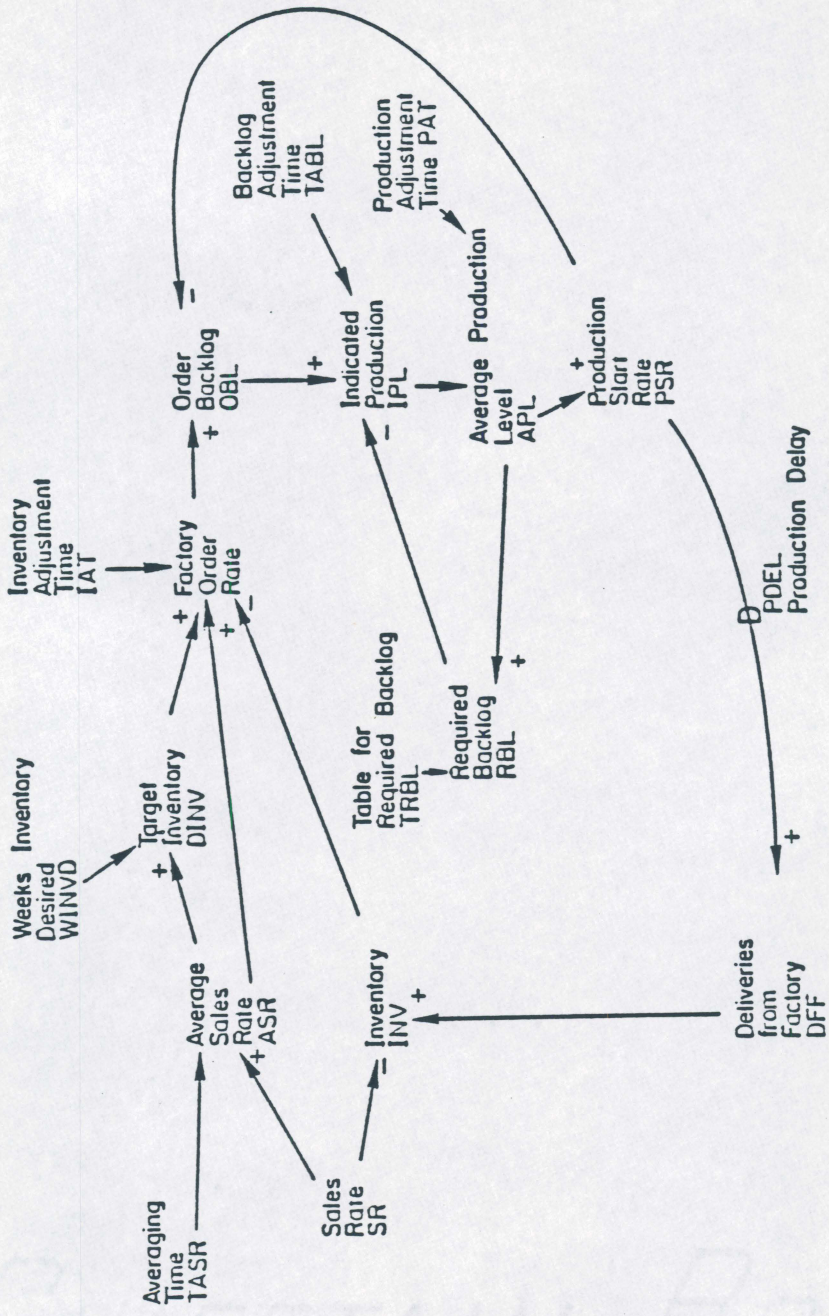

Figure 8. System Dynamics in Inventory Management: [Coyle, 1977]

one of the most important features in the model, there needs to be some justification for linking two factors together. In standard SDM theory, there are are six methods of link justification:

1. Conservation considerations

2. Direct observation

3. Instructions to that effect

4. Accepted theory

5. Statistical evidence

6. Hypothesis or assumption

Conservation considerations are commonsense observations of "what goes in must come out somewhere". Tangible items like people, cash, and merchandise do not usually disappear from the system unless the analyst determines that they are no longer relevant to the model. Conservation considerations can best be described as tangible flows of physical entities. Direct observation, as the name implies, is usually practiced by the model builder from his own personal experiences with the system. The third method, justification by instruction, entails validating links by making the system operate in accordance with them. In effect, the system is made to operate in a stipulated way, rather than described as functioning in an observed way. The fourth method, accepted theory, is justification on the basis of known laws or theories. An economic theory, for example, could be employed in an economic model to enhance the relational structure beyond what would be created with the other methods of justification. In some cases, statistical evidence can be used to infer causality between factors in the model. A major distinction between this technique and the others, is that statistical methods are used to determine the *parameters* for a given link and not the actual existence of a causal link. The final and quickest way to build an SDM model is through hypothesis, assumption, and belief. This approach enables the builder to tailor the model precisely

to his purposes as well as easily replace alternatives that may be challenged by other relationships in the model. Naturally, this approach has some disadvantages. The hypothesis approach is unscientific and may lead to unproductive controversy since the information (and thus the model) will vary from person to person. More importantly, the results of the model depend on the information in the model, and any conclusions drawn from the model are valid only with respect to the underlying hypotheses [Coyle, 1977]. The importance of the links in the SDM model is underscored by the variety of possible methods for justifying them. The links of the model create the actual influence structure which contains a wealth of information about the problem.

In system dynamics an influence model must pass a fundamental test to be considered a dynamic model. The model must meet a closure property such that it has at least one feedback loop, so that:

> starting from any point in the influence diagram it must be possible to return to that point by following the influence lines, in the direction of causation, in such a way as to not cross one's track. [Coyle, 1977]

One possible exception to this rule are links from parameters that are input to the model from the outside world. The purpose in requiring closure in the dynamic model is that the model's dynamic behavior derives from the operation of feedback loops in a closed system. If a model has no feedback loops, it is a static influence model since further iteration will not change the model.

While both system dynamics modelling and personal construct theory create models of the client's reality, SDM is a more direct approach to model building. In both SDM and PCT the user enters his concepts into the model, but in SDM the relationships are directly added to the model by the client, instead of indirectly by the client's elicited classification scheme. The benefit of SDM lies in the user's ability to directly indicate causal influences between the concepts, as well as indicate if the influence is positive or negative. Although the client has more direct control over the model construction, he is required to use the SDM structural paradigm. The "A influences B" building-block is flexible enough to permit accurate system formulation by the client, but restricted enough to provide the consultant with a known structural format. The flavor of SDM is more like a negotiative approach to consulting in that both the client and consultant compromise to facilitate a better understanding of the situation by *both* parties. The compromise is vital to an expert system

implementation since it gives the system a basic semantic framework to operate from while allowing the client sufficient freedom to define his problem adequately.

The basic definition of a system in the context of SDM is "a collection of parts organized for a purpose". While the parts of the system are important, the *organization* of the components is the key to the representational strength of system dynamics. Since system dynamics is actually a dynamics modelling paradigm, this use of this structural feature as a representation method is overlooked. In the design domain, the actual concepts are not as important as the *relationships* between the concepts since a change in one concept usually affects the other concepts in the model.

The domain of chart design provides an added motivation for SDM. Time is a prevalent issue in charting. The most common classes of charts usually involve time as a key aspect, and even if time is not directly involved in a chart, there is usually some dynamic element against which the other quantities are plotted. The quantity plotted along the x-axis is typically an independent variable, and the quantities on the y-axis are affected by changes in the independent quantity. This "dynamism" in charting is a form of dependency, and one of the strongest representational components of SDM. Even the simplest influence structure in SDM, "A = = > B", implies a relationship between the "variables" A and B. Relationship of variables is a basic chart type, and this intuitive link between chart types and certain system dynamics models provided the initial impetus for exploring SDM as a problem model for the design domain.

4.2.1 Modifications to the System Dynamics Paradigm

Since system dynamics is a dynamic modelling paradigm, there are aspects that are not requisite for a static problem model. For example, delays are often used in SDM models to indicate a non-immediate causal influence between two concepts. A delay is not necessary for a static model, but the ability to express different types of causal effects between concepts may prove useful and

even necessary in more complex domains. System dynamics also uses positive and negative signs on the links to indicate the basic nature of the influence. While these are vital to the dynamic model, and perhaps a more generic problem model, they are not necessary for this domain primarily because charts typically *display* positive and negative influences between concepts. For more complex domains that require a more complex problem model, different types of causality may be incorporated into the model and interpreted by the background knowledge. SDM offers significant expressive capability along this dimension. In CHARTMAKER, the justification of links is left to the client's discretion. Since the problem model seeks to represent the situation as the client perceives it, the client will most likely express the relationships on the basis of hypothesis, belief, and assumption. The client must be aware, however, that an incorrect model will probably produce incorrect results. As noted earlier, system dynamics also places validity constraints on a model. The most important is that of closure. A model must have at least one "loop" to be considered valid. This makes sense since it's the only way change can propagate through the entire model; however, such a requirement is not necessary for a problem model. To create the basic paradigm for problem definition, the basic nature of system dynamics -- generic modelling based on user-defined concepts and relationships -- is preserved, and the unnecessary aspects such as delays and validation rules, have been eliminated.

4.3 Implementation

The design problem framework for a true consultant requires more than a single representational structure. The basic design procedure is analysis, synthesis, and evaluation, and the automated consultant requires the additional step of problem definition. These four steps represent four distinct phases of a design consultation, and the processes involved in each of these steps is quite different. The system dynamics model forms only one of a series of internal representations that eventually lead to a rendered chart. The theory of a design consultant presented here requires a

series of three internal representations, each associated with a distinct phase of problem interpretation: a situation model, an intent model, and a rendering model. This "pipeline" of representations is a major departure from traditional representations. Instead of forcing all aspects of a consultation into a single structure, the structure is designed to suit the nature of each particular aspect of a given problem. For the design domain, the entire consultation can be broken down into three basic areas: the client's conception of the situation, the client's primary interest (goal), and the model for the rendered chart (Figure 9 on page 66). Through the pipeline, each ensuing model is constructed using information derived from the previous model and from background information already known about the design domain.

An extended example will help to illustrate the theoretical principles of the automated design consultant. Consider a user who is making a presentation based on a set of data regarding radio sales. He has data for a wide variety of radios ranging from Walkmans to console stereos, and he isn't quite sure how this should be presented to the audience, either in terms of which charts he should use, or what those charts should contain.

4.3.1 The Situation Model

At the outset of a consultation with CHARTMAKER, the client is presented with a menu that allows him to construct a logical relationship diagram showing the basic concepts of the problem model and how they relate to one another:

- ADD a concept to the model
- DROP a concept from the model
- LINK two concepts (causal or attribute link)
- SNIP a link
- DISPLAY current problem model

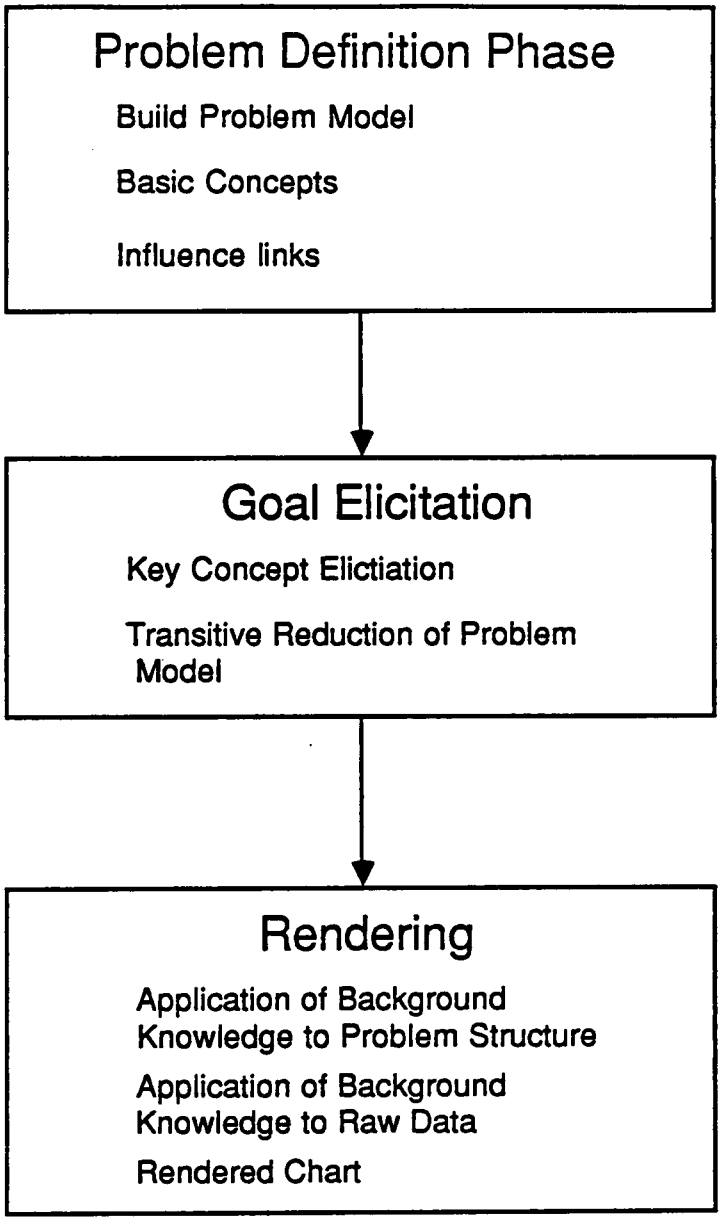

Figure 9. System Design for CHARTMAKER

- FINISH building the model

The client first ADDs the basic concepts of the model, which for this example would be walkmans, discmans, portables, console stereos, and total radios. He then adds the causal influences (if any) to the model by LINKing the concepts together. The primary concept in the model is "total radios" since all the different types of radios directly influence the number of total radios. Using this interpretation of the situation, all the supporting concepts such as walkmans, discmans, and portables, can be linked to the "total radios" concept. For example, using the A = = > B basis, a part of the model would be walkmans = = > total radios. If the client wishes to see what the model looks like, he can DISPLAY it. CHARTMAKER lists the relationships by the concepts that they affect. For a given concept, both the "influences" and the "is influenced by" links are given. The user-defined situation model is shown in Figure 10 on page 68.

4.4 Conclusions

SDM offers a useful compromise between the client's and consultant's perspectives. While the client is free to use his own terminology and express the relationships between the concepts in the model, he must use the "A influences B" architecture of SDM. This standardized portion of the model allows the automated consultant to apply its background knowledge to a known structural component while the user still retains a reasonable degree of free expression. Alternatively, PCT is too empathetic in that everything about the problem is defined by the client, and the automated consultant has no way to interpret the situation model effectively beyond the context of heuristic classification. While SDM is a viable tool for problem definition, it does not resolve the entire entire representational problem either. CHARTMAKER needs to be able to understand the problem model. From the generic problem description, the expert system must determine what the

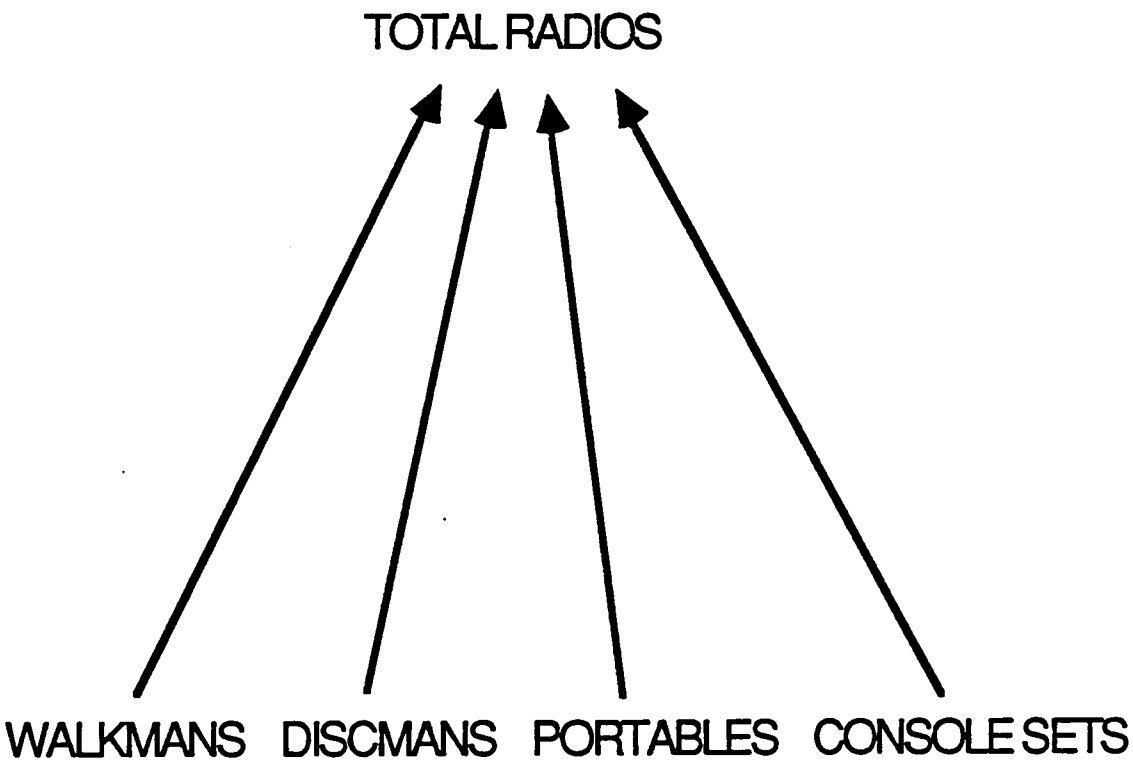

Figure 10. Problem Model for Radio Data

underlying problem is and map the problem into its background knowledge to produce a solution. Chapter 5 explains in depth how this is accomplished

5.0 Problem Solution: Goal Elicitation

While the problem model provides a useful way for the client to express himself, the automated consultant needs a somewhat less arbitrary structure to analyze and interpret. To provide a more suitable model for CHARTMAKER, the system transforms the client's "cognitive map" of the situation into a more manageable and useful representation. The objective of the reduction process is to discern the nature of the underlying problem in the client's situation. In a consultation, the client often provides more information than is necessary since the client has his own ideas concerning what is important about the problem. For example, in the auto mechanic example from the previous chapter, the customer may offer the information that the car has been overheating recently. While this information may be indicative of another problem, it is not particularly relevant to a cracked distributor cap, and the mechanic needs to "screen out" such irrelevant information when searching for a solution. The mechanic must not be overly critical in screening out information, however, since knowledge that "the car stalls after car washes" may not be of vital interest, but it can provide a basis for further inquiry which can provide useful information such as "the car only stalls when it is wet". The automated consultant must find a way to focus attention on the relevant concepts of the situation and ignore the ones that have marginal or no impact on the solution without accidentally ignoring the impact of seemingly relevant concepts.

5.1 Background Knowledge

A key component of an expert system is its *background knowledge*. Background knowledge is what the system knows about its domain of expertise. For example, R1, the VAX system configuration expert system, has background knowledge about VAX components as well as knowledge about how these components can be combined to form a functional system. In conventional expert systems, the domain knowledge is represented with rules. While rules form an integral part of domain knowledge, they are not the sole source of a human consultant's expertise. A supporting hypothesis of this work is that the underlying representational structure is a vital facet of background knowledge. Although it is impossible to conclusively determine what mental models a human expert uses, it is reasonable to assume his representations derive from the domain theory or methodology. The representational pipeline used by CHARTMAKER follows basic design methodology and uses three different representations for the different facets of a consultation. Moreover, CHARTMAKER employs transformations that create the next representation in the pipeline from the preceding model and background knowledge (rules) about the models. Contrary to the standard expert system paradigm, CHARTMAKER's background knowledge consists not only of rules, but also of representations and representation transformations.

5.2 Application of Background Knowledge: Focusing the Analysis

In CHARTMAKER, the cognitive map of the client's situation is still lacking in one critical respect. The map must in some way provide an indication of those concepts most relevant or most important to the actual problem. That is, the actual problem is derived from the situational de-

scription (just as the mechanic deduces the real problem from a situational description about the car), and either the client or the consultant (or both) must identify the salient features of the consultation since these will have the greatest impact on the problem definition and thus problem solution. The process of key concept selection can follow any of the three paradigms discussed earlier: coercive, empathetic, or negotiative. With the coercive approach, the consultant determines what the key facets of the problem are, as in the auto mechanic example. A consultation in the design domain, however, is really an examination of what the *client* feels is important. For effective design, either the client should decide what the key facets are (empathetic), or the client and the consultant should collaborate (negotiative) in deciding what aspects of the problem merit the most concern. In any case, the actual situation must be transformed into a more manageable model that the consultant can more easily understand and interpret. A complexity reduction is necessary since the client can compose an arbitrarily complex problem model, and the number of possible structures, even for reasonable models, numbers in the thousands. One solution to this problem is to exhaustively code background knowledge to interpret all the possible problem models. This approach is neither realistic nor a reasonable interpretation of how a human consultant formulates a solution for a complex problem model. A more intelligent approach exploits structural similarity and effectively "reduces" the problem model to a structure indicative of an entire class of problems. CHARTMAKER, like a human consultant, uses background knowledge to convert situational descriptions into problem classes as well as map classes of problems into design solutions. The process of eliminating unnecessary detail and focusing on the high-level relational structure of the problem is known as clustering since the concepts are grouped based on their similarities and the relational analysis transacts between groups of concepts instead of individual concepts. This grouping or "clustering" of concepts effectively reduces the number of concepts and relationships while preserving the underlying relational structure of the problem.

Clustering reduces a cognitive map to a more manageable structure based on hierarchical relational dependencies. Using clustering, the consultant first identifies key concepts within the model based on his background knowledge of the domain. A key concept is selected on the basis of its appro-

priateness for a particular area of concern. All concepts that have consequences for a given key concept (on the tail of an arrow or sequence of arrows that leads to a key concept in the cognitive map) are considered part of the group for that concept. The cognitive map is then reduced by replacing groups of concepts with symbols and then adding the directed influence links between the groups. To illustrate, the concepts in the the generic problem model in Figure 11 on page 74 can be grouped based on the key concepts in the model (Figure 12 on page 75) [Eden, Jones, Sims, 1983]. A concept is a member of a particular group if it influences, directly or indirectly, the key concept of that group, and since a concept can influence more than one other concept, a given concept may be a member of more than one group. The key concept of a group linked to a concept in another group is not considered part of that group. Instead, the key concept influences the entire group. The key concepts and their constituent influences form the abstracted problem model in Figure 13 on page 76 [Eden, Jones, Sims, 1983]. This "intent model" is the second in the pipeline of representations.

5.3 Implementation

One of the problems with automating the process of clustering lies in key concept selection. Key concepts are selected for their appropriateness as descriptions of an area of concern, and if the automated consultant independently determines what the key concepts are, it must have some idea what the concepts actually mean. Since the client described the problem in his own terminology, the automated consultant would need a natural language facility to understand the concepts used in the client's description. Instead of adding this complication to the system, it actually makes more sense to follow the underlying paradigm of the design domain. Unlike the diagnosis domain where the consultant is primarily concerned with the observable symptoms of the problem, the design consultant is primarily concerned with the client's interests and preferences. The ultimate design should meet the client's described purposes, and the client has a far better idea what his purposes

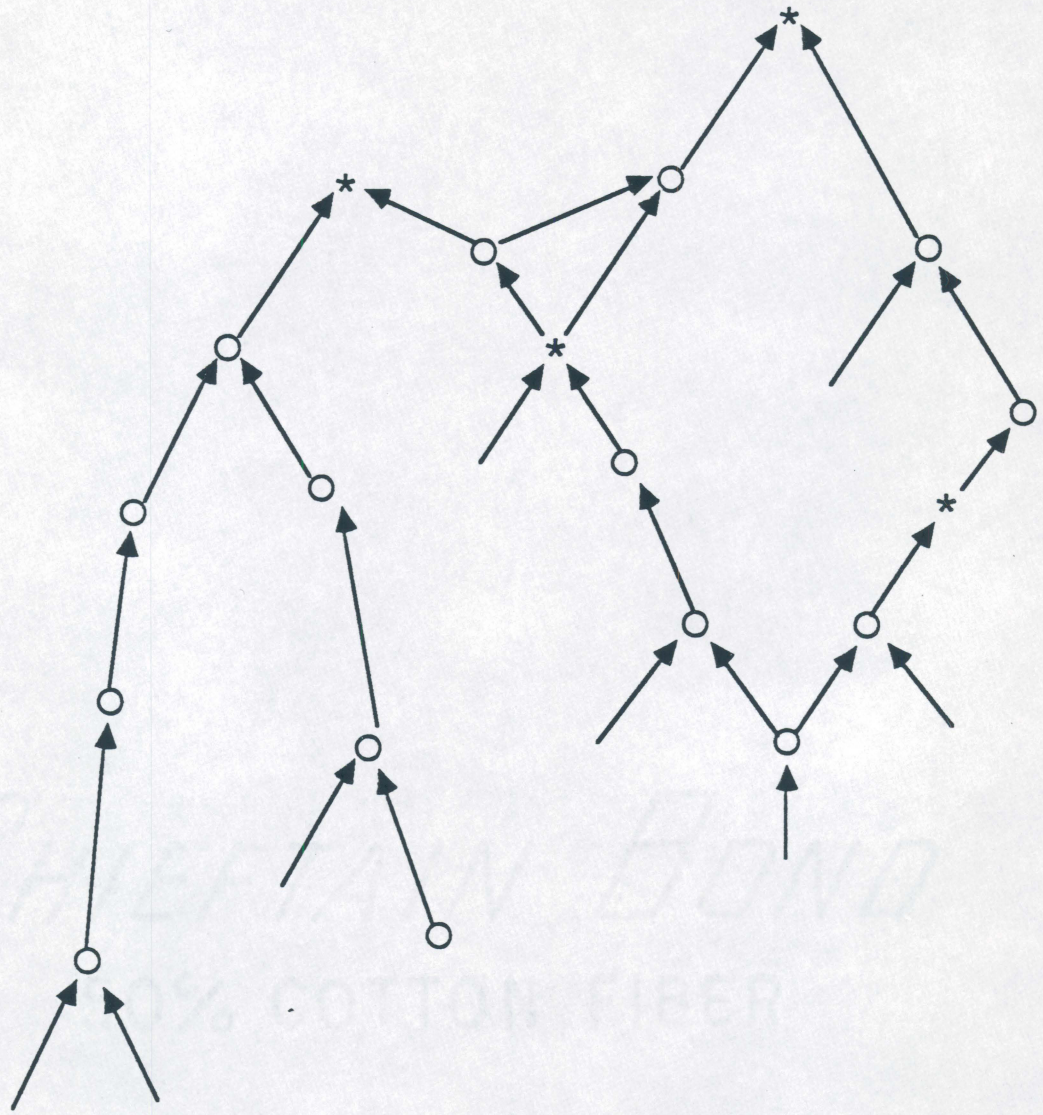

Figure 11. The Generic Problem Model: [Eden, Jones, and Sims, 1983]

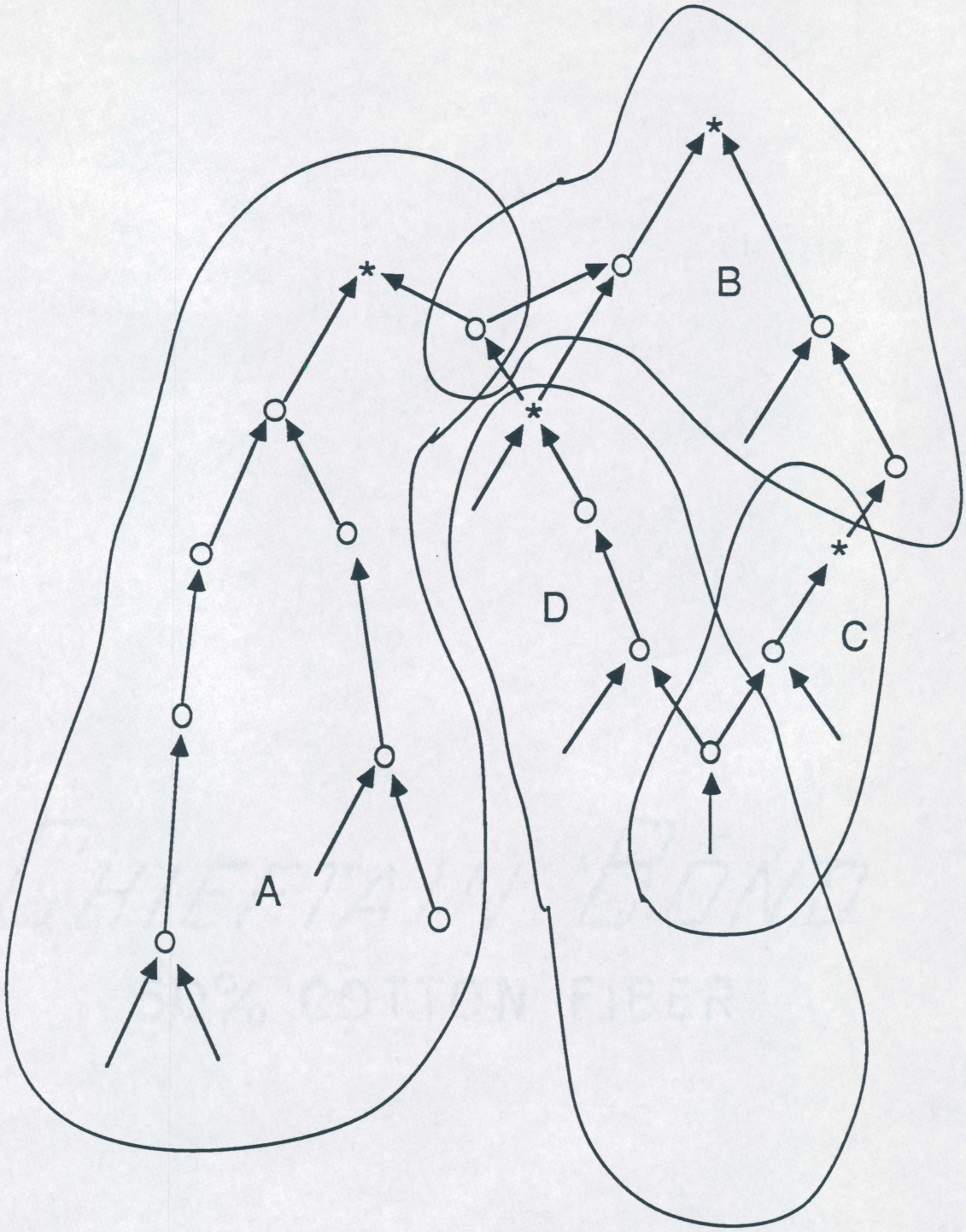

Figure 12. The Problem Model Grouped by Key Concepts: [Eden, Jones, and Sims, 1983]

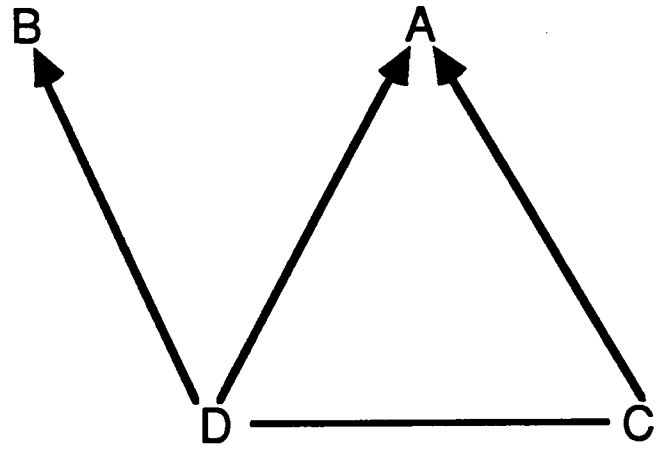

Figure 13. The Abstracted Problem Model: [Eden, Jones, and Sims, 1983]

are than does the consultant. Since CHARTMAKER operates in the design domain (albeit in a restricted sub-area), it is reasonable to allow the client to specify the key concepts of the model. Once the key issues are determined, the automated consultant performs a transitive reduction between the key aspects of the model. While CHARTMAKER does not actually "cluster" the concepts together, it does produce the same result. In transitive reduction, a key concept is selected and all its influence links are traced. If an influence link terminates at a non-key concept, all the influence links from that concept are traced. If the link terminates at another key concept, then an abstracted influence link is established between the two key concepts and added to the abstracted problem model. This process is repeated until all abstracted links have been found. The resultant abstracted problem model which consists of the key concepts and their abstracted links is identical to the one produced by clustering.

In CHARTMAKER, once the user is satisfied with the situation model, he selects the FINISH option, thus completing the first phase of the consultation. CHARTMAKER then displays all the concepts in the model and asks the user to identify the key concepts. In the charting domain a key concept reflects a basic issue that the presenter wishes to convey through use of the chart. In the example problem, the user may decide that the key issues in the presentation are both the "Walkmans" and the "portables," and choose these as the key concepts. Once the client identifies the key concepts, the automated consultant investigates how the key concepts affect one another by analyzing the transitive closure of influence in the original model and producing a reduced model consisting only of the key concepts and their interrelationships (Figure 14 on page 78). In this model, clustering did not produce any virtual links. The lack of influence links in the reduced model is by no means a pathological situation since the isolation of the key concepts of the model provides useful structural information about the problem. Alternatively, the client may decide that "total radios" and "console sets" are the focus of this facet of his presentation and select these as the key concepts of the problem model. After transitive reduction, a different simplified problem model is produced (Figure 15 on page 79).

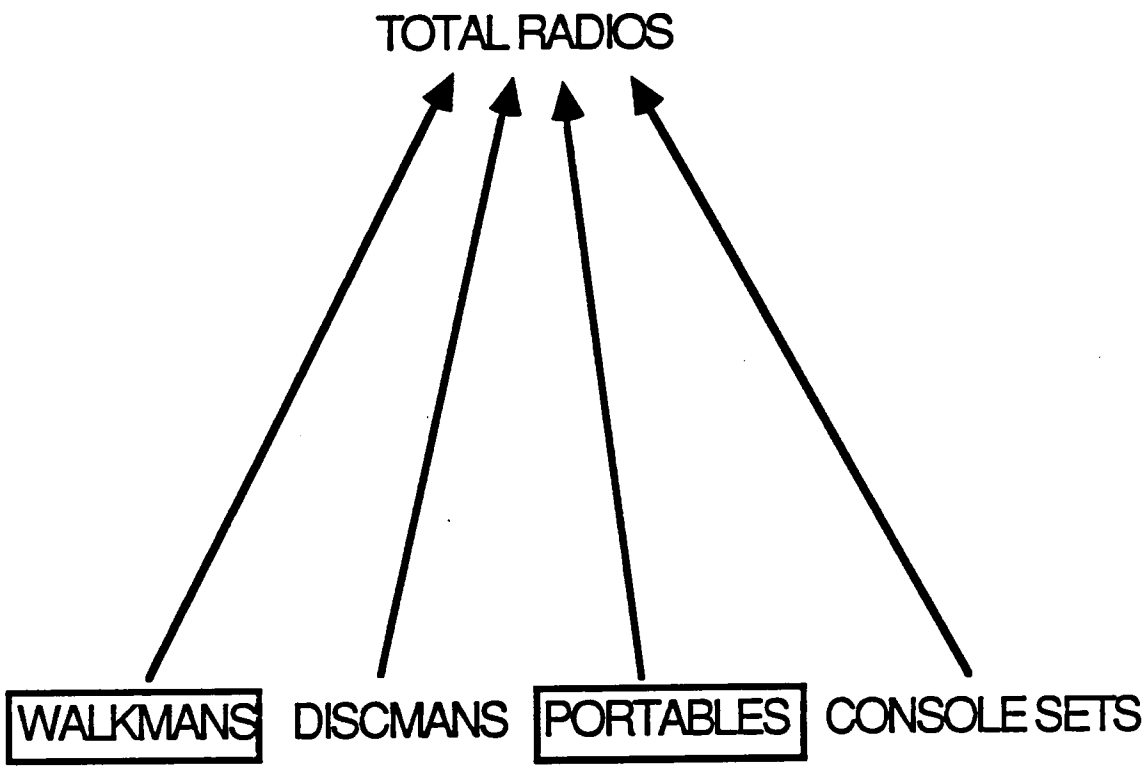

Figure 14. Goal Model For Radio Data Example (Comparison)

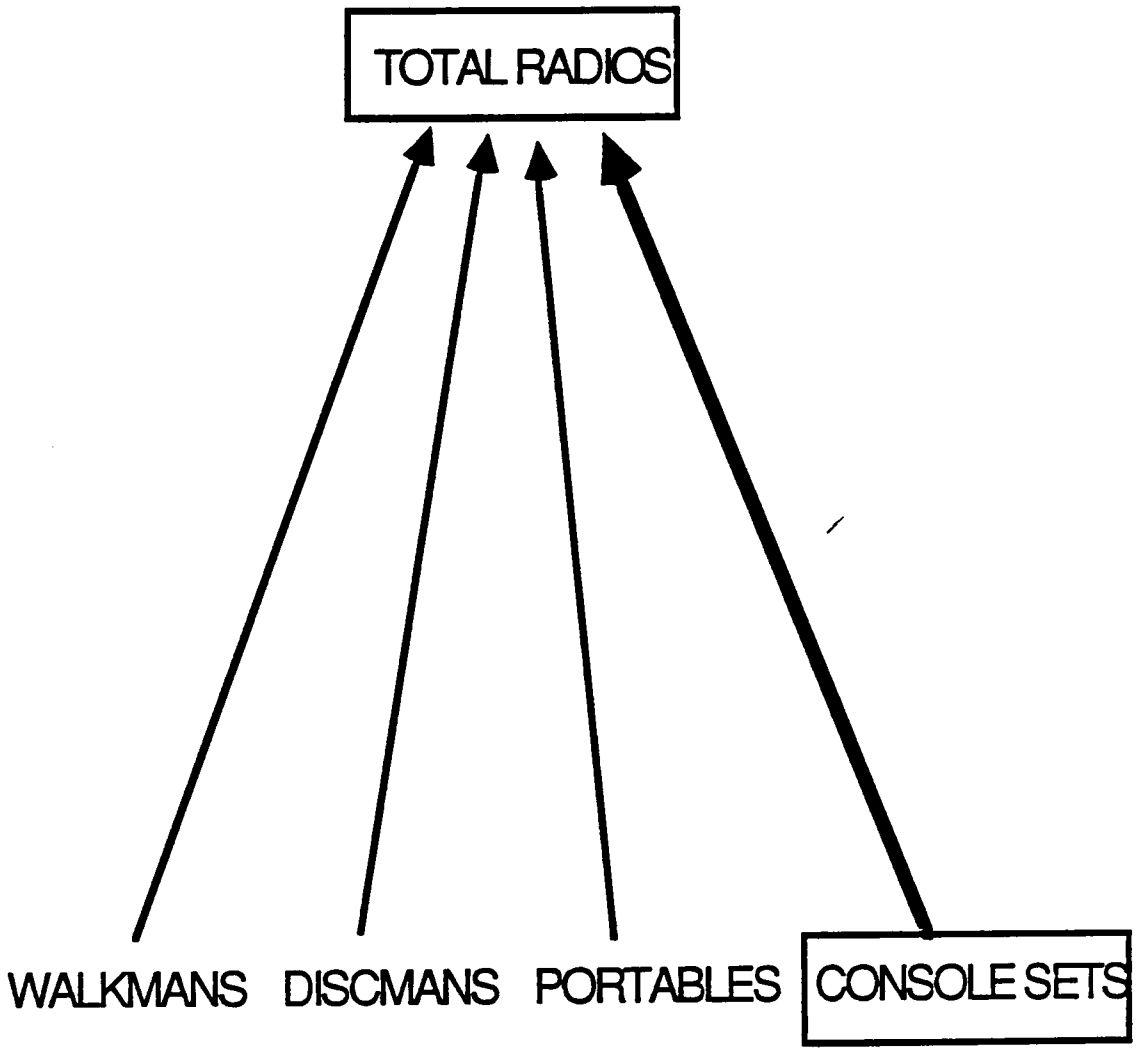

Figure 15. Goal Model For Radio Data Example (Relationship of Variables)

Since the concepts have a unidirectional virtual link, the implied goal is the "relationship of two variables". This is intuitively satisfying since a relationship chart will illustrate the causal effect of one entity on another. For example, a chart showing the relationship between boiling point and pressure shows the impact a change in one entity has on the other. In this case, the number of console sets is the independent variable, so it would be automatically labelled on the x axis. From the reduced model, the system then determines exactly what the user is trying to demonstrate.

Once the underlying problem has been deduced, CHARTMAKER applies its background knowledge in charting to produce a suitable chart for the data and the situation (for a more explicit treatment of CHARTMAKER's knowledge, see the code listings in appendix B). CHARTMAKER interprets problem structures to indicate the type of chart needed, and then examines the supporting data which dictates the exact type of chart (Chapter 6).

5.4 Conclusions

The reduction phase represents a new approach to goal elicitation. Instead of asking the user directly what type of chart he wants, the system derives this information from the user's problem description. Indirect goal elicitation is vital in the design domain. Returning to the human factors example, when a client requests a human factors specialist to design a computer screen display, the specialist typically asks questions like, "What information do you need to see most often?" or "What information do you look for first?". The consultant uses these questions to construct a mental model of the client's situation. From this mental model, the consultant focuses on the primary issues while disregarding irrelevant information and applies his background knowledge in graphic arts and human factors to the basic structure of the problem to create a screen design. In essence, the client cannot explicitly describe the type of screen he wants, but he can describe the situation that dictates the design, and the consultant uses this situational description in determining the client's

"goal." CHARTMAKER follows the same paradigm by analyzing the relationships between the key concepts and focusing on the underlying problem structure. This abstraction of the original situation model acts as a bridge between the client and the consultant. The goal model retains the basic information of the client's situation, but it focuses the consultant's analysis on those issues most important to the problem. From this analysis, the consultant can produce a rendered design (Chapter 6).

6.0 Problem Solution: Rendering

Once CHARTMAKER has obtained the goal model, it can make a decision about rendering, which constitutes the third and final phase of the consultation. From the abstracted problem representation, the automated consultant will match the problem structure to a particular type of chart and use the input data in conjunction with the data pattern to render the chart. Although the target chart class is determined by the goal model, the rendering phase determines exactly what style of chart is to be rendered. For example, a trend chart can be rendered as either a column chart or a line chart. One of the selection criteria in this case is the number of available time periods. From its basic charting knowledge, the system knows that if only a few time periods of data are available, a column chart will be more legible and is therefore preferable to a line chart. While these "rules of thumb" are only a part of CHARTMAKER's background knowledge, they are the primary component of most conventional expert systems and are usually implemented with the If/Then productions discussed in the literature review. The rendering model contains more specific information than does the goal model, such as discrete or continuous plotting, the ranges of the axes, and labelling, and makes these decisions on the basis of its background charting knowledge. This model relies more heavily on specific background knowledge since it is most closely associated with the actual domain; the rendering model is ultimately used to create the actual image.

6.1 *Application of Background Knowledge: Chart Selection*

From the goal model, the system determines exactly how to display what the user is trying to demonstrate. CHARTMAKER accomplishes this by mapping the goal model into its background knowledge of charting, and interprets the relational structure of the goal model as indicative of a particular class, or style of charts. This interpretation is actually a form of heuristic classification. Since there is no established method for mapping situational descriptions into specific charts, a reasonable guideline would dictate that the chosen taxonomic criteria reflect a basic intuition about the original situation. For example, if two concepts have mutual causation links (after transitive reduction of the situation model), the implied goal is the "relationship of two variables". This is intuitively satisfying since a relationship chart will illustrate the mutual causal effect of two entities. For example, a chart showing the relationship between boiling point and pressure shows the impact that a change in one entity has on the other. If the causation effect transpires in only one direction, a relationship situation is still implied. For example, a chart showing the impact of years of education on yearly salary is still an instance of the relationship of variables, but the causality is unidirectional. In this case, CHARTMAKER's background knowledge would dictate that since "number of years of education" is the independent variable (since it is on the tail of the influence arrow), it would be automatically labelled on the x axis, whereas in the previous example, the system would either display two different charts (one with temperature as the independent variable, the other with pressure) or ask the user which concept has the "predominant influence".

If the goal model has a structure that does not imply a relationship between variables, it can be classified into other basic areas. For example, if only one concept is selected, no relationships can be explored since there are no other key concepts to establish links with. This indicates the user is interested in emphasizing a trend in the data. Generally, trend charts show the behavior of a

single entity over a period of time, and since time is a ubiquitous issue in the charting domain, it need not appear explicitly in the situation model. Alternatively, if there are two (or more) key concepts in the goal model with no direct causal link but with a common parent, or head concept, the desire for a comparison between the two is likely, although this instance could represent a need to see each concept in the context of a "part of the whole" situation. The distinction can be made by whether or not the parent concept is also selected as a key concept. If it is, the user has indicated he is interested in the impact the child concepts have on the parent concept, or on how the parts affect the whole. Again, the situation model has the intuitive flavor of the chart it represents.

The preservation of this basic intuition is both a function of the model and the system's background knowledge. If the intuition captured by the structure of the intent model is not utilized by the background knowledge, then the representational strength of the structure is limited by the background knowledge applied to it. Other relationship possibilities exist and can arise through the use of the system but may not be recognized by the background knowledge. Reduced structures that are not instances of the three categories mentioned above can represent either combinations of the three fundamental categories or non-chartable situations. For example, the problem model may reduce to three primary concepts with mutual causation links. This is really an instance of multiple relationships between variables, and should be rendered as two or three distinct charts. Alternatively, the reduced structure could consist of a single concept with a self-referent causal link. This structure does not currently represent a chartable situation, but the background knowledge of relational structure could be enhanced to interpret this structure as a certain type of chart, if one exists.

Of course, since a user may improperly specify the problem, the system needs to be capable of detecting and resolving inconsistent or incomplete problem models. For example, if there is some ambiguity as to which chart should be rendered, or if no chart matches the problem structure, the system allows the user to augment or alter the original problem model and respecify the key concepts in an attempt to resolve the inconsistency. If the problem still cannot be resolved, the system

permits the user to choose whatever type of chart he feels is appropriate and then renders it if the required supporting data is available.

6.2 *Integrating Data with the Chart*

At some point in the consultation, an association must be made between the concepts that the client uses in the problem model and the actual data. If the client specifies "Walkmans" as a key concept in the model, there should be some data (e.g. number of Walkmans sold) that corresponds to the concept. Actually, an entire class of values may correspond to a particular concept (e.g. number of Walkmans sold over a certain period of time) since the concept may be affected by the presence of a time component in the charting situation. This conceptual mapping can be achieved a variety of ways. The simplest way is to present the user with a "data pattern" and instruct him to input the data values corresponding to the displayed concept. A more advanced implementation could employ some rudimentary natural language to determine how the available data relates to the concepts in the goal model.

6.3 *Implementation*

Before rendering the chart, the system elicits the raw data from the user. To construct a data pattern, the system displays the key concepts of the model and asks the user to enter a data value, in the prescribed order, for each concept. The user enters his data, line by line, and after he is finished, the system analyzes the nature of the data to determine the exact chart type. Once CHARTMAKER determines the appropriate chart type, it produces the rendered chart.

Returning to the example using radio data (Chapter 5), the client may have a dataset consisting of the following entries:

walkmans	1981	30000
discmans	1981	2000
auto sets	1981	250
portables	1981	250
home sets	1982	40000
.	.	.
.	.	.
.	.	.
portables	1987	7000
home sets	1987	300

For the first example where the client indicated that "Walkmans" and "Portables" were the key issues, the system would display the data pattern:

Walkmans	Portables	Time

and request the client to use that format to enter the data values. Note that "Time" appears in the data pattern, but does not appear anywhere in the data model. CHARTMAKER, through its background knowledge, knows that since a trend chart is going to be displayed, a time component must be part of the data pattern. The user may then enter the data:

700	100	1984
750	150	1985
688	300	1986
521	487	1987
372	698	1988

The rendering model for this example is shown in Figure 16 on page 88 and the rendered chart is shown in Figure 18 on page 90. In the second example, the system presents the data pattern:

Total radios	Console sets

and the user responds with:

4900	220
5300	300
5700	410
7000	440

and produce the rendering model shown in Figure 17 on page 89 and the rendered chart in Figure 19 on page 91.

Once the image has been rendered, the user has the option of returning to the original problem model (to make modifications), returning to the problem definition stage to create a new model, or rendering a different chart based on the input data. The ability to return to the previous problem model allows the user to chart different aspects of a larger problem by changing his emphasis through the key concepts.

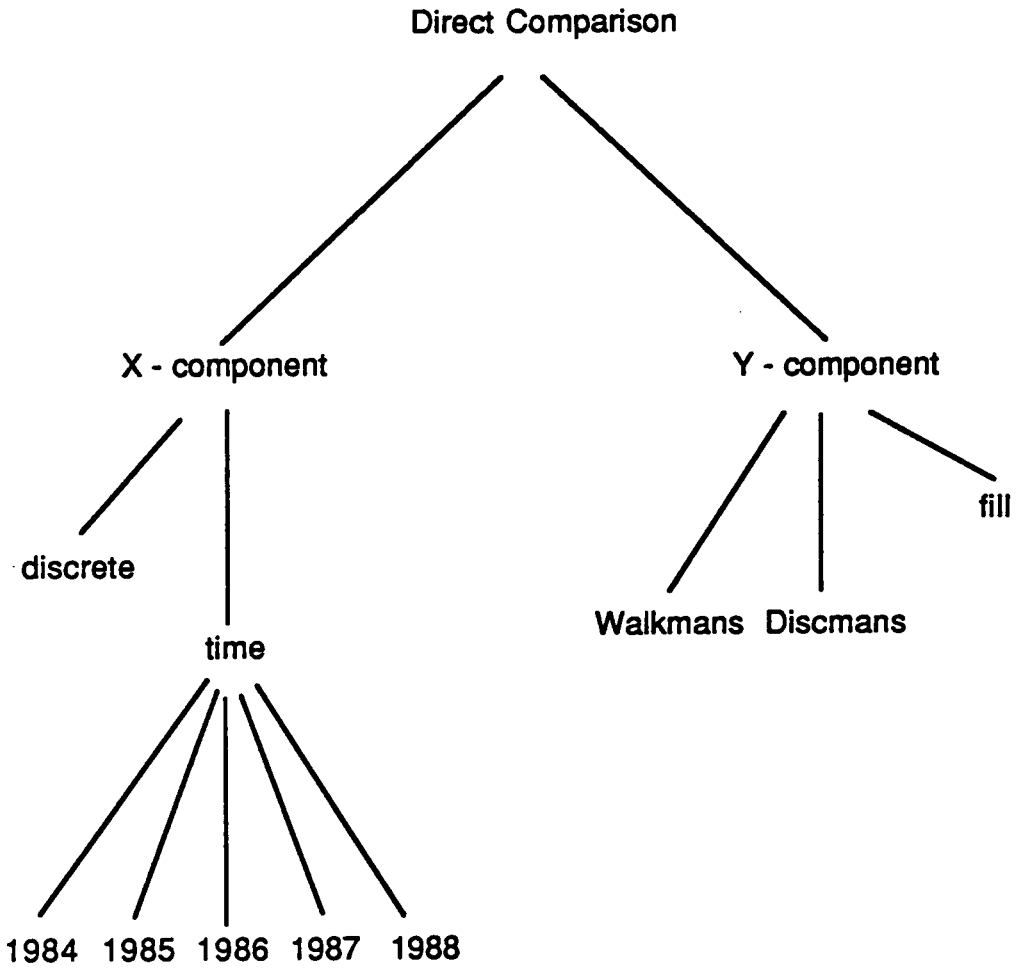

Figure 16. Rendering Model for the Radio Data Example (Comparison)

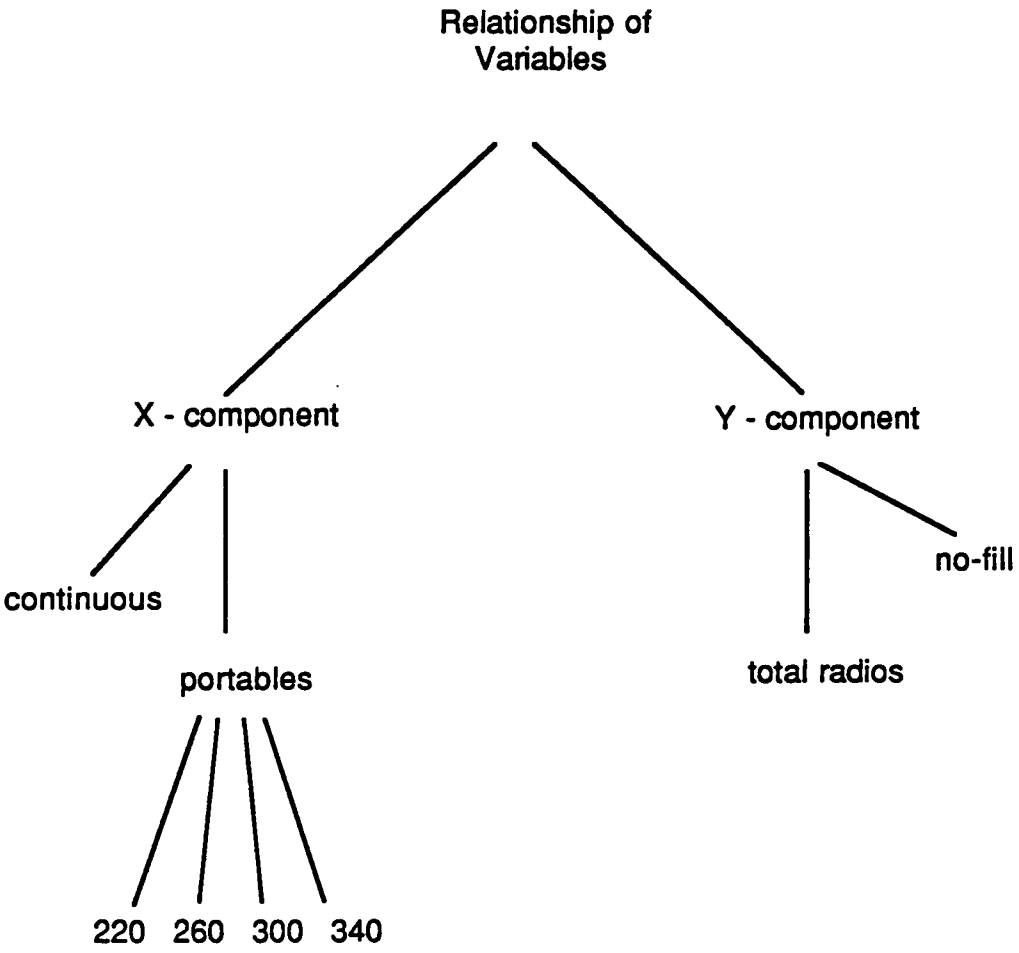

Figure 17. Rendering Model for the Radio Data Example (Relationship)

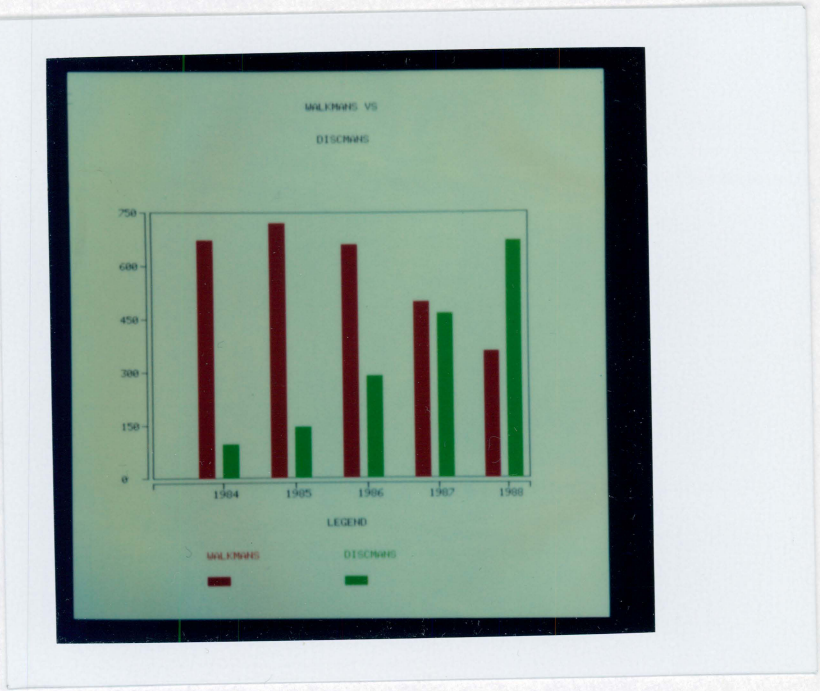

Figure 18. Rendered Chart for the Radio Data Example (Comparison)

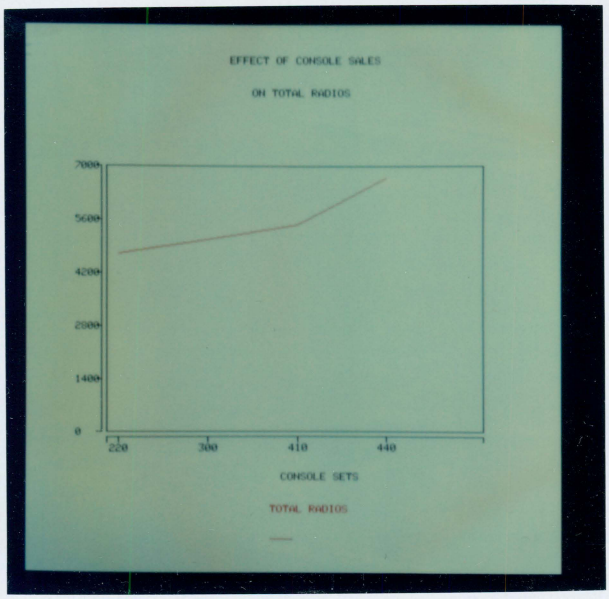

Figure 19. Rendered Chart for the Radio Data Example (Relationship)

6.4 *Conclusions*

The rendering model is the final stage in the representation pipeline in Figure 9 on page 66. The use of a pipeline of representations mirrors basic design methodology (Figure 6 on page 53) as well as underscores the differing nature of the basic aspects of a design consultation. The rendering model is created from both the goal model (second stage in the pipeline) and the background knowledge in charting which is applied to the goal model. While the preceding models in the pipeline focused on the less concrete and more client-oriented aspects of the consultation, the rendering model represents the specifics of the rendered chart, the final result of the consultation. A primary theme of this thesis is that an expert system's expertise does not derive solely from its collection of If/Then heuristics, but also from the underlying representational structures and consultation methodologies. Only when *all* these facets of expertise are implemented and integrated effectively can the "true consultant" be realized.

7.0 Results

Perhaps the best way to illustrate the utility of CHARTMAKER is to return to one of the first examples of SDM in this thesis:. the magazine publishing model in Figure 5 on page 33. Using this system dynamics model as the basis of a problem model, a variety of charts can be constructed from different presentation goals. To simplify the example, the entire model need not be considered, but the entire model could be entered in a consultation and the client could, through a series of key concept specifications, focus on different parts of the model. With this approach, an entire presentation could by built by CHARTMAKER, based on the client's differing areas of concern. For the following examples, a fragment (Figure 20 on page 94) of the original SDM model that considers the magazine's attractiveness to its readers will be used.

7.1 Example One: Comparison

In the first example, the client is interested in two aspects of the model. After constructing the original problem model, the client selects Editorial Quality and Ad-Ed Ratio (number of advertisement pages vs. the number of editorial pages) as his two basic concerns. CHARTMAKER,

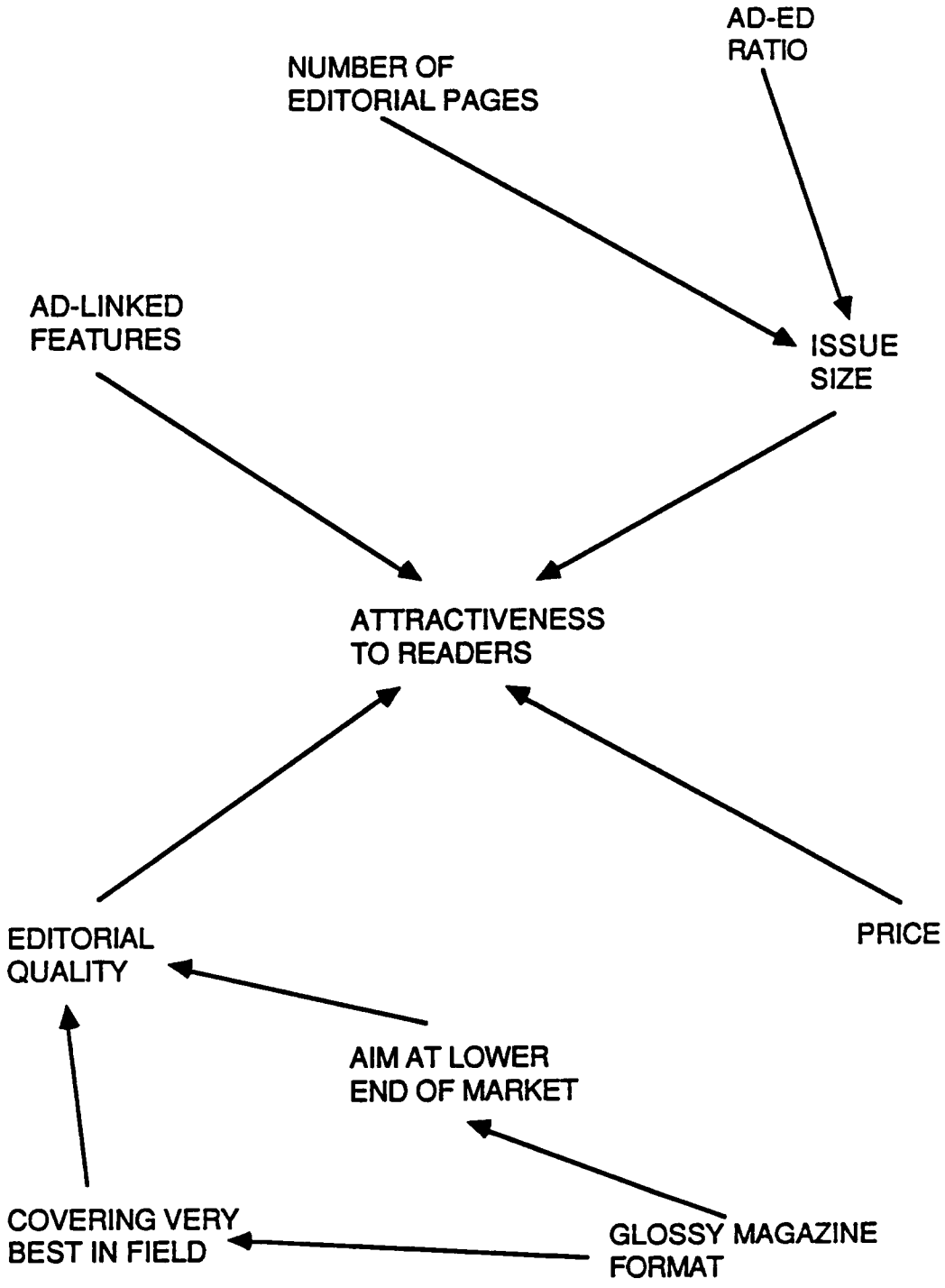

Figure 20. Problem Model for Magazine Publishing Example

then produces the goal model shown in Figure 21 on page 96. In the model, no transitive influence links are shown since no influence path exists between two key concepts. The key concepts in the model are effectively isolated from one another. The background knowledge of CHARTMAKER interprets this as a direct comparison between the two key concepts: Editorial Quality and the Ad-Ed Ratio. Moreover, CHARTMAKER, through its background knowledge recognizes that a direct comparison requires a time component. Therefore, the data pattern is presented as:

Editorial Quality	Ad-Ed Ratio	Time

The client then responds with the data (assume some imaginary publishing metric for editorial quality):

22	4	1984
31	5	1985
44	55	1986
75	61	1987

Once the data is entered, CHARTMAKER combines the raw data with the goal model to create the rendering model (Figure 22 on page 97) and the final rendered chart (Figure 23 on page 98). One problem with rendering this type of chart is scaling different quantitative measures. For example, the sales figures for two companies are both quantified in dollars so the y-axis of the chart can be labelled "dollars", but editorial quality and the ad-ed ratio are not like quantities so the y-axis cannot be uniquely labelled. There are two possible solutions when comparing unlike quantities. The quantities can be normalized to fall within a specified range that will be labelled on the y-axis and include the measure for that item in the chart's legend, or the y-axis can have multiple labels. The latter tends to be confusing, especially when more than two items are being compared. Normalizing the quantities produces a more readable graph, and it is the approach used by CHARTMAKER.

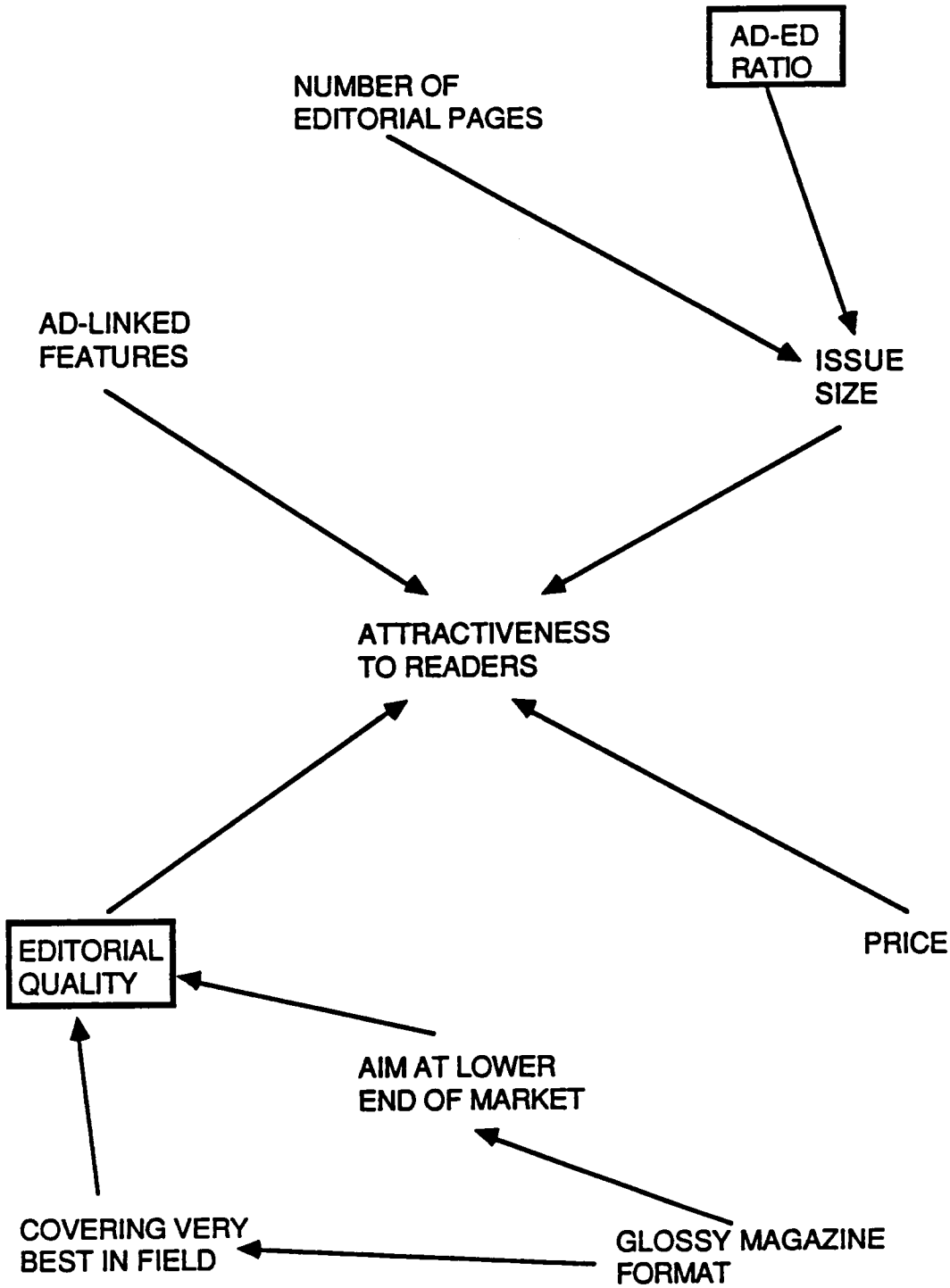

Figure 21. Goal Model for Magazine Publishing Example (Comparison)

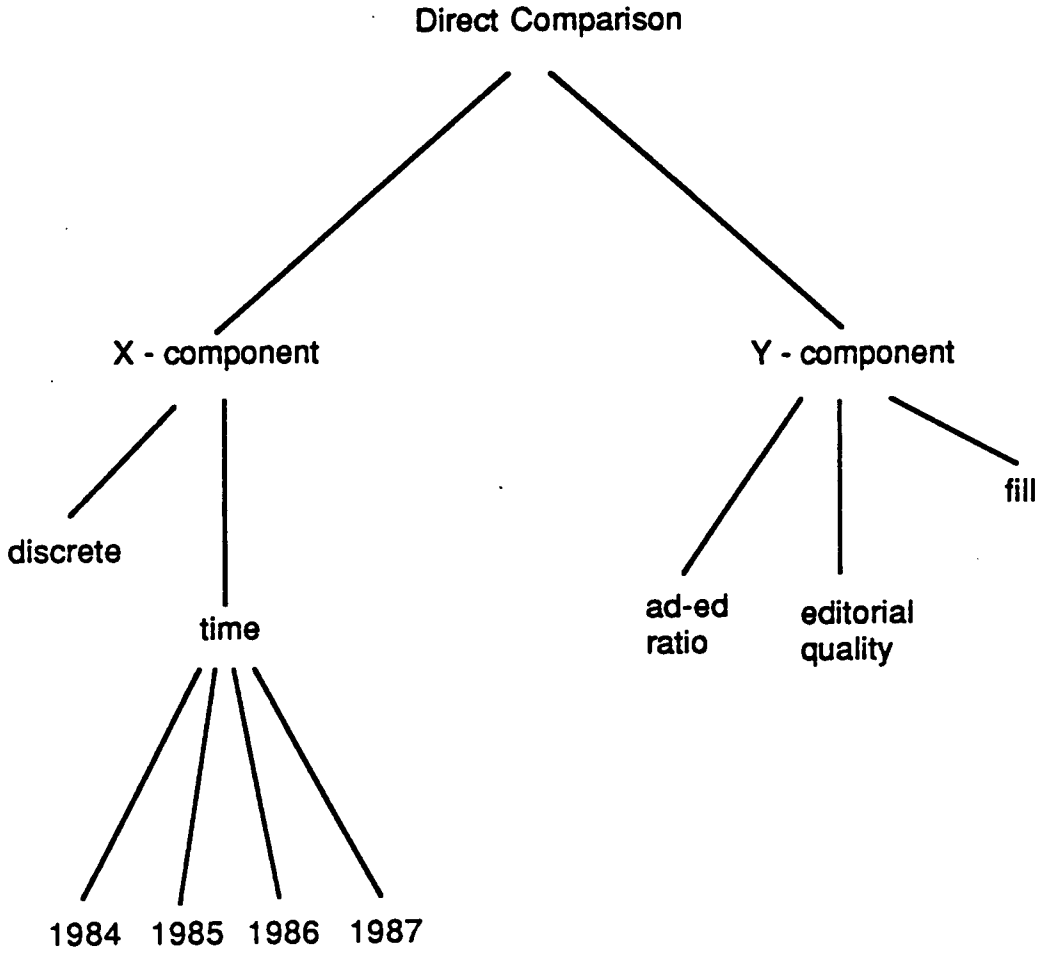

Figure 22. Rendering Model for Magazine Publishing Example (Comparison)

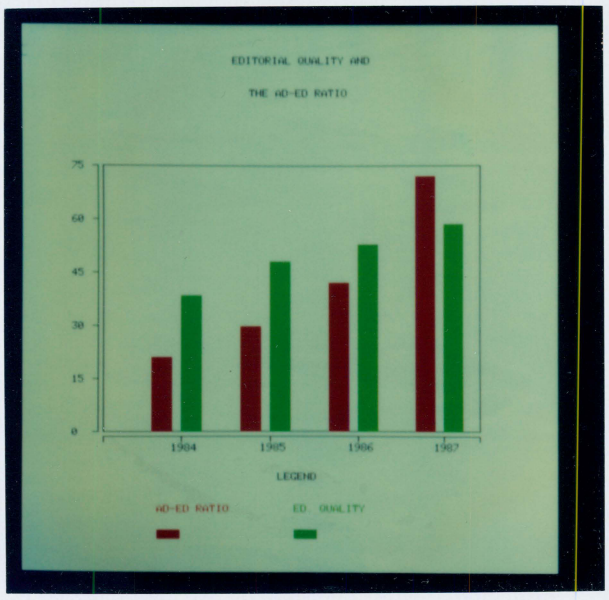

Figure 23. Rendered Chart for Magazine Publishing Example (Comparison)

7.2 *Example Two: Trend*

The second example represents the most common type of chart. Using the original problem model, the client selects only Ad-Linked Features as a key concept. CHARTMAKER recognizes this as a trend situation since there is only one concept to display in the chart, and creates the trend goal model in Figure 24 on page 100. No transitive influence links are explored since there are no other key concepts to influence. Again, CHARTMAKER recognizes he presence of time in this charting situation and puts that in the data pattern as well:

Ad-Linked Features	Time

to which the client responds with his data:

55	1984
53	1985
42	1986
35	1987
29	1988
25	1989

The data is combined with the goal model to create the rendering model in Figure 25 on page 101 and the rendered chart in Figure 26 on page 102.

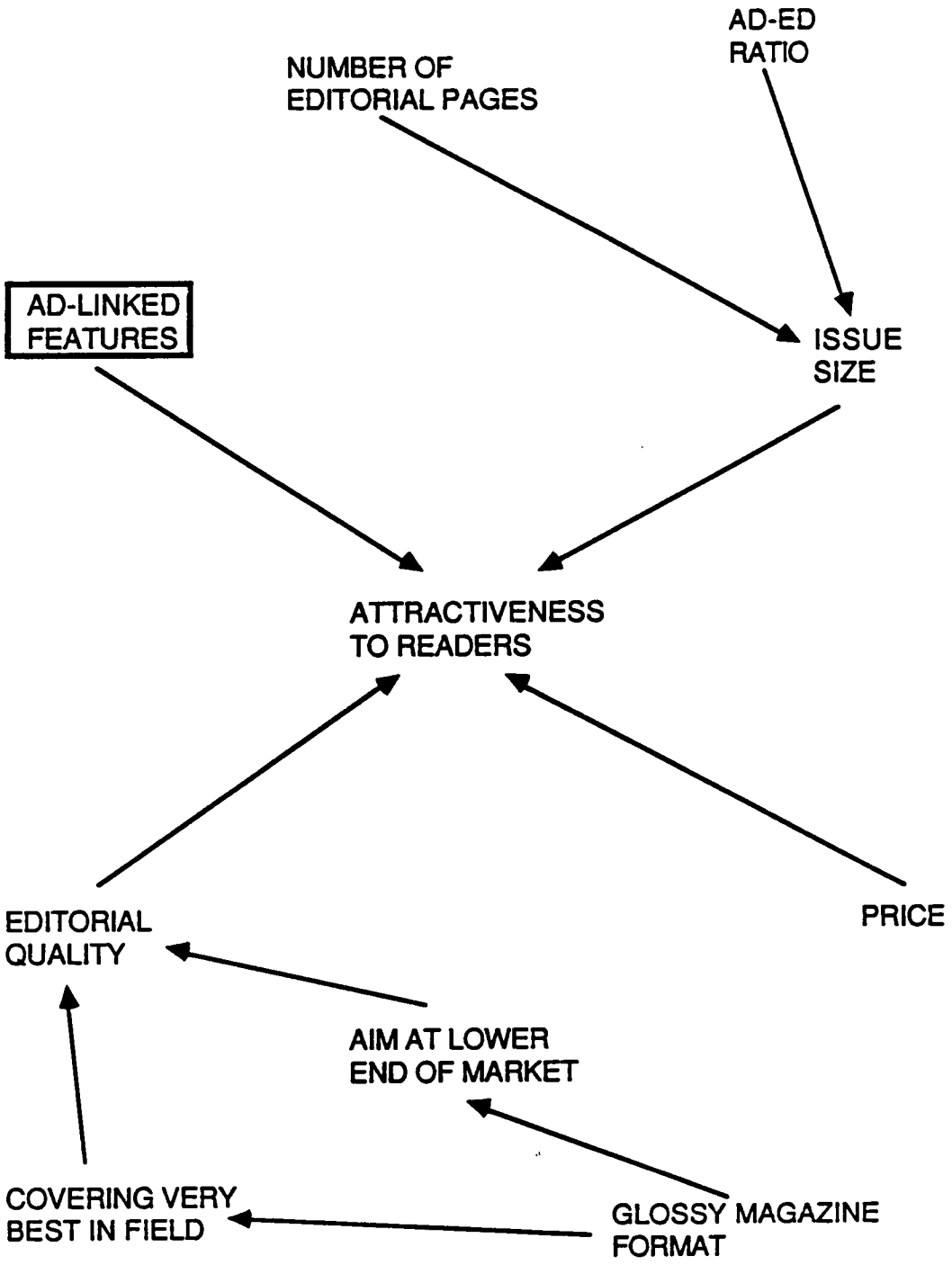

Figure 24. Goal Model for Magazine Publishing Example (Trend)

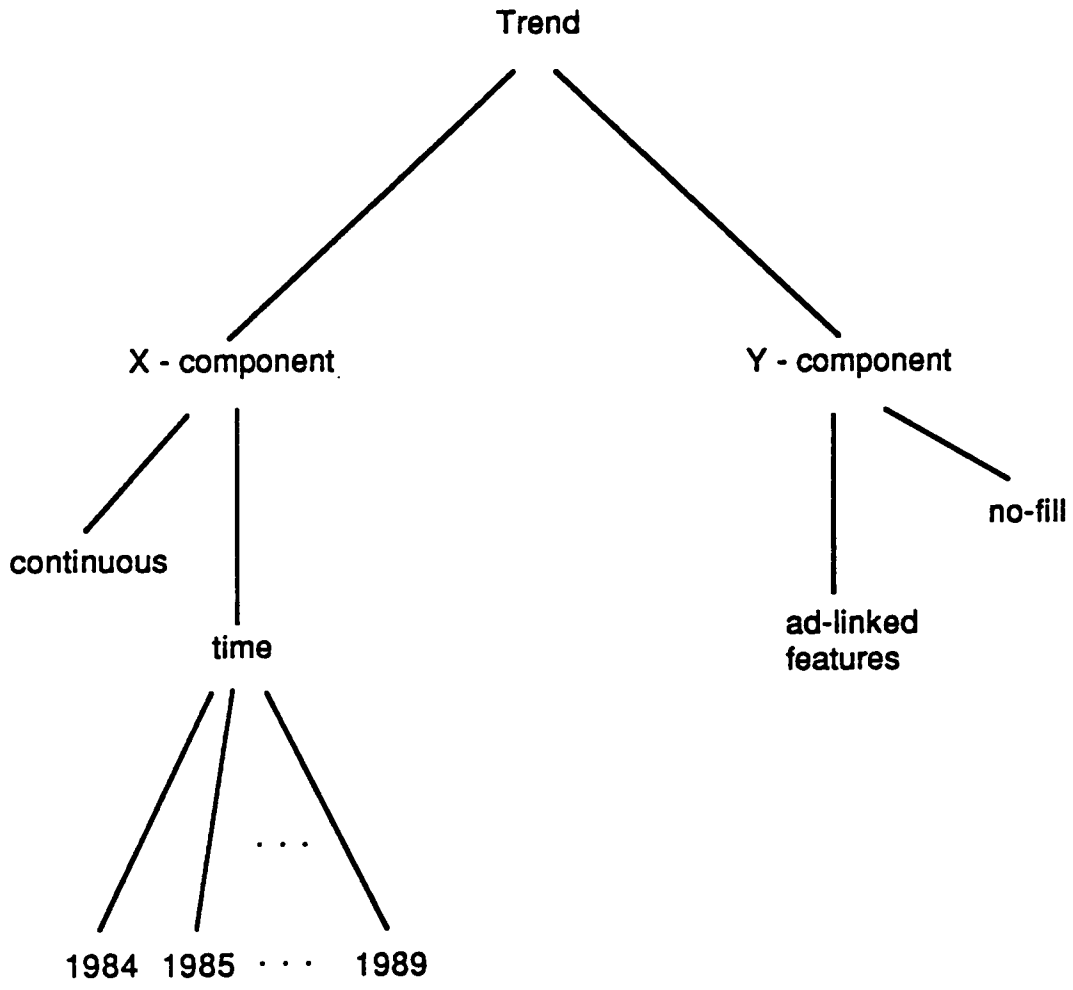

Figure 25. Rendering Model for Magazine Publishing Example (Trend)

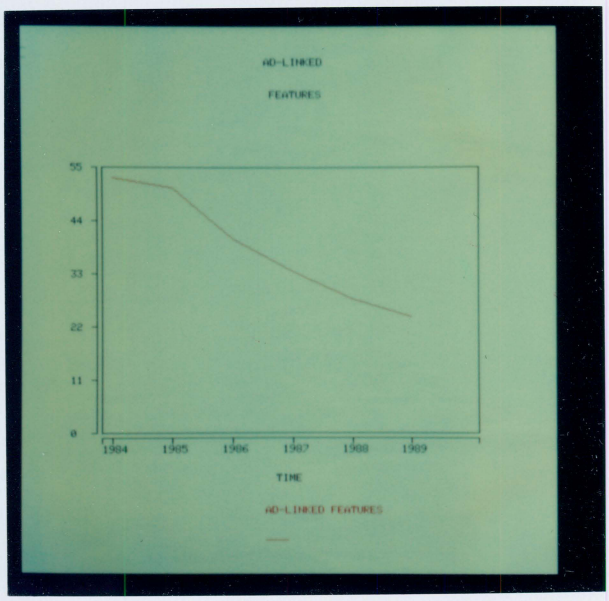

Figure 26. Rendered Chart for Magazine Publishing Example (Trend)

7.3 *Example Three: Relationship*

A chart that shows a relationship between two or more variables is really showing the causal effect that one entity (the independent variable) has on the other entities (the dependent variables). For CHARTMAKER to display a relationship of variables chart, the causal influence must exist in the problem model. In the first example in this chapter, the client selected two concepts and they were rendered as a direct comparison. The reason they were not rendered as a relationship of variables is that there is no causal influence between them in the problem model. If in some other problem model there is a causal link between the ad-ed ratio and the editorial quality, the derived goal model would indicate relationship of variables. For this example, the client selects Number of Editorial Pages and Attractiveness to Readers as his key areas of interest. A causal link does exist between the two concepts via a transitive reduction on Issue Size (Figure 27 on page 105). Since Number of Editorial Pages is on the tail of the influence arrow, it is the independent variable. Since CHARTMAKER knows the independent quantity is labelled on the x-axis, it also knows that Time is not a part of this charting situation, and the resultant data pattern:

number of editorial pages attractiveness to readers

and the data input by the user:

5	2
10	28
15	35
20	4
25	43
30	45
35	46

produces the rendering model in Figure 28 on page 106 and the rendered chart in Figure 29 on page 107

7.4 Example Four: Relative Comparison

In the final example, the client expands his scope of interest to include five key concepts: Ad-Linked Features, Issue Size, Editorial Quality, Price, and Attractiveness to Readers. The influence links play a key roles in this situation as the derived goal structure (Figure 30 on page 109) shows four of the concepts having a partial influence on the fifth concept, Attractiveness to Readers. Not surprisingly, this indicates a parts-of-a-whole or pie chart situation whereby the four influencing concepts are rendered in the chart, and the influenced concept is should actually appear in the title of the chart. While it is possible to plot a sequence of pie charts, CHARTMAKER currently supports only one pie chart per screen; therefore, in response to the data pattern:

Ad-linked Features	Issue Size	Price	Editorial Quality	Total

the client enters:

23	17	44	33	120

(only one line of data). The concept Total is added to the data pattern by CHARTMAKER. CHARTMAKER allows other factors (whether specified in the problem model or not) to influence the main concept, and it requires the client to specify a total influence factor which is simply the

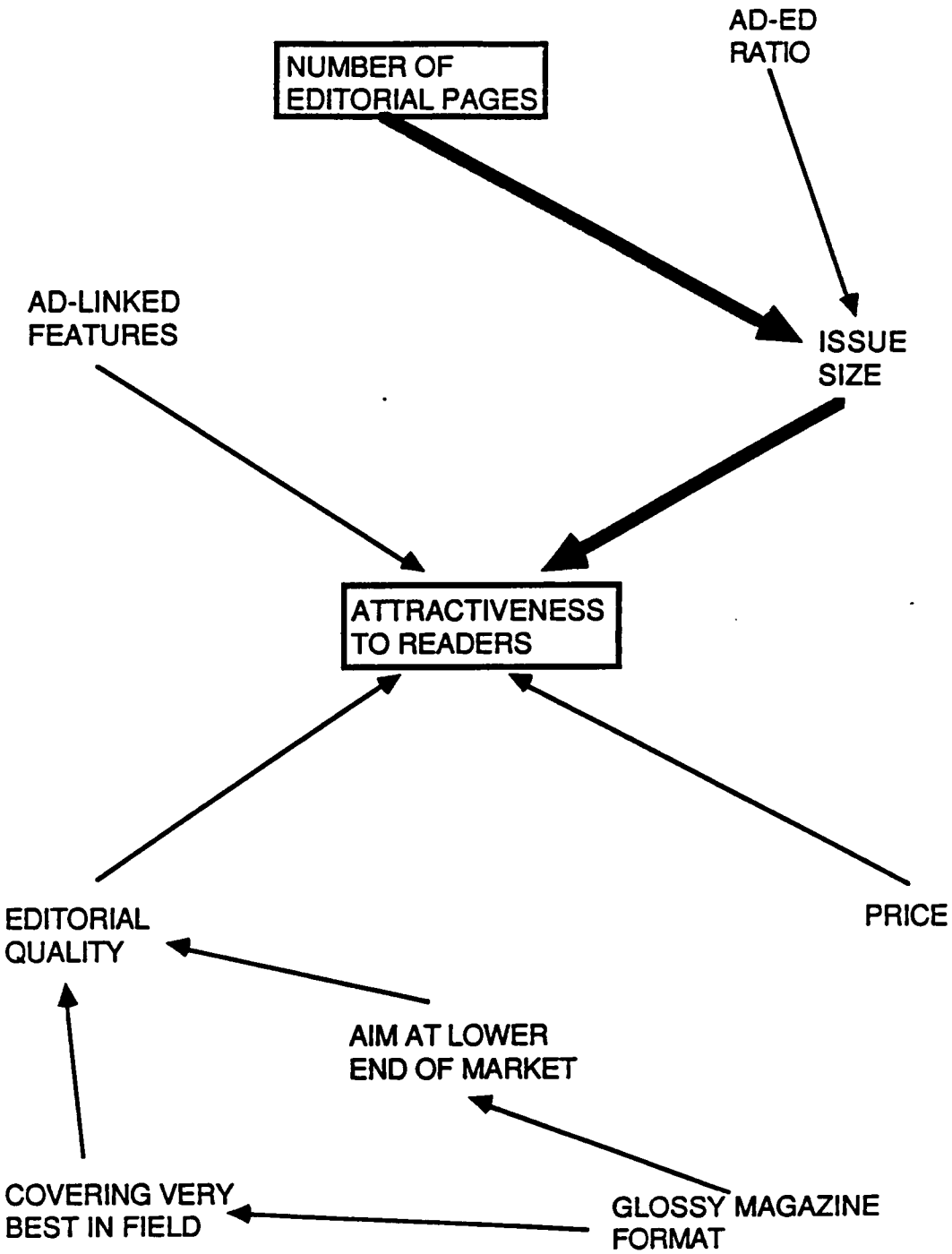

Figure 27. Goal Model for Magazine Publishing Example (Relationship)

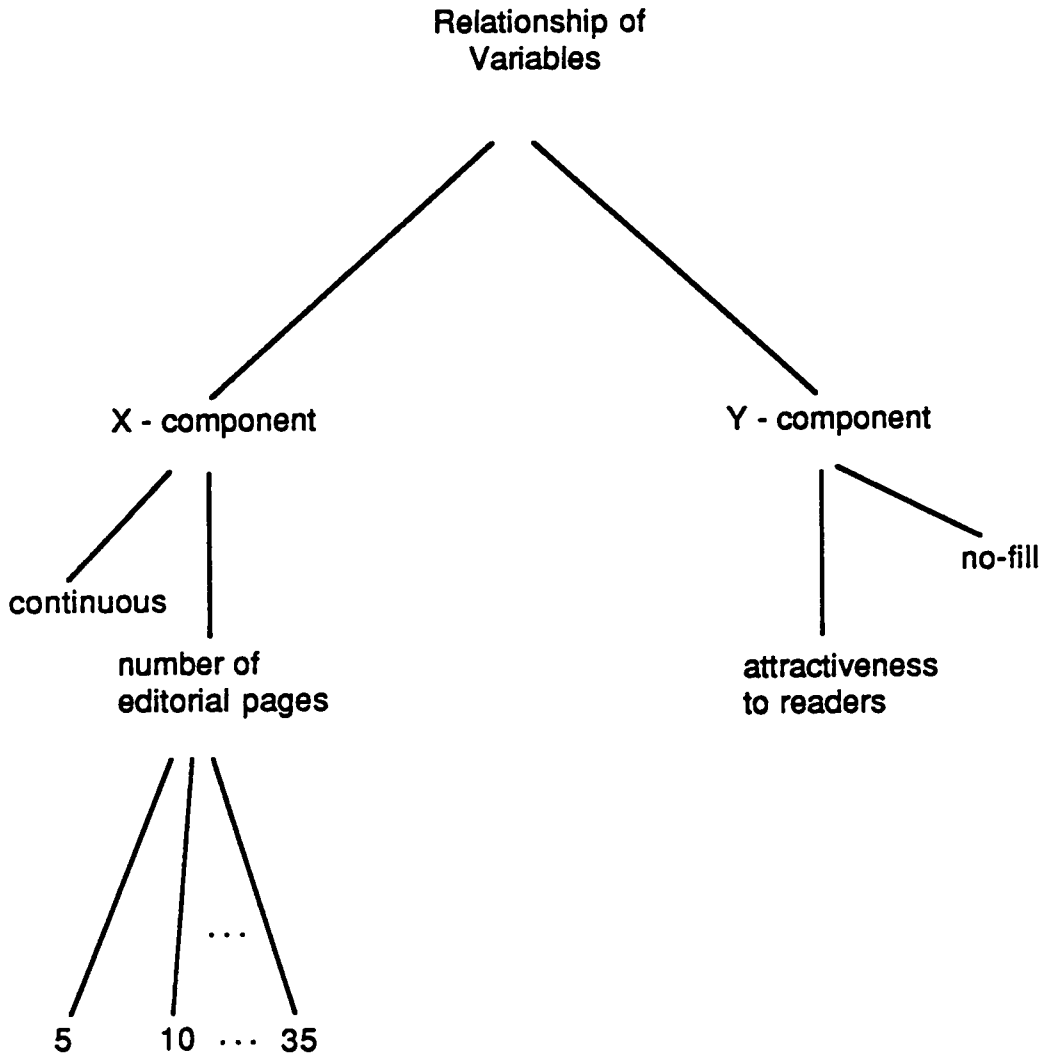

Figure 28. Rendering Model for Magazine Publishing Example (Relationship)

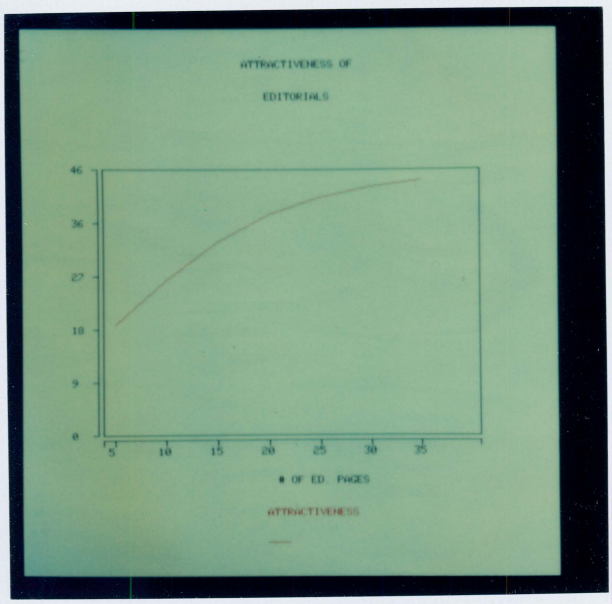

Figure 29. Rendered Chart for Magazine Publishing Example (Relationship)

sum of the influencing factors in the goal model plus any other unspecified influence. CHARTMAKER will render any difference between the sum of the specified factors and the total as "other" in the pie chart. From the data and the goal model, CHARTMAKER constructs the rendering model in Figure 30 on page 109 and the rendered pie chart in Figure 32 on page 111.

7.5 Conclusions

From the examples given in this chapter, one can see that CHARTMAKER's basic approach differs from that of conventional "question and answer" expert systems. The client builds his own model of the problem, and the validity of the charts produced from a client's model are explicitly dependent on the model. This question of validity is actually the realization of CHARTMAKER's goal: to interpret the world as the *client* sees it, and not as the knowledge engineer or programmer sees it. In more complex areas of the design domain, validity of the problem model becomes far less important, since the problem model is actually a representation of the client's preferences and personal beliefs which are too subjective to be "validated". This shift in emphasis from the knowledge engineer's viewpoint in conventional expert systems to the client's viewpoint in CHARTMAKER is the key issue in building a *true consultant.*

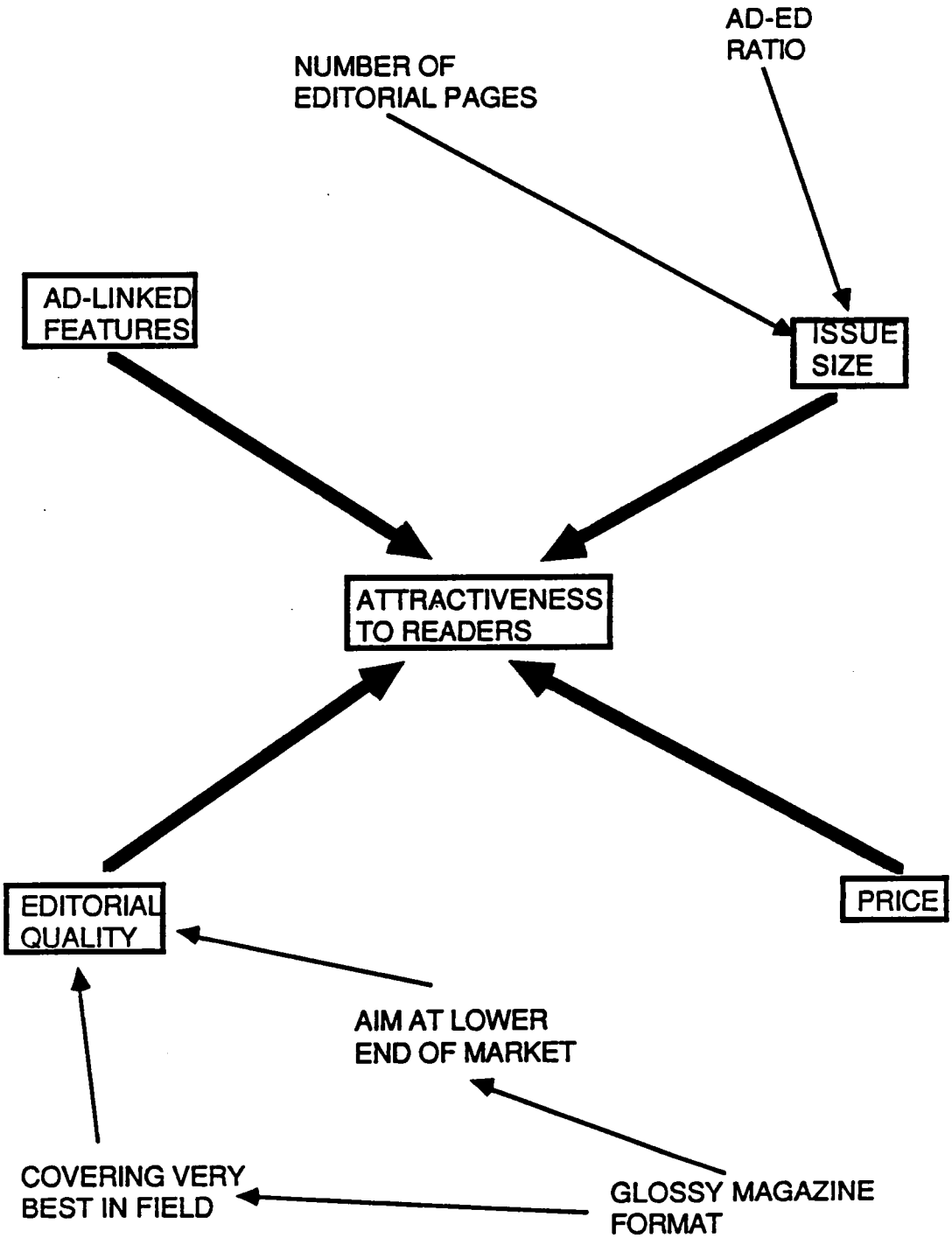

Figure 30. Goal Model for Magazine Publishing Example (Rel. Comparison)

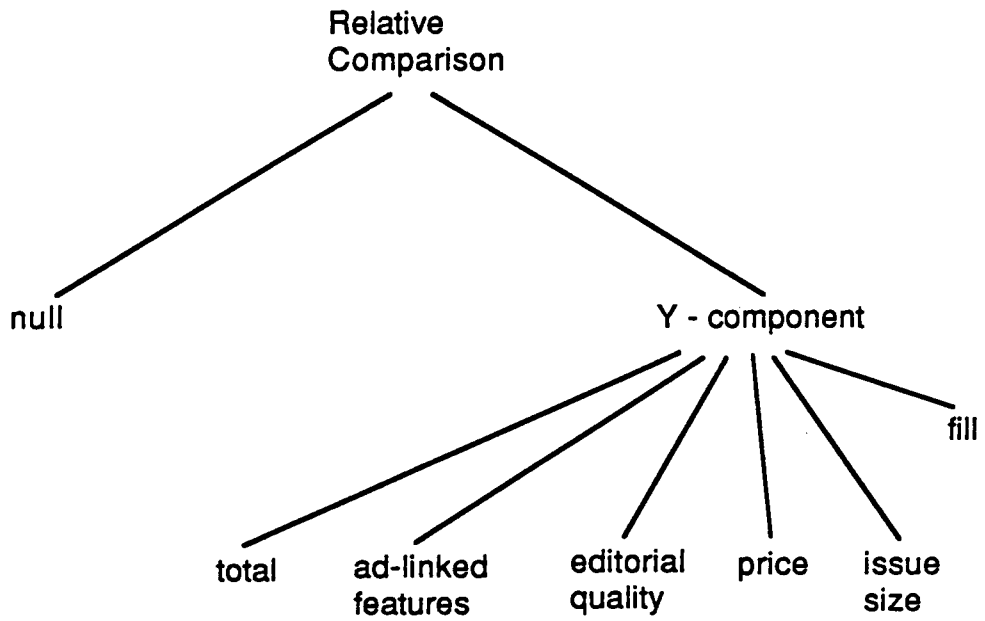

Figure 31. Rendering Model for Magazine Publishing Example (Rel. Comparison)

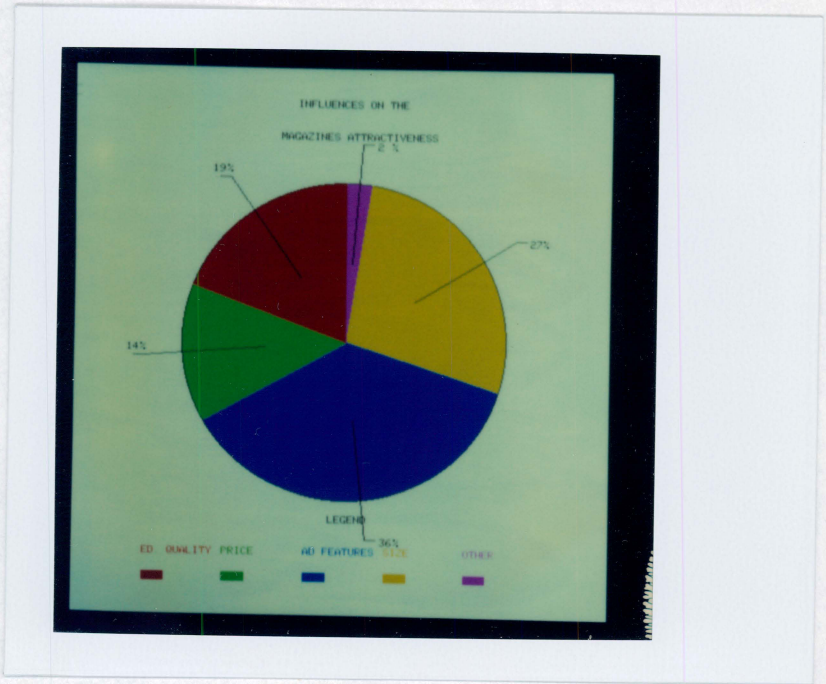

Figure 32. Rendered Chart for Magazine Publishing Example (Rel. Comparison)

8.0 Conclusion

In constructing a system dynamics model, the goal is to define a user's problem as thoroughly as possible before attempting to render a design. During the planning process, or problem definition phase, the problem is first defined by the user in terms of the necessary concepts and their associated influences. Instead of requiring the user to specify "pie" or "bar" as in Lotus-123 or Microsoft Chart, or even to select a functional description like "trend" or "comparison of variables," the system attempts to discern the appropriate chart from the nature of the data and the user's actual objective in the presentation. This is achieved by drawing inferences from the situation model. The user's implied objective may not necessarily be as simplistic as "to show a pie chart", but may be something more abstract and less declarative, such as "to show the importance of increased sales on employee bonuses." From this high-level description and the user's indication of key concepts, the automated consultant determines the actual purpose of the presentation without actually asking the user to state his goal. In essence, the user may not always be able to do this succinctly, but he can always describe his situation -- as he would to a human graphic design consultant. We were surprised to learn that the effect of much of the intentionality and subtlety of natural language can be captured and utilized without entailing the prohibitive cost of a natural language system.

8.1 *Ramifications*

The aim of this thesis is to advance the state-of-the-art in expert systems by illustrating the utility of user-oriented problem representations. Our system draws on the considerable research and field use already supporting system dynamics modelling to help create a model of the client's problem. CHARTMAKER captures the user's intent and beliefs to model the problem instead of using pre-programmed questions as do conventional expert systems. Moreover, the method of representation described here exhibits greater tolerance of improperly specified models as well as allowing for greater semantic content, since the model is built by the user with the user's own terminology. Most importantly, we have shown how generic problem models can be mapped into the background knowledge of an expert by applying background knowledge about the actual domain to build an appropriate sequence of representations.

In summary, the contributions of this work are:

- The true consultant concept
- Problem definition without predefined questions
- Defining the client's problem in his own terms
- Mapping a generic problem into background knowledge
- Indirect goal elicitation
- Separation of knowledge sources (e.g. interviewing, goal elicitation, problem classification)

- SDM structure as a static problem model
- Pipeline of representations
  - representations tailored to particular facets of the consultation
  - structural transformations to map from one representation to another

Finally, the utility of this method is not limited to the charting domain. The same principles can be employed in other domains where a "true consultant" is needed. CHARTMAKER opens the way for construction of a whole new class of expert systems.

8.2 Shortcomings in Other Domains

The CHARTMAKER system we have built uses a design theory to map from a user-described problem into a set of chart types (i.e. a classification system for charts) which forms only a small part of an expert's knowledge about charts. The freely described problem model, the goal model, the pipeline of descriptions, and the knowledge needed to map from one description to the next create a new model of expertise, the "true consultant". In this model, we have separated many different areas of an expert's knowledge that were either combined together or neglected in the rule-based paradigm of expertise. Application of this new model of expertise to other domains will require that the underlying theory be adjusted. In diagnostic systems, for example, a "theory of diagnosis" that compares the model of actual to ideal behavior will replace the "theory of design" used here. Each problem domain requires its own theoretical framework to make the true consultant idea work.

8.3 *Future Work*

As with any new approach to a problem, there are many ways to expand this representational paradigm. The strength of the technique can be increased by increasing both the complexity of the situational model and the background knowledge. For example, adding different types of causal relationships to the model such as the direct (+) and inverse (-) influences of the original system dynamics model greatly expands the representational flexibility of the situation model. The background knowledge must be enhanced to interpret more complex models. For example, + and - signs could be incorporated into CHARTMAKER to perform data and model validation by determining if the input data corresponds to the model: if the model indicates that concept A has an inverse influence on concept B, and the data shows A is increasing, then the data for B must be decreasing. As the types of causality becomes more varied, the actual concepts in the model will become more important which will probably require the background knowledge to employ a rudimentary natural language system.

One of the "design" expert systems mentioned in the literature review, Dominic, improved existing design by changing certain parameters of the existing design and then determining how that change impacted on the overall design. If the change produced an improved design, the system would incorporate that change in the new design. This type of iterative refinement is perfectly suited to system dynamics modeling. If the problem model is actually a system dynamics model, the iterative approach could be added to the true consultant and truly fulfill the last phase in the design process, refinement.

8.3.1 Alternative Domains

For the principles developed in this thesis to be useful, they must have application in other domains. To illustrate possible extensions of the true consultant approach, two problem domains are investigated: automating an airline reservation agent and designing a screen for an air traffic controller. Since these domains differ significantly from chart making, they help illustrate the potential of the discoveries in this thesis.

8.3.1.1 Airline Reservation Agent

A domain that shares some surprising similarities with CHARTMAKER is the expert airline reservation agent. The purpose of an airline reservation agent is to determine the client's flight requirements and then locate an appropriate flight (or set of flights) from a vast amount of flight information that might meet his requirements. In a general sense, the flight information is the raw data and the preferences of the traveler represent a situation. The situation model dictates what data is displayed. In most situations, the departure and arrival points of a flight are the most important issues, but other factors such as departure time, number of intermediate stops, and type of aircraft are dependent on the traveler's personal preferences. Once these preferences are specified, the system can retrieve the flight data that best matches the traveler's needs and display them on the screen by analyzing the structure of the client's situation model. Essentially, the display screen is an interface between the client and the raw data, much like a chart is an interface between the audience and the raw data.

A major difference in this domain is that the interface is dynamic. Through the consultation process, it responds to the client's changing demands. For a dynamic interface to work effectively, it must have some understanding of how it relates to the consultation process. For example, the reservation expert could interface with a travel agent or directly with the traveler. The target user

has a significant impact on the design of the system since the system that interfaces directly with the traveler needs to incorporate the knowledge of the travel agent as well. To clarify, in the indirect case the travel agent and the expert system act as an interface between the traveler and the flight data (Figure 33 on page 118) while in the direct case, the system is the only interface between the traveler and the data (Figure 34 on page 119).

Beyond the consultation model, the system needs an interaction model that specifies the information flow through the system. CHARTMAKER's interaction model can be seen as the top portion of Figure 6 on page 53, and as illustrated by the figure, the interaction model derives from the underlying domain theory. The flow of information originates with the client and his situation model that, through interaction with the system, leads to a preference (goal) model which in turn leads to a set of recommendations, the result of the consultation. For CHARTMAKER, the interaction model was built-in to the supporting code. While CHARTMAKER's interaction model supports feedback on a coarse level, the airline reservation agent requires a finer level of feedback. The reservation process where a trip is constructed by segments, and previous segments may need to be revised before a final itinerary is created. On the interaction level, information flows from the different portions of the screen. A key flow is from the traveler's preference model to the displayed flight information (from the flight data base). The preference model determines what data is to be displayed, while the displayed flight information can lead to a revised preference model. Understanding flow of data between the components of the system is another important facet of the expert's knowledge, and it will vary for different domains.

8.3.1.2 Carrier Air Traffic Control

Designing screens for air traffic controllers is another potential application of the true consultant principles. Specifically, controllers on aircraft carriers need to view a wide variety of information about the incoming aircraft to make intelligent decisions about clearances for takeoffs and landings. There are three phases of air traffic control for carrier landings: the marshalling phase, the approach

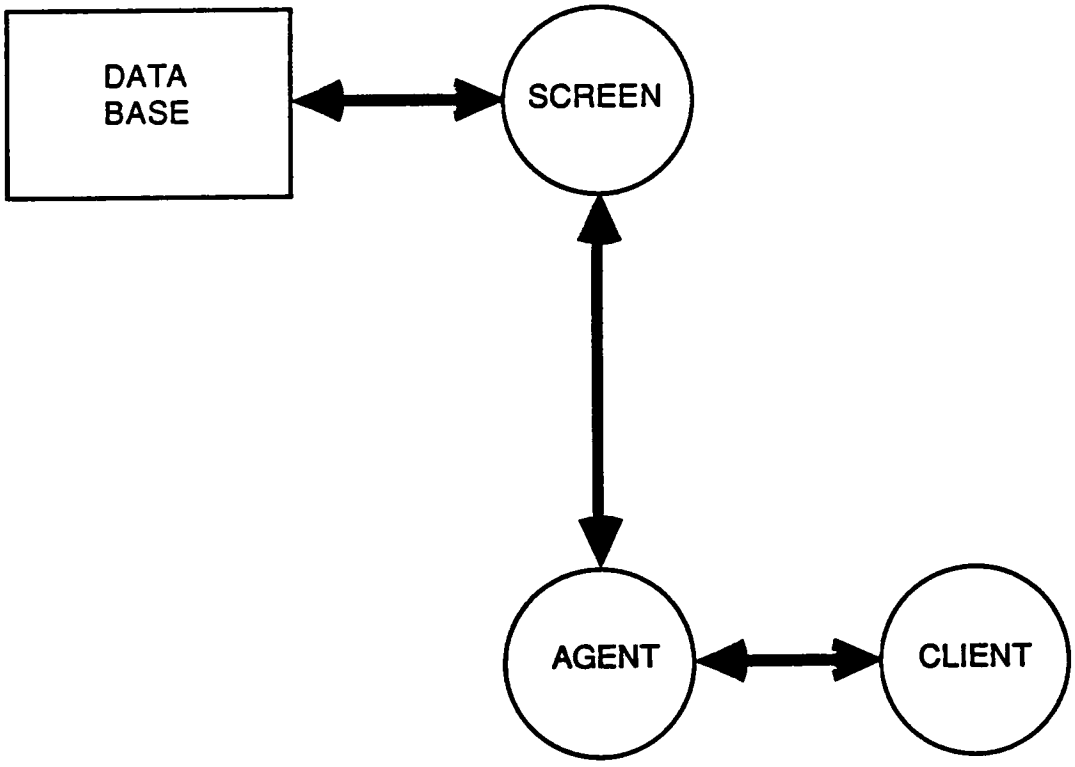

Figure 33. The Indirect Consultation Model

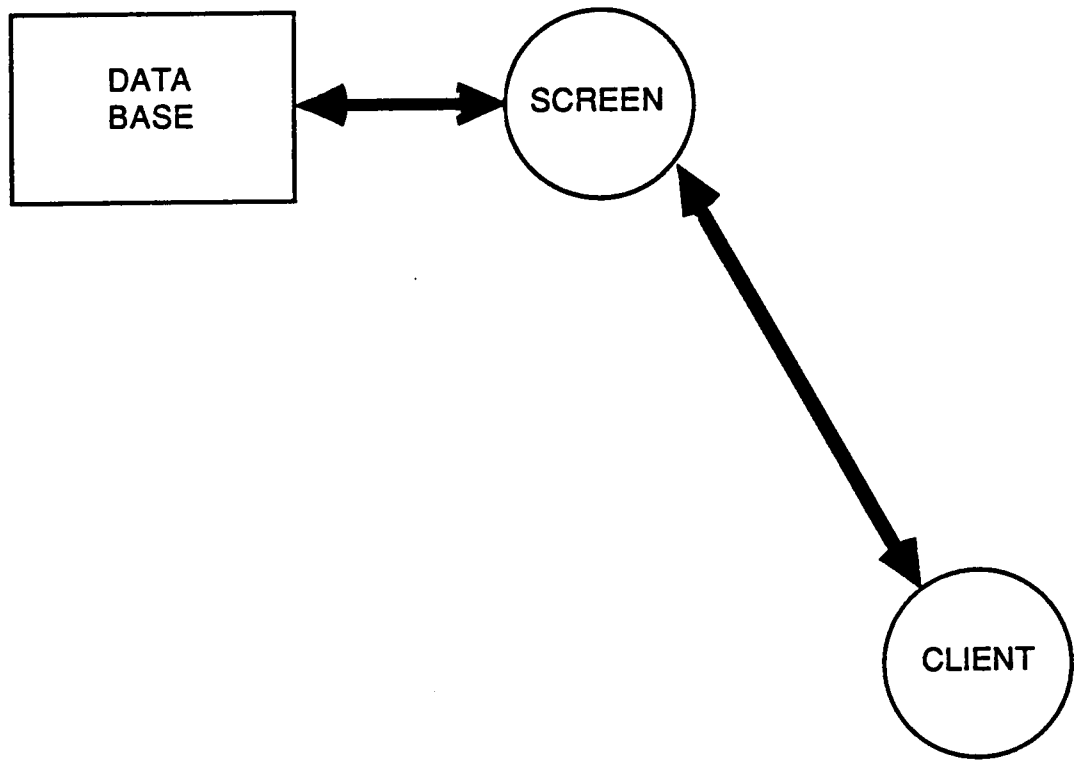

Figure 34. The Direct Consultation Model

phase, and the final phase. In the marshalling phase planes are stacked at intervals of 1000 feet beginning 50 miles from the carrier and circle in their assigned position until they are cleared to land. The controller must know details about the incoming aircraft such as amount of fuel and physical condition of the aircraft and pilot to determine their landing priority. In the approach phase the controller guides the altitude and position of the aircraft from the exit of the marshalling phase to the entry of the final phase. The final phase encompasses the last 15 miles of the flight. The final phase begins at 1200 feet and terminates on the deck of the carrier. Currently, there is an individual controller for each phase of the landing, and all three controllers use the same display which is based on World War II technology.

There are five basic components to the controller's display: plan view, stack, glide path, marshall status board, and message area. Using the approach designed for CHARTMAKER, each of these components can be regarded as concepts in the situation model. An extension to the CHARTMAKER approach could allow the addition of attribute links to the concepts since they are more complicated in this example and often require greater specification than those in the charting domain. Another modification to CHARTMAKER methodology is in key concept elicitation. The primary issue in rendering a screen of this kind is the priority of the screen components. The most important, or highest priority component should occupy the dominant position on the display screen. CHARTMAKER currently has no way to specify priority among key concepts, so a useful extension views the key concept elicitation process as a prioritization process as well. The order in which the key concepts are specified indicates the priority the client assigns to the concepts. The implementation of priorities in the system alters the interaction model. The interaction model needs to specify that hierarchical preferences in the situation model translate into specific placement objectives on the rendered screen. The components (key concepts) of the display can then be rendered appropriately. Using this method, the three individual controllers who are responsible for different phases of the landing can design their screens based on what they feel is important to their particular phase of the landing. This information is an implicit part of the con-

sultation model, since the type of client and his objectives in the consultation can vary -- as does the client of the true consultant.

In CHARTMAKER, the consultation model and the interaction model are built-in to the supporting code, but as the domains become more dynamic and more complex, the consultation model should be made an explicit, declarative, and thus more easily modified, expression. For the true consultant approach to work as a general approach to expert system design, both the consultation and interaction models need to be expressed as declarative representations. This affords far greater flexibility and is vital to the extension of the true consultant concept so that more domains can be explored (as above), without building them from the ground up. Instead, the declarative knowledge is integrated with a generic true consultant shell, thus aiding rapid expansion of the true consultant concept into other domains.

BIBLIOGRAPHY

Aikins, J., "Prototypical Knowledge for Expert Systems," *Artificial Intelligence Journal*, Vol. 20, No. 20, 1983, 163-210.

Archer, L., *Systematic Methods for Designers*, London: The Design Council, 1965.

Asimow, M., *Introduction to Design*, New Jersey: Prentice-Hall Inc., 1962.

Bannister, D. and Mair, J., *The Evaluation of Personal Constructs*, London: Academic Press, 1968.

Boose, J., *Expertise Transfer for Expert System Design*, New York: Elsevier, 1986.

Brittain, S., "Understanding Natural Languages," *AI Expert*, May 1987, 31-38.

Building Expert Systems, Hayes-Roth, F., Waterman, D., and Lenat, D., editors, London: Adisson-Wesley Publishing Company, 1983.

Caudill, M., "Neural Networks Primer," *AI Expert*, December 1987, 46-52.

Chandrasekaran, B., "Generic Tasks in Knowledge-Based Reasoning: High-Level Building Blocks for Expert System Design," *IEEE Expert*, Vol 1, No. 3, 1986.

Charniak, E., McDermott, D., *Introduction to Artificial Intelligence*, Massachusetts: Addison Wesley Publishing Company, 1985.

Cox, C. and Caldeway, D., "Software Reviews," *Computer Language*, California: Miller Freeman Communications, October 1987, 93-121.

Coyle, R., *Management System Dynamics*, New York: Wiley, 1977.

Cross, N., *The Automated Architect*, London: Pion Limited, 1977.

Eden, C., Jones S., and Sims, D., *Messing About in Problems: An Informal Approach to Their Identification and Management*, New York: Pergamon Press, 1983.

Enrick, F., *Effective Graphic Communication*, New Jersey: Auerbach, 1972.

Gregory, S., "The Development of an Automatic Adsorption Drier", *The Chemical Engineer*, December 1964, 293-302.

Gregory, S., Ed., *The Design Method*, London: Butterworth, 1966.

Howe, A., Cohen, P. and Dixon, J., "Dominic: A Domain Independent Program for Mechanical Engineering Design," *Applications of Artificial Intelligence in Engineering Problems*, Vol. 1, Berlin: Springer-Verlag, 1986, 289-300.

Jones, J., "A Method of Systematic Design," *Conference on Design Methods*, Jones, J. and Thornley, D., eds., Oxford: Pergamon Press, 1963, 53-73.

Kelly, G., "A Brief Introduction to Personal Construct Theory," *Perspectives in Personal Construct Theory*, Bannister, D., Ed., London: Academic Press, 1966.

Krippendorff, K., *Content Analysis: An Introduction to Its Method*, California: Sage Publications, 1980.

Levesque, H. and Brachman, R., "A Fundamental Tradeoff in Knowledge Representation," *Readings in Knowledge Representation*, Brachman, R. and Levesque, H., eds., California: Morgan Kaufmann Publishers, 1985, 41-67.

Levin, P., *Decision Making in Urban Design*, Building Research Station Note 51/66, Building Research Station, Garston, Herts, England, 1966.

Marcus, S., Stout, J. and McDermott, J., "VT: An Expert Elevator Designer," *AI Magazine*, Vol. 8, No. 4, Winter 1987, 41-57.

Marples, D., *The Decisions of Engineering Design*, London: Institute of Engineering Desginers, 1960

McCarthy, J., "First Order Theories of Individual Concepts and Propositions," *Readings in Knowledge Representation*, Brachman, R. and Levesque, H., eds., California: Morgan Kaufmann Publishers, 1985, 523-533.

McKim, R., *Thinking Visually*, California: Lifetime Learning Publications, 1980.

Meng, B.,"Get to the Point," *Macworld*, April 1988, 137-143.

Merrit, B., "Anatomy of a Diagnostic System," *AI Expert*, September 1987, 52-62.

Nilsson, N., *Principles of Artificial Intelligence*, California: Tioga Publishing Company, 1980.

Page, J., "A review of the papers presented at the conference," *Conference on Design Methods*, Jones, J. and Thornley, D., eds., Oxford: Pergamon Press, 1963, 205-215.

Puccia, C., Levins, R., *Qualitative Modeling of Complex Systems*, Massachussets: Harvard University Press, 1985.

Reed, S., "The Graphics Challenge," Personal Computing, New Jersey: Hayden Publishing Co., January 1988, 155-167.

Richardson G. and Pugh III, A., *Introduction to System Dynamics with DYNAMO*, Cambridge, Massachusetts: MIT Press, 1981.

Robertshaw, J., Mecca, S., and Rerick, M., *Problem Solving: A Systems Approach*, New York: Petrocelli Books Inc., 1978.

Samad, T., *A Natural Language Interface for Computer-Aided Design*, Massachusetts: Kluwer Academic Publishers, 1986.

Schmid, C., *Handbook of Graphic Presentation*, New York: John Wiley and Sons, 1954.

Shaw, M., *On Becoming a Personal Scientist*, London: Academic Press, 1980.

Smith, B., "Reflections and Semantics in a Procedural Language," Ph.D. thesis and Tech. Report MIT/LCS/TR-272, M.I.T., Cambridge MA, 1982.

Sowa, J., *Conceptual Structures: Information Processing in Mind and Machine*, Massachusetts: Addison-Wesley Publishing Company, 1984.

Spear, M., *Practical Charting Techniques*, New York: McGraw-Hill, 1969.

Tanimoto, S., *The Elements of Artificial Intelligence*, Maryland: Computer Science Press, 1987.

Tufte, E., *Visual Display of Quantitative Information*, Chesire, Connecticut: Graphics Press, 1983.

Wade, J., *Architecture, Problems, and Purposes*, New York: John Wiley and Sons, 1977.

Waterman, F., *A Guide to Expert Systems*, Massachusetts: Addison-Wesley Publishing Company, 1985.

Winograd, T., "Natural Language: The Continuing Challenge," AI Expert, May 1987, 7-8.

Appendix A. User Manual for CHARTMAKER

CHARTMAKER is relatively simple to operate since it is primarily menu driven. Before running CHARTMAKER, however, you must gain access to the CHARTMAKER and the HC PROLOG disks. The CHARTMAKER system resides on the VAX3/5 system under dua4:[shulokta] and the PROLOG system is under dua4:[roachjw]. To invoke the system (after establishing the disk links] CHARTMAKER (the name of an exec file in [shulokta]). Once the system loads, the user may begin a consultation. A consultation with CHARTMAKER can be delineated into four phases: model construction, goal extraction, data integration, and chart revision.

A.1 Model Construction

In the first phase of the consultation, the user is presented with a menu that allows him to construct the problem model:

- ADD a concept to the model
- DROP a concept from the model

- LINK two concepts (causal)
- SNIP a link
- DISPLAY current problem model
- FINISH building the model

The ADD option allows the user to add concepts to the model. He can add as many concepts as he wants, and terminates this segment by entering a null line. DROP is the reverse of add. DROP removes a concept from the model. This option is useful if the client makes a mistake in building the model or changes his mind. LINK associates a causal link with two concepts. When the user selects this option, CHARTMAKER prompts him for the "tail" concept (the one that does the influencing) first and then for the "head" concept (the one that is influenced). Again, the user can enter as many pairs of concepts as he likes and terminate this segment with a null response. SNIP deletes a link between two concepts in a manner analogous to DROP. At any time the user can display the model by selecting DISPLAY. This displays the model by its concepts. For each concept, CHARTMAKER displays the set of concepts influenced by the given concept as well as the ones that influence the given concept. Once the user is satisfied with the problem model, he terminates this phase of the consultation by selecting FINISH, and CHARTMAKER automatically proceeds to the next phase: goal extraction.

A.2 Goal Extraction

In the second phase, the system determines the key features of the problem model. CHARTMAKER presents all the concepts that were entered in the problem definition phase and asks the user to identify a key concept. CHARTMAKER removes this concept from the concept list and presents the revised list to the client and elicits another key concept. This process repeats

until the user identifies all the key concepts. From the key concepts CHARTMAKER produces the goal model which is combined with the raw data to create the chart.

A.3 Data Integration

CHARTMAKER uses the key concepts to elicit the raw data from the user. On one line, CHARTMAKER displays the key concepts as well as any required supporting concepts such as "time" or "total". The user then enters the data, line by line, matching the data format shown by CHARTMAKER. The user completes data entry by entering a null line. At this point, CHARTMKAER has all the necessary information to render a chart.

A.4 Chart Revision

The client, however, may not be satisfied with the rendered chart. Therefore, after the chart is displayed, the user is presented with another menu that allows him to revise the chart:

1. Another consultation with the old model
2. Another consultation with a new model
3. Plot another chart type from this problem model
4. Quit

The first option returns the user to the key concept selection phase 1 and allows him to alter the original problem model and proceed with another consultation. The second option takes the user

back to phase 1 to construct a new problem model (the old model is deleted). The third option permits the user to explicitly specify a particular type of chart. Before rendering the chart, CHARTMAKER determines if it has all the necessary data to plot that type of chart. If it doesn't, it will prompt the user for additional information. The fourth option, QUIT, terminates the consultation.

Appendix B. Code Listings for CHARTMAKER

```
; ***** f.hc *****
; This module builds the problem model, elicits key concepts, and
; produces the goal model.
(assert

;*****************************************************************************
; This routine initiates the first three phases of the consultation
; 1. Build the situation model
; 2. Identify the key concepts
; 3. Construct the reduced situation model

((go) if

; Build the problem model
 (print "\f")
 (go1)
```

```
; Identify the key concepts
  (print "\f")
  (choose)

; Identify the reduced model and display it
  (print "\f")
  (display_reduced)
)

;*************************************************************************
; Handle the menu selections and route them to the proper routine.

((go1) if

  (print "Problem Model construction menu:")

  (tester *choice)

  (or  (not (*choice)) (go1))

)

;*************************************************************************
; Determine virtual links in the model by traversing the concrete links.
; A virtual link can be a concrete link,

((v_link *to *from) if
```

```
  (c_link *to *from)
)

; or a sequence of concrete links.

((v_link *to *from) if
  (c_link *step *from)
  (v_link *to *step)
)

;*************************************************************************
; Pause after displaying the situation model and wait for the user's
; response.

((display) if

  (or (display1) ((print "\n\nPress enter to continue...") (getline)))
)

;*************************************************************************
; Display the current situation model by concept and its associated
; causal links.

((display1) if

  (print "\f")

; Print the header.
```

```
  (print "The current state of the problem model:")

; Select a concept
  (model *concept)
  (print " ")
  (print " ")
  (write *concept ":")
  (print " ")

; Display the "influenced by" links
  (or ((c_link *concept *from)
       (write *concept " <--- " *from "   ") (print))

; Display the "influences" links
      ((c_link *to *concept)
       (write *concept " ---> " *to "   ") (print)))

; Go back and get another concept
  (fail)
)

;*****************************************************************************
; Display the reduced model of the situation

((display_reduced) if

; Print the header
  (print "\n")
```

```
  (print "The reduced state of the problem model:")
  (print " ")
  (print " ")

; Get a list of the key concepts
  (or (bagof *keys (key *keys) *ans) (true))

; Display the virtual links for the key concepts
  (display_k_r *ans)
)

;*****************************************************************************
; This routine displays the key relationships for a set of attributes.

((display_k_r nil))
((display_k_r (nil)))

((display_k_r (*nextkey . *morekeys)) if

; Get the set of concepts that a given concept may be linked to.
  (or (bagof *b ((key *b) (not (= = *nextkey *b))) *ans) (true))

; Print the name of the current concept.
  (write *nextkey ":")
  (print " ")

; Display all the key concepts that it relates to.
  (display_r1 *nextkey *ans)
```

```
  (print "\n")
  (print "\n")

; Repeat for the rest of the concepts.
  (display_k_r *morekeys)

)

;*****************************************************************************
; This function displays all the derived links for a concept.

((display_r1 *concept nil))
((display_r1 *concept (nil)))

((display_r1 *concept  (*tofrom . *rest)) if

; Display the derived "influenced by" links
  (or ((v_link *concept *tofrom)
      (write *concept " <--- " *tofrom "  ")

; Add the derived link to the knowledge base
      (assert ((k_link *concept *tofrom)))
      (print))
     (true))

; Display the "influences" links
  (or ((v_link *tofrom *concept)
      (write *concept " ---> " *tofrom "  ")
```

```
; Add the derived link to the knowledge base
      (assert ((k_link *tofrom *concept)))
      (print))
     (true))

; Explore the remaining possible links
  (display_rl *concept *rest)

)

;*****************************************************************************
; Write out a list of elements, one at a time.

; Output a blank when finished.
((writeach nil) if
  (print "\n")
)

((writeach (*x . *y)) if
  (write *x)

; Seperate the elements by a comma
  (or (null *y) (write ", "))
  (writeach *y)
)
```

```
;*************************************************************************
; Query the user for the key concepts

((choose) if

  (print "\n")
  (or ((bagof *m ((and (model *m) (not (key *m)))) *ans)
      (write "Available concepts: ")
      (writeach *ans)

      (print "Please identify a key concept (enter null to finish):")
      (:= *k (getline))

      (or (= = *k "")
         ((or (not (model *k)) (assert ((key *k))))
         (choose))))

     ((print "No more available concepts.") (true))
     )
)

;*************************************************************************
; Link two concepts together

((link) if
  (link1)
)
```

```
((link1) if

  (print "\n")
  (print "Tail concept?")

; Get the tail concept of the relationship
  (:= *tail (getline))

; If there is a tail concept, get the head concept.
  (or (= = "" *tail)
      ((print "Head concept?") (:= *head (getline)))
  )

; Add the link to the knowledge base, get next link.
  (or (= = "" *tail)
      ((assert ((c_link *head *tail))) (link1))
  )

)

;*****************************************************************************
; Add a concept to the situation model

((add) if
  (add1)
)

((add1) if
```

```
   (print "Name of new concept (null to terminate)?")
   (:= *m (getline))
   (or (= = "" *m) ((assert ((model *m))) (add1)))

)

;*************************************************************************
; Delete a concept from the current problem model

((drop) if

; Get the name of the concept to delete
   (print "Enter name of concept to delete ")
   (:= *dd (getline))

; Delete the concept
   (retract ((model *dd)))

; Delete the "influenced by" links for this concept
   (or ((bagof *ee (c_link *dd *ee) *ans)
        (delete_head *dd *ans))
       (true))

; Delete the "influences" links for this concept
   (or ((bagof *e (c_link *e *dd) *ans1)
        (delete_tail *dd *ans1))
```

```
      (true))
)

;*************************************************************************
; Delete all the "influenced by" links for the given concept

((delete_head *x (nil)))

((delete_head *x (*y . *z)) if

  (retract ((c_link *x *y)))
  (delete_head *x *z)
)

;*************************************************************************
; Delete all the "influences" links for the given concept

((delete_tail *x (nil)))

((delete_tail *x (*y . *z)) if

  (retract ((c_link *y *x)))
  (delete_tail *x *z)

)

;*************************************************************************
; Delete a link from the current model
```

```
((snip) if

; Get the tail concept of the link to be deleted
  (print "Tail concept?")
  (:= *b (getline))

; Get the head concept of the link to be deleted
  (print "Head concept?")
  (:= *a (getline))

;  Delete the link
  (retract ((c_link *a *b)))

)

;*************************************************************************

((store) if
   (write "Enter filename to store representation = = > ")
   (:= *file (getline))
   (print "\n")
   (save "*file")
)

;*************************************************************************

((restore) if
```

```
  (or
    ((write "Enter filename to restore representation = = > ")
     (:= *file (getline))
     (print "\n")
;    (load "*file")
     )

    ((print "File not found...current model still active.") (true))
  )
)

;*************************************************************************

((create) if

  (print "create routine")
)

;*************************************************************************

((tester *choice) if

   (print \n)
   (print "Please select a menu option:" \n)

   (print "ADD a concept to the model" )
   (print "DROP a concept from the model ")
   (print "LINK two concepts" )
```

```
   (print "SNIP a link" )
   (print "DISPLAY current problem model" )
   (print "STORE current problem model" )
   (print "RESTORE a previous problem model" )
   (print "CREATE a data pattern" )
   (print "FINISH" \n)

   (:= *myrep (getline))

   (or (menval *myrep *choice)
      ((print \n "Please type one of the displayed options!" )
      (tester *choice))
      )

  (cut))

;*****************************************************************************
;*****************************************************************************

; Validate the input command
 ((menval ADD add))
 ((menval LINK link))
 ((menval DROP drop))
 ((menval DISPLAY display))
 ((menval STORE store))
 ((menval RESTORE restore))
 ((menval CREATE create))
 ((menval FINISH finish))
```

```
 ((menval SNIP snip))

)
; END

(print "Finished loading basic module")
; ***** newchart.hc *****
; This module elicts the data and invokes the appropriate charting routine.

(assert

;*****************************************************************************
; Top level function that invokes a consultation with CHARTMAKER.

((consult) if

; Perform the consultation
  (chartmaker)

; Present the final menu options.
  (redo)
)

;*****************************************************************************
; Run the consultation: build the problem model, elicit the goal, and render
; the chart.
```

```
((chartmaker) if

; Delete the old graphics driver data file
      (sys "delete chart.dat.*")

; Build the problem model
      (go)

; Determine the client's goal
      (purpose *objective)

; Render the chart
      (render *objective)
)

;*****************************************************************************
; Get the necessary raw data for chart rendering

((render *objective) if

; Get the title for the chart
      (get_subject *subject_lst)

; Get the field names for the data
      (get_field_names *fld_lst *objective)
;XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
      (get_categories *fld_lst *cat_lst)
;XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
; Get the data from the user
      (get_data *cat_lst *data_lst)

; Build the data file for the graphics drivers
      (output_data *objective *subject_lst *cat_lst *data_lst)

; Invoke the appropriate graphics driver
      (run_chart_prog *objective)
)

;*****************************************************************************
; Invoke the trend chart driver

((run_chart_prog trend) if
; (sys "run linechrt")
)

;*****************************************************************************
; Invoke the line chart driver

((run_chart_prog relationship) if
; (sys "run linechrt")
)

;*****************************************************************************
; Invoke the bar chart driver
```

```
((run_chart_prog comparison) if
; (sys "run barchrt")
)

;*****************************************************************************
; Invoke the pie chart driver

((run_chart_prog comparison_tot) if
; (sys "run piechrt")
)



;*****************************************************************************
;
((write_cat_lst nil))
((write_cat_lst ((*name *cat) . *rest)) if
        (write *name \t *cat \n)
        (write_cat_lst *rest)
        (cut)
)

;*****************************************************************************
; Get the title of the chart from the client.

((get_subject (*first_line  *second_line)) if

        (print "\n")
        (print "PLEASE ENTER THE SUBJECT OF YOUR CHART AS YOU WISH IT TO")
```

```
      (print "APPEAR ON THE CHART ITSELF (LIMIT TWO LINES). \n")
      (print "ENTER NULL LINE TO TERMINATE QUERY. \n")
      (write "= = > ")

      (:= *first_line (getline))

      (or ((= = *first_line "")
          (= = *second_line "")
         )
         ((write "= = > ")
          (:= *second_line (getline))
         )
      )

      (cut)
) ; end get_subject


;*****************************************************************************
; Get the field names for the data (key concepts).

((get_field_names *fld_lst *objective) if
  (cat_set *objective)
  (bagof *ks (key *ks) *fld_lst)
)

;*****************************************************************************
; If the desired chart is a trend chart, add TIME as a data field.
```

```
((cat_set trend) if
  (assert ((key TIME)))
)

;*****************************************************************************
; If it's a relationship chart, determine the independent variable.
;
((cat_set relationship) if
 (k_link *hd *indep)
 (retract ((key *indep)))
 (assert ((key *indep)))
)

;*****************************************************************************
; If the chart is a direct comparison, add TIME as a data field.

((cat_set comparison) if
  (assert ((key TIME)))
)

;*****************************************************************************
; If the chart is a relative comparison, extract the parent concept.

((cat_set comparison_tot) if
  (headc *dad)
  (retract ((key *dad)))
```

```
  (assert ((key *dad)))
)

;*****************************************************************************
;
((get_categories *fld_lst *cat_lst) if
        (print "\n")
        (get_cat_list *fld_lst *cat_lst)
        (cut)
)

;*****************************************************************************
;
((get_cat_list nil nil))
((get_cat_list (*first_fld . *rest_flds) ((*first_fld *desig) . *rest_cats)) if
        (:= *des_string TIME)
        (valid_des *des_string *desig)
        (get_cat_list *rest_flds *rest_cats)
        (cut)
)

;*****************************************************************************
;
((pad_it *name) if
        (:= *len (strlen *name))
        (:= *pads (- 20 *len))
        (write_pads *pads)
)
```

```
;*****************************************************************************
;
((write_pads *pads) if
        (< *pads 1)
        (cut)
)
((write_pads *pads) if
        (write " ")
        (:= *npads (- *pads 1))
        (write_pads *npads)
)

;*****************************************************************************
;
((valid_des "" other))
((valid_des TIME TIME))
((valid_des "TIME" TIME))
((valid_des quantity quantity))
((valid_des "QUANTITY" quantity))
((valid_des other other))
((valid_des "OTHER" other))

;*****************************************************************************
;
((valid_des * *desig) if
        (write "\n!INVALID DESIGNATION: PLEASE TRY AGAIN!\n")
        (write "                    DESIGNATION : ")
```

```
        (:= *des_string (getline))
        (valid_des *des_string *desig)
)

;*****************************************************************************
;
((get_data *cat_lst *data_lst) if
        (print "\n")
        (print "\n")
        (print "DATA INPUT SELECTION MENU")
        (print "------------------------- \n")
        (print "Options:   1. Input data from keyboard \n")
        (print "           2. Input data from file \n")
        (query_user_for_input_choice *choice)
        (or ((= = *choice 1)
            (get_terminal_data *cat_lst *data_lst)
           )
           ((= = *choice 2)
            (get_file_data *cat_lst *data_lst)
           )
           (print "Error: bad processing in get_data rule ")
        )
        (cut)
)

;*****************************************************************************
;
((query_user_for_input_choice *choice) if
```

```
        (write "PLEASE ENTER OPTION NUMBER OF INPUT CHOICE = = > ")
        (:= *reply (read))
        (or (valid_input_choice *reply *choice)
            ((print \n "Answer not 1 or 2, please try again")
             (query_user_for_input_choice *choice)
            )
        )
        (cut)
)

;*****************************************************************************
;
((valid_input_choice 1 1))
((valid_input_choice 2 2))

;*****************************************************************************
; Read the data values from the keyboard

((get_terminal_data *cat_lst *data_lst) if
        (number_of_items *cat_lst *n_items)
        (print "\n")
        (print "PLEASE ENTER YOUR DATA BY SPECIFYING VALUES FOR EACH OF")
        (print "THE CATEGORIES IN THE FOLLOWING FORMAT, ONE SET PER LINE:\n\n")
        (write "    ")
        (write_lst *cat_lst)
        (print "\n")
        (print "ENTER A NULL LINE TO TERMINATE DATA ENTRY.\n")
        (print "ENTER DATA HERE:\n")
```

```
        (get_data_lines *n_items *data_lst)
        (cut)
)

;*****************************************************************************
;
((get_data_lines *n_items *data_list) if
        (write "= = > ")
        (:= *line (getline))

        (or ((= = *line "")
             (= = *data_list nil)
             (cut)
            )
            ((valid_line *n_items *line)
             (get_data_lines *n_items *rest)
             (= = *data_list (*line . *rest))
            )
        )
        (cut)
)

((valid_line *x *y))

;*****************************************************************************
;
((number_of_items nil 0))
((number_of_items ((* TIME) . *rest) *n_items) if
```

```
        (number_of_items *rest *num)
        (:= *n_items (+ 1 *num))
)
((number_of_items ((* quantity) . *rest) *n_items) if
        (number_of_items *rest *num)
        (:= *n_items (+ 1 *num))
)
((number_of_items ((* *) . *rest) *n_items) if
        (number_of_items *rest *n_items)
)

;*****************************************************************************
;
((write_lst nil))
((write_lst ((*head quantity) . *tail)) if
        (write *head)
        (:= *len (strlen *head))
        (:= *pads (- 15 *len))
        (write_pads *pads)
        (write_lst *tail)
        (cut)
)

((write_lst ((*head TIME) . *tail)) if
        (write *head)
        (:= *len (strlen *head))
        (:= *pads (- 15 *len))
        (write_pads *pads)
```

```
        (write_lst *tail)
        (cut)
)
((write_lst (* . *tail)) if
        (write_lst *tail)
        (cut)
)

;*****************************************************************************
; Output the charting data: data values and concept labels.

((output_data *objective *subject_lst *cat_lst *data_lst) if
        (open "chart.dat" fd1 w)
        (write fd1 *objective "\n")
        (write_subj_lst *subject_lst)
        (:= *llen (length (quote *cat_lst)))
        (write fd1 *llen "\n")
        (write_valid_cats *cat_lst)
        (write_data *data_lst)
        (close fd1)
        (cut)
)

;*****************************************************************************
;
((write_subj_lst nil))
((write_subj_lst (""))
```

```
((write_subj_lst ("" . ""))
((write_subj_lst (*line . *rest)) if
        (write fd1 "subject " *line \n)
        (write_subj_lst *rest)
)

;*****************************************************************************
;
((write_valid_cats nil))
((write_valid_cats ((*fld_name TIME) . *rest)) if
        (write fd1 "field_desc  " *fld_name "\n")
        (write_valid_cats *rest)
)
((write_valid_cats ((*fld_name quantity) . *rest)) if
        (write fd1 "field_desc  " *fld_name "\n")
        (write_valid_cats *rest)
)
((write_valid_cats ((* *) . *rest)) if
        (write_valid_cats *rest)
)

;*****************************************************************************
;
((write_data nil))
((write_data (*first . *rest)) if
        (write fd1 *first "\n")
```

```
      (write_data *rest)
)
); end assert


; ***** class.hc *****
; This module maps the reduced problem model into the background
; knowledge to determine the chart type.
(assert

;*****************************************************************************
; Determine if the problem structure indicates relationship of variables.

((purpose relationship) if

; More than one key concept?
 (bagof *keys (key *keys) *ans)
 (:= *ll (length (quote *ans)))
 (> *ll 1)

; A influences B structure?
 (key *k)
 (parent *k *ans)
)

;*****************************************************************************
; This function determines if a given node is a parent node (at the
; head of the relationship arrow), for any of a list of nodes.
```

```
; Empty list of candidate children (base case)
((parent *dad nil))
((parent *dad (nil)))

((parent *dad  (*kid . *rest)) if

; Does a link exist between the two nodes?
    (or
     (v_link *kid *dad)
     (= = *dad *kid)
    )

; Check the next node in the list
   (parent *dad *rest)
)

;*****************************************************************************
; Is the model indicative of a relative comparison?
;

((purpose comparison_tot) if

; Is there a parent concept with many children?
    (key *main)
    (bagof *kids (k_link *main *kids) *ans)
    (:= *totkids (length (quote *ans)))
    (> *totkids 1)
```

```
    (assert ((headc *main)))

)

;*****************************************************************************
; Is the model indicative of a direct comparison?
;

((purpose comparison) if

; Isolated key concepts?
    (not (k_link *a *b))
    (bagof *kys (key *kys) *ans2)
    (:= *tl (length (quote *ans2)))
    (> *tl 1)

)

;*****************************************************************************
; Is the model indicative of a trend?
;

((purpose trend) if
  (key *thiskey)
  (not (bagof *otherkey ((key *otherkey) (not (= = *thiskey *otherkey))) *ans))
)
```

```
;*****************************************************************************
)
; Pathological (under or overspecified) goal model?
;
;*****************************************************************************
((purpose unknown) if
  (print "No purpose could be derived from the model, it may be underspecified")
)

;*****************************************************************************

(print "Finished loading classification subsystem")
; *****FINAL.HC*****
; This module presents the final model and sets up the necessary
; environment for the desired option.
(assert

;*****************************************************************************
; After the chart has been displayed, display the final option menu and get
; the user's response.

((final *reply) if

; Display the menu
  (final_options)

; Get the client's response
```

```
  (get_choice *reply)
)

;*****************************************************************************
; Display the final option menu:
;       1. Return to the old problem model
;       2. Return to problem definition phase to create a new model
;       3. Plot of particular type of chart using the input data
;       4. End the consultation

((final_options) if
  (print "\n")
  (print "Final options:")
  (print "----------")
  (print "  1. Another consultation with old model")
  (print "  2. Another consultation with new model")
  (print "  3. Plot another chart type from this model")
  (print "  4. Quit")
  (print "\n")
  (print "Enter number of desired option.")
)

;*****************************************************************************
; Get the client's menu choice

((get_choice *reply) if
  (:= *reply (read))
)
```

```
;*****************************************************************************
; Take the appropriate action for the client's selection.

((redo) if
 (final *ans)
 (or

; Return to the situation definition  phase with the same situation model
; to modify or re-use the existing model.
   ((= = *ans 1)  (delete ((key . *)))
                  (delete ((k_link . *)))
                  (consult)
                  )

; Return to the situation definition phase to create a new problem model
   ((= = *ans 2)  (delete ((key . *)))
                  (delete ((k_link . *)))
                  (delete ((model . *)))
                  (delete ((c_link .*)))
                  (consult)
                  )

; Plot a particular type of chart, as chosen by the user
   ((= = *ans 3) (print "Select from the following chart types:")
                 (print "1. line chart")
                 (print "2. bar chart")
```

```
            (print "3. pie chart")
            (print "\n")
            (print "Enter number of desired chart")
            (:= *chart (read))
            (obj *chart *style)
            (render *style)
            (redo)
            )

; End the consultation
    (quit)
  )
)

;*****************************************************************************
; Set up the rendering model for a bar chart (user-selected).

((obj 2 comparison) if
  (delete ((key total)))
  (delete ((key TIME)))
)

;*****************************************************************************
; Set up the rendering model for a pie chart (user-selected).

((obj 3 comparison_tot) if
  (delete ((key TIME)))
)
```

```
;*****************************************************************************
; Set up the rendering model for a line chart (user-selected).

((obj 1 relationship) if

  (delete ((key TIME)))
  (delete ((key total)))

; Determine the quantity to be labeled n the x axis.
  (or ((not (k_link *h *t))
       (print "What is the name of the independent quantity?")
       (:= *ind (getline))
       (print "What concept is affected by the independent concept?")
       (:= *dep (getline))
       (smartASSERT key *dep)
       (smartASSERT key *ind)
       (smartASSERT k_link *dep *ind))
    (true))
)

;*****************************************************************************
; If a one-place predicate of this form exists, don't re-assert it, otherwise
; assert it.

((smartASSERT *p *c1) if
  (or (*p *c1) (ASSERT ((*p *c1))))
)
```

```
;*****************************************************************************
; If a two-place predicate of this form exists, don't re-assert it, otherwise
; assert it.

((smartASSERT *p *c1 *c2) if
  (or (*p *c1 *c2) (ASSERT ((*p *c1 *c2))))
)



)
(print "final loaded")
C*********************************************************************
C  This routine creates a piechart from the data in the file
C  CHART.DAT
C*********************************************************************
C
      INTEGER   dat(50,7), total, other
      integer*2 cstr(10)
      INTEGER   time(50), tj
      INTEGER   ITEMS,I,J,LEN,X,Y,LASTX,LASTY(7),INCX,yearS,MAXQ
      INTEGER   WIDTH,xx,yy
      INTEGER*2 ITM_NM(20)
      REAL      YSCALE

c  Set up the radius of the pie
      iradi = 150
```

```
c  Initialize the rastertech display
     CALL INITIL

c  Get the number of items in the comparison
     READ (9,*) ITEMS
     ITEMS = ITEMS

c  get the name o the first item
     READ (9,60) ITM_NM

c  render it in the legend portion of the chart
     CALL MOVABS(-15,-170)
     CALL TEXT1(6,'LEGEND')
     X = (((1 - ITEMS) * 200) / (ITEMS)) - 25
     INCX = (2 * IABS(X)) / (ITEMS)
     items = items - 1


c  get the remainder of the item names annd render them in the legend
     DO 10,I = 1,ITEMS
       CALL SETCOLOR(I)
       LEN = LENTH (ITEM_NM)
       CALL MOVABS (X,-200)
       CALL TEXT1 (15, ITM_NM)
       CALL MOVABS (X, -225)
       CALL RECREL (20,8)
       CALL MOVABS (X + 3, -222)
       CALL AREA1
```

```
      X = X + INCX
      READ (9,60) ITM_NM
10    CONTINUE

     I = 1
     MAXQ = 0

c  read in the data for the categories
20    READ (9,*,END=30) (dat(I,tj), tj = 1,items), total
      DO 25,J=1,ITEMS
       IF (dat(I,J).GT.MAXQ) THEN
        MAXQ = dat(I,J)
       ENDIF
25     CONTINUE
      I = I + 1
     GO TO 20



30    CLOSE(9)

C  KEEP TRACK OF THE TOTAL AMOUNT TO CALCULATE "OTHER"
     sum = 0.

C  SUM THE DATA
     do 77 j=1,items
      sum = sum + dat(1,j)
 77   continue
```

```
C  CALCULATE "OTHER" IF ANY
     other = total - sum

C  IF THERE IS AN "OTHER" RENDER IT IN THE LEGEND
     if (other .gt. 0) then
       CALL SETCOLOR(Items + 1)
       CALL MOVABS (X,-200)
       CALL TEXT1 (6, 'OTHER')
       CALL MOVABS (X, -225)
       CALL RECREL (20,8)
       CALL MOVABS (X + 3, -222)
       CALL AREA1
     endif

C  DETERMINE THE SPACING IN THE CHART
     yearS = I - 1
     INCX = 330 / ((yearS * ITEMS) + yearS)
     WIDTH = 2 * INCX / 3
     PRINT *,'INCX = ', INCX
     X = -INT(REAL(INCX * ITEMS + 1) * (REAL(yearS) / 2.0)) - 20
     PRINT *, 'INCX = ', INCX, '  LASTX = ', LASTX
     YSCALE = 240.0 / REAL(MAXQ)
35   CONTINUE

41   continue

C  DRAW THE OUTLINE OF THE PIE
     call movabs(0,0)
```

```
      call drwabs(0,iradi)
      call movabs(0,0)
      call setcolor(8)
      call circle(iradi)
      call movabs(0,0)
      pang = .25

C  ADD THE DATA FOR THE "OTHER" CATEGORY
      dat(1,items + 1) = other

C  RENDER THE CATEGORIES AS PIE WEDGES
      DO 45,J = 1,ITEMS + 1
       CALL SETCOLOR(J)

C  FIRST LINE OF THE WEDGE
       angr = float(dat(1,j))/float(total)
       xx = int(iradi*cos(2*3.141592*(angr + pang)))
       yy = int(iradi*sin(2*3.141592*(angr + pang)))
       call drwabs(xx,yy)

C  SECOND LINE OF THE WEDGE
       xx = int(iradi/2*cos(2*3.141592*(((angr)/2) + pang)))
       yy = int(iradi/2*sin(2*3.141592*(((angr)/2) + pang)))
       call movabs(xx,yy)

C  FILL IN THE WEDGE
       call setcolor(j)
       call area1
```

```
C  LOCATE THE PERCENTAGE FIELD
      xx = int((iradi*2.5)/2*cos(2*3.141592*
     &    (((angr)/2) + pang)))

      yy = int((iradi*2.5)/2*sin(2*3.141592*
     &    (((angr)/2) + pang)))

C  LABEL THE PERCENTAGE FOR THAT CATEGORY
      call setcolor(8)
      call drwabs(xx,yy)
      nx = isign(10,xx)
      call drwrel(nx,0)
      call movrel(nx/2-1, -1*nx/2)
      call nchr(100.*(angr),cstr,ilen)
      call val8(20)
      call text1(ilen, cstr)
      call movrel(13,0)
      cstr(1) = '%%'
      call text1(1,cstr)

C  RESET FOR THE NEXT CATEGORY
      call movabs(0,0)
      pang = pang + angr

45    CONTINUE
```

```
      CALL QUIT

      STOP

50    FORMAT (7X,I1)
60    FORMAT(12X,20A2)
110   FORMAT(8(I10))
      END
C
      SUBROUTINE SETCOLOR(I)

      INTEGER I

C  MAP INTO THE COLOR TABLE BASED ON I
      IF (I.EQ.1) THEN
        CALL VALUE (230,0,0)
      ELSE IF (I.EQ.2) THEN
        CALL VALUE (0,230,0)
      ELSE IF (I.EQ.3) THEN
        CALL VALUE (0,0,230)
      ELSE IF (I.EQ.4) THEN
        CALL VALUE (230,230,0)
      ELSE IF (I.EQ.5) THEN
        CALL VALUE (230,0,230)
      ELSE IF (I.EQ.6) THEN
        CALL VALUE (0,230,230)
      ELSE IF (I.EQ.7) THEN
        CALL VALUE (230,230,230)
```

```
      else if (i.eq.8) then
        call value(0,0,0)
      ENDIF

      RETURN
      END
C
C  INITIALIZE THE RASTERTECH SCREEN (CALLS TO RASTERTECH ROUTINES)
      SUBROUTINE INITIL

      INTEGER LEN,X
      INTEGER*2 SUBJ1(40), SUBJ2(40)

C     THROW AWAY LINE WHICH SAYS TREND
      OPEN(UNIT=9,FILE='CHART.dat',STATUS='OLD')
      READ (9,*)
      READ (9,100) SUBJ1
      READ (9,100) SUBJ2
      CALL RTINIT('DEV',3)
      CALL ENTGRA
      CALL COLD
      CALL RTINIT('DEV',3)
      CALL ENTGRA
      CALL VAL8(245)
      CALL CLEAR
      CALL VAL8(20)
c
C  LABEL THE SUBJECT OF THE CHART
```

```
      LEN = LENTH(SUBJ1)
      X = -LEN * 5
      CALL MOVABS(X,220)
      CALL TEXT1(2 * LEN,SUBJ1)
      LEN = LENTH(SUBJ2)
      X = -LEN * 5
      CALL MOVABS(X,190)
      CALL TEXT1(2 * LEN,SUBJ2)

      RETURN
100   FORMAT(8X,40A2)
      END
c


C  RETURN THE LENGTH OF THE GIVEN STRING
      INTEGER FUNCTION LENTH(STR)
C
      INTEGER*2 STR(40)
      INTEGER  I
C
      I = 40
      DOWHILE ((STR(I).EQ.' ').AND.(I.GT.0))
        I = I - 1
      END DO

      LENTH = I
```

```
      RETURN
      END
C*********************************************************************
C  This routine creates a bar chart from the input data.
C*********************************************************************
      INTEGER   DAT(50,7)
      integer*2 cstr(10)
      INTEGER   TIME(50), tj
      INTEGER   ITEMS,I,J,LEN,X,Y,LASTX,LASTY(7),INCX,yearS,MAXQ
      INTEGER   WIDTH
      INTEGER*2 ITM_NM(20)
      REAL      YSCALE
C
C  Initialize the Rastertech graphic display
      CALL INITIL
C
C  Get the number of items to be displayed (categories)
      READ (9,*) ITEMS
C
C  Discount the independent quantity (It represents the x-axis)
      ITEMS = ITEMS - 1
C
C  Read in an item name and place the name with its associated color
C  in the legend.
      READ (9,60) ITM_NM
      CALL MOVABS(-15,-170)
      CALL TEXT1(6,'LEGEND')
      X = (((1 - ITEMS) * 200) / (ITEMS)) - 25
```

```
      INCX = (2 * IABS(X)) / (ITEMS)

c  do it for the remainder of the items
      DO 10,I = 1,ITEMS

        CALL SETCOLOR(I)
        LEN = LENTH (ITEM_NM)
        CALL MOVABS (X,-200)
        CALL TEXT1 (15, ITM_NM)
        CALL MOVABS (X, -225)
        CALL RECREL (20,8)
        CALL MOVABS (X + 3, -222)
        CALL AREA1
        X = X + INCX

c  get the next item
        READ (9,60) ITM_NM
c
10    CONTINUE
C
C
      I = 1
      MAXQ = 0

C  READ IN THE DATA FOR THE OBSERVATION PERIODS
20    READ (9,*,END = 30) (DAT(I,tj), tj = 1,items), time(i)

C  FIND THE MAX FOR SCALING PURPOSES
```

```
      DO 25,J = 1,ITEMS
       IF (DAT(I,J).GT.MAXQ) THEn
        MAXQ = DAT(I,J)
       ENDIF
25     CONTINUE

      I = I + 1

     GO TO 20


30    CLOSE(9)

C  COMPUTE THE SCALING FACTORS
     yearS = I - 1
     INCX = 330 / ((yearS * ITEMS) + yearS)
     WIDTH = 2 * INCX / 3
     PRINT *,'INCX = ', INCX
     X = -INT(REAL(INCX * ITEMS + 1) * (REAL(yearS) / 2.0)) - 20
     PRINT *, 'INCX = ', INCX, ' LASTX = ', LASTX
     YSCALE = 240.0 / REAL(MAXQ)

35    CONTINUE

C  LABEL THE AXES
     tinc = float(maxq)/5.
     ylbl = 0.
     CALL VAL8(20)
```

```
      call movabs(-205,-128)



      do 41 jj = 1,6
       print*,ylbl, 'ylbl',maxq,tinc,'tinc'
       call nchr(ylbl, cstr, ilen)
       call text1(ilen,cstr)
       ylbl = ylbl + tinc
       call movrel(0,50)
 41   continue

C  DRAW A SET OF BARS FOR EACH YEAR
      DO 40,I = 1,yearS


       CALL VAL8(20)
       call movabs(x + items*incx/2,-130)
       call drwrel(0,-5)
       call nchr(float(time(i)), cstr,ilen)
       call movrel(-5, -8)
       call text1(ilen,cstr)


C  DRAW THE BARS FOR THAT YEAR
       DO 45,J = 1,ITEMS


        CALL SETCOLOR(J)
        Y = INT((REAL(DAT(I,J)) * YSCALE)) - 125
        CALL MOVABS(X,-125)
        CALL RECTAN(X + WIDTH,Y)
```

```
      CALL MOVABS (X + 2, Y - 2)
      CALL AREA1
      X = X + INCX

45    CONTINUE

C  GOTO THE NEXT YEAR
      X = X + INCX
40    CONTINUE

      CALL QUIT
      STOP

50    FORMAT (7X,I1)
60    FORMAT(12X,20A2)
110   FORMAT(8(I10))

      END
C
C
C
C  This routine sets the current color for the graphics DRAW commands.
      SUBROUTINE SETCOLOR(I)

      INTEGER I
C
      IF (I.EQ.1) THEN
```

```
        CALL VALUE (230,0,0)
C
      ELSE IF (I.EQ.2) THEN
        CALL VALUE (0,230,0)
C
      ELSE IF (I.EQ.3) THEN
        CALL VALUE (0,0,230)
c
      ELSE IF (I.EQ.4) THEN
        CALL VALUE (230,230,0)
c
      ELSE IF (I.EQ.5) THEN
        CALL VALUE (230,0,230)
c
      ELSE IF (I.EQ.6) THEN
        CALL VALUE (0,230,230)
c
      ELSE IF (I.EQ.7) THEN
        CALL VALUE (230,230,230)
      ENDIF

      RETURN
      END
C*********************************************************************
C     THIS PROGRAM READS THE FILE 'CHART.DAT' AND
C     PRODUCES A LINE CHART FROM IT
C*********************************************************************
C
```

```
      INTEGER   DAT(50,7)
      INTEGER   TIME(50), tj
      INTEGER   ITEMS,I,J,LEN,X,Y,LASTX,LASTY(7),INCX,YEARS,MAXQ
      INTEGER*2 PASLEN
      INTEGER*2 ITM_NM(16), cstr(10)
      REAL      YSCALE
C
c  Initialize the rastertech display
      CALL INITIL

c  get the number of different categories to display
      READ (9,*) ITEMS
      ITEMS = ITEMS - 1

C  GET THE NAME OF THE FIRST CATEGORY
      READ (9,60) ITM_NM
      X = (((1 - ITEMS) * 200) / (ITEMS)) - 25
      INCX = (2 * IABS(X)) / (ITEMS)

C  GET THE NAMES OF THE OTHER CATEGORIES AND RENDER THEM IN THE LEGEND
      DO 10,I = 1,ITEMS

C  RENDER THE NAME OF THE CURRENT CATEGORY
        CALL SETCOLOR(I)
        LEN = LENTH (ITEM_NM,16)
        CALL MOVABS (X,-200)
        PASLEN =  (LEN + 1) / 2
        CALL TEXT1 (PASLEN, ITM_NM)
```

```
      CALL MOVABS (X, -225)
      CALL DRWREL (20,0)

C  GET THE NEXT CATEGORY
      READ (9,60) ITM_NM
      X = X + INCX

10    CONTINUE

C  RENDER THE LAST CATEGORY NAME
     CALL MOVABS(-15,-170)
     CALL VAL8(20)
     CALL TEXT1(15,ITM_NM)

     I = 1
     MAXQ = 0

C  READ IN THE DATA FOR THE OBSERVATION PERIODS
20    READ (9,*,END=30) (DAT(I,tj), tj = 1,items), time(i)

C  FIND THE MAX DATA VALUE FOR VERTICAL SCALING
      DO 25,J=1,ITEMS
       IF (DAT(I,J).GT.MAXQ) THEN
        MAXQ = DAT(I,J)
       ENDIF
25     CONTINUE

      I = I + 1
```

```
C  GET THE DATA FOR THE NEXT OBSERVATION PERIOD
     GO TO 20

30   CLOSE(9)

C  DETERMINE THE SCALING FACTORS
     YEARS = I - 1
     INCX = 330 / YEARS
     LASTX = -INT(REAL(INCX) * (REAL(YEARS) / 2.0))
     YSCALE = 240.0 / REAL(MAXQ)


     DO 35,J = 1,ITEMS
       LASTY(J) = INT((REAL(DAT(1,J)) * YSCALE)) - 125
35   CONTINUE


     tinc = float(maxq)/5.
     ylbl = 0.
     call movabs(-205, -128)
     CALL VAL8(20)

C  SET UP THE AXIS LABELS
     do 41 jj = 1,6
       print *,ylbl, 'ylbl'
       call NCHR(ylbl, cstr, ilen)
       call text1(ilen, cstr)
```

```
      ylbl = ylbl + tinc
      call movrel(0,50)
41    continue



     DO 40,I=2,YEARS
      CALL VAL8(20)
      X = LASTX + INCX
      call movabs(x, -130)
      call drwrel(0,-5)
      call NCHR(float(time(i)), cstr, ilen)
      call movrel(-5,-8)
      call text1(ilen, cstr)

C  PLOT THE LINE
      DO 45,J=1,ITEMS
       CALL SETCOLOR(J)
       Y = INT((REAL(DAT(I,J)) * YSCALE)) - 125
       CALL MOVABS(LASTX,LASTY(J))
       CALL DRWABS(X,Y)

C  REMEMBER WHERE THE LINE LEFT OFF
       LASTY(J) = Y

C  PLOT THE NEXT SEGMENT
45     CONTINUE

      LASTX = X
```

```
C  START THE NEXT LINE
40    CONTINUE

      CALL QUIT
      STOP

50    FORMAT (7X,I1)
60    FORMAT(12X,20A2)
110   FORMAT(8(I10))

      END
C
C
C  MAP INTO THE COLOR TABLE USING THE INDEX I
      SUBROUTINE SETCOLOR(I)

      INTEGER I

      IF (I.EQ.1) THEN
        CALL VALUE (210,0,0)
      ELSE IF (I.EQ.2) THEN
        CALL VALUE (0,210,0)
      ELSE IF (I.EQ.3) THEN
        CALL VALUE (0,0,210)
      ELSE IF (I.EQ.4) THEN
        CALL VALUE (210,210,0)
      ELSE IF (I.EQ.5) THEN
```

```
    CALL VALUE (210,0,210)
ELSE IF (I.EQ.6) THEN
    CALL VALUE (0,210,210)
ELSE IF (I.EQ.7) THEN
    CALL VALUE (210,210,210)
ENDIF


RETURN
END
```

The 4 page vita has been removed from the scanned document

The vita has been removed from the scanned document

The vita has been removed from the scanned document

The vita has been removed from the scanned document