

**Automating the Design of Packaging Families Using  
*PackIT*, The Packager's Inferencing Tool**

by

Thomas R. Amari

Bachelor of Arts  
Boston University  
Boston, Massachusetts  
1984

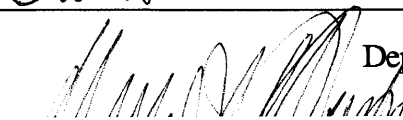
Submitted to the Department of Architecture  
in Partial Fulfillment of the Requirements of the  
Degree

MASTER OF SCIENCE IN VISUAL STUDIES  
at the  
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

JUNE, 1987

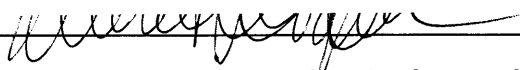
© Massachusetts Institute of Technology 1987

Signature of the author \_\_\_\_\_



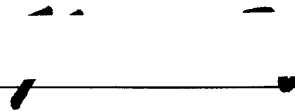
Thomas R. Amari  
Department of Architecture  
May 15, 1987

Certified by \_\_\_\_\_



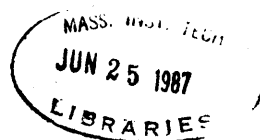
Muriel Cooper  
Associate Professor of Visual Studies  
Thesis Supervisor

Accepted by \_\_\_\_\_



Nicholas Negroponte  
Chairman  
Departmental Committee for Graduate Students

**Rotch**



Automating the Design of Packaging Families Using  
*PackIT*, The Packager's Inferencing Tool

by

Thomas R. Amari

Submitted to the Department of Architecture on May 15, 1987  
in partial fulfillment of the requirements for the degree of  
Master of Science in Visual Studies

**Abstract**

While it is generally accepted that knowledge-based computer systems play important roles in a great number of fields, their usefulness has not been fully explored in the area of graphic design. Graphics has been employed as an expert system user-interface tool, but rarely are these roles reversed with an expert system serving as a tool within a workstation for graphic design.

This thesis describes *PackIT, The Packager's Inferencing Tool*, a rule-based expert system which investigates the value of such systems in the field of packaging design. It is generally held that the domain of graphic design is too subjective and informal, and is therefore not amenable to the level of precision required by knowledge-based systems. This thesis attempts to illustrate that there are, in fact, objective guidelines frequently used by graphic designers which may be identified and encoded within a formal knowledge base. The design of packages was chosen as an area of study because it provides a unique model of two dimensional and three dimensional graphic problems.

PackIT's specific domain of expertise is in the area of package families and product lines. Since each package in a family must be unique and still retain a visual unity with other members of the family, its design is subject to a number of strict rules. If a new member of the package family with a different shape is to be designed, the graphic objects appearing on this package must be altered in various ways in order to correctly fit. An expert system, with its rule base incorporating guidelines of design would provide the resources to do this automatically. Such rules have been formalized in PackIT, which is an initial attempt at addressing the issue of computers as design assistants for visualization.

A variety of informal knowledge acquisition methods were employed to gather PackIT's limited but expandable rule base. Among them were: discussions with packaging designers, observation of designers at work, examination of an extensive family of packages, analysis of corporate identification manuals and packaging texts, and feedback from informal testing of the system. The rules consider design, packaging, and marketing strategies, with specific reference to color, spatiality, and relationships between graphic elements.

This work was made possible by the generous support of Hell GmbH, West Germany and Hewlett-Packard.

Thesis Supervisor: Muriel Cooper  
Title: Associate Professor of Visual Studies

## CONTENTS

---

Chapter 1	Introduction	6
	The Package Designer's Workstation	8
Chapter 2	Why is PackIT Needed?	12
	The Package as a Marketing Tool	12
	Usefulness of Knowledge-Based Packaging Systems	13
	Scenario	14
	How PackIT Can Achieve	17
Chapter 3	Design as an Application Domain	19
	Criteria For Choosing an Expert System	20
	Differences Between Conventional and Design Expert Systems	25
Chapter 4	A Packaging Family Case Study	30
Chapter 5	Knowledge Acquisition For Design Expert Systems	37
	Knowledge Acquisition For PackIT	39
Chapter 6	Rules For Design Expert Systems	45
	Techniques of Art and Design as Rules	45
	Packaging Rules Extracted From Texts and Manuals	52
Chapter 7	How Strict are the Rules of Design?	64
	When a Designer Chooses to Break a Rule	67
Chapter 8	PackIT's Rules	71
Chapter 9	The PackIT Expert System	80
	Programming Environment	81
	The Structure of Rules	82
	The Inference Engine	83
	Levels of Inferencing	86
Chapter 10	Features of PackIT	89
Chapter 11	Future Work	94
Chapter 12	Conclusion	102

Appendix A	Expert Systems	106
	Successful Systems	107
	Structure of Expert Systems	109
	Developing an Expert System	114
Appendix B	The Bad News	115
	Weak Points of Expert Systems	116
Appendix C	Guidelines for Knowledge Acquisition	122
Appendix D	The Graphic-Object Class	125
Bibliography		127

## **ACKNOWLEDGEMENTS**

---

*I would like to thank the following people for making this work possible:*

Muriel Cooper, for giving me guidance and providing me with the opportunity to work in such a unique and fertile research environment. Ron MacNeil for his abundance of suggestions and ideas. Patrick Purcell for his expert advice and invaluable support. Alka Badshah for getting me started. Russell Greenlee for technical consultation and academic companionship. Carl Olson for helpful ideas and professional advice. Bob Sabiston for his spectacular graphics software and imagery. Marybeth Randall for her concern and assistance. Eileen Baird for advice and suggestions. Adina Sabghir for her darkroom wizardry. Frithjof Zöllich for helpful comments. Rebecca Kowals for her enduring support and concern. And I would especially like to thank Gloria and Joseph Amari for the years of encouragement and enthusiastic support which has made all of this possible.

## CHAPTER 1

---

### *Introduction*

In past years there has been a steady increase in the number of design applications for which computers have become vital components. In the more recent past, the range of computer usage has extended into the field of graphic design. While computerized design systems are often much more powerful than the traditional tools of the trade, they are rarely capable of providing design knowledge or expertise in any form. Suppose, for a moment, that a graphic artist is attempting to design a layout in a computer graphic environment. It seems that it would be immensely valuable if the computer could assist this artist, not merely as a new medium for expression, but rather as a kind of knowledgeable consultant.

While it is generally accepted that artificial intelligence (AI) and expert systems play important roles in a great number of fields, such as medical diagnosis and geological exploration, their usefulness has not been fully explored in the area of visual design. After some consideration it becomes clear that the process of layout, with its immeasurable number of guidelines, is a logical area for the use of such a system. These guidelines that govern the design process may come in the form of practical or aesthetic considerations, and in either case may be captured in the form of a rule base. (Those readers not familiar with the concepts of expert systems are requested to refer to appendix A before continuing.)

This paper describes *PackIT, The Packager's Inferencing Tool*, a rule-based expert system which represents an initial attempt at using an intelligent computer system to assist

designers in the layout of package families or product lines. If a company has a number of related products, it is of great importance that the designs project the company identity throughout the line of product packages. Each company has a number of rules which govern the design of any of their packages, such as placement of logos, background colors, typefaces used, etc. This is an attempt to retain some unity of visual style and form throughout their various products. The packages, while remaining distinct, have highly defined visual relationships with each other, and thus form an entire package family with qualities immediately identifiable as belonging to that company.



**Figure 1.1** *A package family.*

These concepts are of great importance to companies since the package is regarded as a primary advertising instrument. It is often the only thing that distinguishes one product from another in the marketplace. The designers must be aware of exactly which visual

factors are important for that company, and they must know what types of design will prompt certain responses from the consumer.

PackIT's rule base contains general rules of graphic design as well as the packaging rules of a hypothetical company. The user (who is expected to be a graphic designer) gives PackIT a number of graphic objects which are to appear on the package, as well as the specifications such as size and shape for each container desired. It takes these specifications and generates suggested layouts in keeping with that company's image. These specifications may be altered in order to create entire families of packages. PackIT is intended to present options and suggestions by following a simple set of rules (as would an apprentice designer) and thus serves as a visualization tool for the experienced designer.

In developing PackIT, the relationships that may exist throughout the members of a typical product line or package family have been formally defined and incorporated as rules within the knowledge base. One of PackIT's fundamental concepts is the positional relationships existing between graphic objects themselves on the package. The placement of an object is specified in relation to the various objects surrounding it, rather than as an absolute position. These well-defined links form a large tree structure specifying the package's object hierarchy.

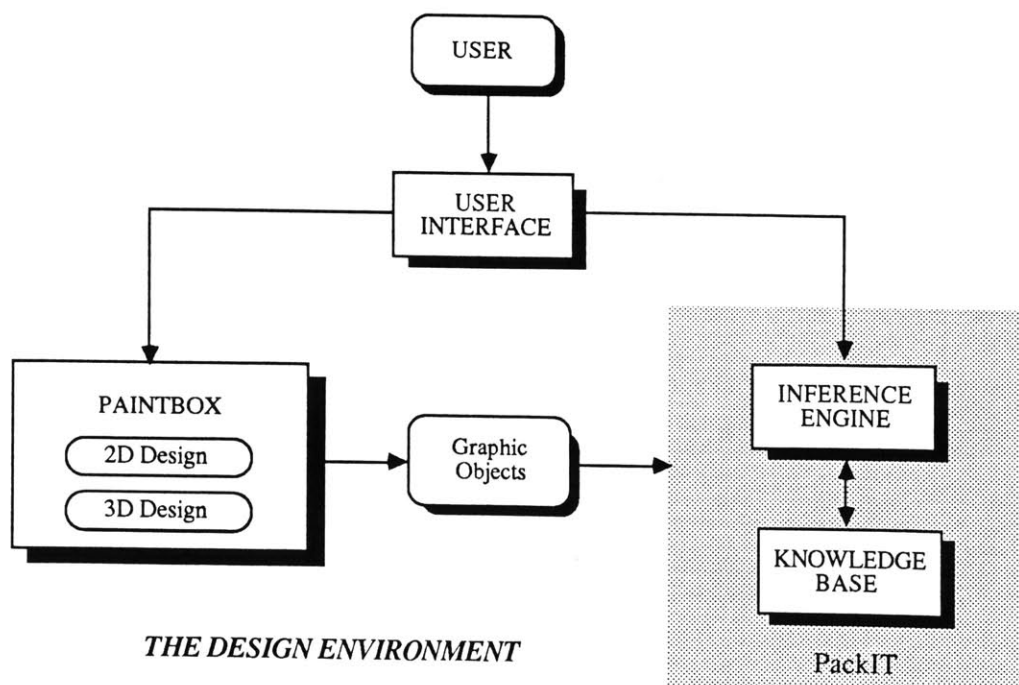
### **The Package Designer's Workstation**

The Visible Language Workshop (VLW) is a research group located within the Media Laboratory at the Massachusetts Institute of Technology. The primary concern of researchers at the VLW is to investigate the impact that the electronic revolution is having



on the areas of graphic arts and visual communication. PackIT has been created as a sub-system within a complete computerized design environment under development here. This ideal workstation incorporates two and three dimensional design facilities, knowledge-based graphics, sound as a user interface, and high-quality output.

As part of the PackIT design workstation, the user has access to computerized *paintbox* tools. This set of programs is based on the workspace of a painter or graphic artist, but also offers many features that could only be provided in a computer setting. Tools were developed which permit the artist to work in both a two dimensional and a simulated three dimensional workspace. The user is able to use these tools to create and edit individual graphic objects which are to appear on the package layout.



**Figure 1.2**

*The user initially creates a number of individual graphic objects with Paintbox software. These objects are then sent to the PackIT environment where they will be arranged to form the actual package layout. The interface serves as an intermediary between the user and the inference engine, which will make the layout decisions by accessing rules in the knowledge base.*

The *package sizing* function of the workstation clearly illustrates the need for an assisting expert system to aid in the design process. This function permits the enlargement or reduction of the three dimensional view of the package along any of the three axes  $x$ ,  $y$ , or  $z$ . When the package shape is altered using this function, the image on each of the sides is merely squashed or stretched into a distorted form. It is important, however, that images do not necessarily become distorted, but perhaps become scaled differently or arranged in a different fashion on the package. PackIT, with its rule base incorporating the knowledge of layout and design, provides the resources to do this automatically.

After observing the process of packaging design it becomes evident that the computer is a likely candidate for usage in such an environment. When a packaging firm is hired to create a package for another company, their designers might have to make hundreds of sketches before deciding which of them are good enough to be presented to their clients. Then actual prototype packages (or *mockups*) must be constructed by hand for each of the designs chosen. The process of making a mockup involves the tedious chores of cutting and pasting various photos and drawings, in addition to the drawing of different graphic elements onto the package with pencil, magic markers, etc.

Upon presentation, the clients may decide that they like the package except for a few minor details, or they may not like it at all; or the client might think it's absolutely perfect, "except for one thing...." The designers must then alter the layout and create a new mockup for presentation at a later date. Even the slightest change requested by the client may require the complete reconstruction of the sample package from scratch. This iterative cycle, which normally could take weeks, can be drastically reduced with the use of a computer-based system.

PackIT is not a substitute for a human designer and does not attempt to design as well as an expert can. (Some obstacles of trying to use expert systems in place of human experts are discussed in appendix B.) PackIT is a visualization and experimentation tool which augments the creative processes of the designer. It represents an investigation into knowledge-based graphic design. Although graphics has been used by a number of expert systems as a user interface aid, this thesis attempts to illustrate that these roles can be reversed, with an expert system serving as a tool within a larger design workstation.

PackIT was initially implemented to handle only boxes. Although its rule base would have to be expanded to accommodate the different shapes and surfaces of cans, jars, bottles, etc., the general concepts discussed in this thesis would remain the same.

PackIT is referred to as a *design expert system* in order to distinguish it from the more conventional expert systems in use today.

---

## *Why is PackIT Needed?*

### **The Package as a Marketing Tool**

Before discussing why a system like PackIT is useful, its domain must be justified as an important area for research. One might ask, “What makes package design a logical application for the development of a knowledge base?” Stanley Sacharow, in his book *The Package as a Marketing Tool*, explains that packages were originally employed simply for the purpose of containing and protecting a product, and little or no thought was put into making the appealing containers that have become commonplace (and even necessary) today.

He describes the new role the package took on:

Until about 1900, a package had to perform only two functions: (1) to ensure safe delivery of the product to the consumer, and (2) to protect the product and ensure its shelf life. But at the turn of the century an entirely new dimension was added to the emerging field of packaging technology. The package itself became a sales agent: It had to be able to “protect what it sells and to sell what it protects.” (Sacharow 1982, 1)

This was clearly an important development because it became necessary to put great consideration into the design of these packages, which, for the first time, had to reflect the ever-changing aesthetic trends and demands of the consumer.

Perhaps the first company to promote the sale of its product through the use of packaging graphics was that National Biscuit Company (Nabisco) in 1895. The display of the *Uneeda* biscuit carton not only protected its contents, but was also an advertising agent thus becoming a landmark in the concept of promoting brand loyalty (Sacharow

1982, 4). Other companies soon followed this trend, which evolved from a refreshing novelty to the packaging requisite of today.

Other milestones include the 1930 birth of King Kullen Supermarkets in Queens New York, which served as a showcase for what must have appeared to the contemporary consumer as an infinite supply of intriguing and enticing products. *“Pile it high, sell it low”* was the motto (Sacharow 1982, 5). The supermarket sped up the utilization of packaging graphics by providing an environment where companies were able to compete for the consumer’s interest. Package design as a marketing tool was the ground-breaking concept, and the inception of supermarkets was the catalyst for the multi-billion dollar industry that we take for granted today.

### **Usefulness of Knowledge-Based Packaging Systems**

In May of 1986, the VLW hosted a package design forum, where a number of experts in the field made presentations. One of the most striking of these was given by Larry Dostal, Vice-President of the packaging firm, Richardson/Smith. His firm had designed the packages for a line of Black & Decker products. This line was quite extensive, consisting of approximately 3,000 items. While the entire family was governed by a strict set of rules, each package was unique.

Dostal presented us with a representative sample of a few dozen of these containers. As we reviewed the content and organization of their labels, we were all able to identify many of the rules that were followed. Upon further inspection, however, one participant noticed a rule violation appearing on one of the packages. It dealt with the arrangement of text on one of the smaller labels.

Soon thereafter, one or two more violations were discovered. These discrepancies were evidently not the result of designer choice, however – they simply appeared to have been overlooked during the design process. This is understandable when one considers that the set of rules to be followed was quite large.

It was subsequently agreed by the participants that a knowledge-based computer environment would be well-suited to handle such situations, especially where rules of great detail are large in number or the package family is extensive. The designer would be liberated from consciously adhering to these numerous constraints and could concentrate more fully on solving the fundamental design problem.

### **Scenario**

Let us consider the designer who has been asked to create a package for a new product in an already existing family of packages. The company who has developed this product prides itself in being a conspicuous force in the marketplace – its packages are highly distinctive and, in the consumer's eye, have become synonymous with the company itself. The task of the designer is a sizable one: he must create a unique design which captures the essence of the new product and which simultaneously conforms to the standards of the other family members. The new package must serve as a powerful representative of the company in the marketplace.

Let us further assume that the designer has access to a computerized workstation. This workstation provides tools such as color manipulators, image sizing facilities, and paintbrush editors. These tools, while greatly valuable, are merely ignorant agents which facilitate the fundamental activities of the user. But what if the computer were to

participate in the actual design of the package? What if it had some of the problem-solving knowledge of the designer himself? Such a system could contain not only general rules of graphic design, but also the numerous, highly-specific rules of the individual company – rules which often become burdensome to the designer and impede his creativity.

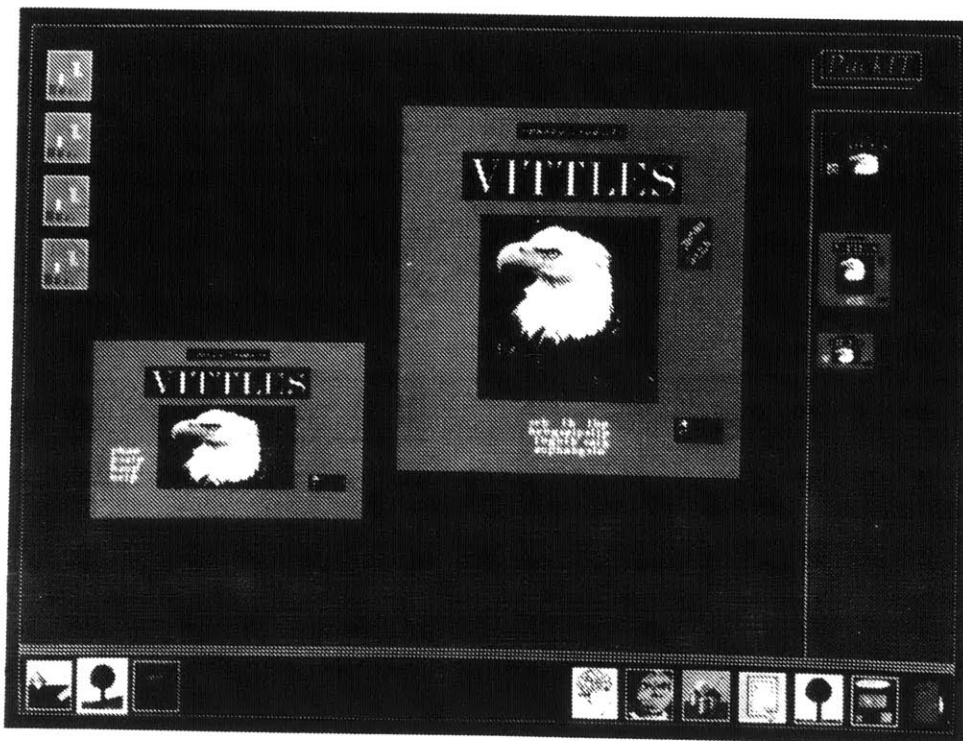
PackIT represents the initial stages of such a system. Its knowledge base contains a number of graphic design rules, in addition to the more specific rules of a particular hypothetical company. While design itself is a high-level process, these rules serve as a filter and a source of guidance for PackIT's user (who is ideally a graphic designer of some expertise). The user is able to create individual graphic objects with the computer painting system described in chapter 1. He may then specify the particular package characteristics of this member of the family, such as size, shape, marketing area, etc. PackIT will take these objects and, using its rulebase as a guide, lay them out in an appropriate fashion on a computer display of the package's side. In doing so it may find it necessary to resize, reshape, crop, exchange, or eliminate any of the objects available to it.

After PackIT has completed its task, the user may manually reposition any of the elements on the package, or choose to rerun the process with slightly different specifications. PackIT is thus a visualization tool for the designer; it is not expected to produce a final composition. It does, however, strictly follow the rules given it, and quickly present a number of alternatives to the user. Its value is readily apparent when one considers the *Black & Decker* scenario previously described.

It is important to mention that this scenario has two versions. The one described above presents the case where a designer is asked to take an already existing family and add a new package to it which conforms to the style of its predecessors. But there is

another closely related version of the story where the designer starts without an existing family and actually creates the style and defines the relationships which are to permeate the family in the future.

PackIT is able to handle both situations either by allowing the user to specify all the package characteristics himself, or by providing default settings to serve as the prototype after which other packages are modelled. This prototype package is defined in PackIT's knowledge base.



**Figure 2.1** *The PackIT Interface.*



### **How PackIT Can Achieve**

One might ask “how can a computer system incorporate the ‘creative expertise’ of a human designer?” The answer is, it can’t. It can, however, attempt to simulate the expertise displayed by designers. Rather than dealing with complex issues like analysis of form or content, it can access lower-level techniques, the kind which might serve to guide an apprentice designer who is not yet ready to work intuitively and must often mimic the actions of his master. While design expert systems may not be capable of creating ground-breaking works of art, they can be adept at simulating aesthetic intelligence by utilizing these types of guidelines and strategies. I propose that this intelligence is no more mysterious than that of a chemist or physician (to cite examples from the more successful conventional expert systems). Rather, the difficulty lies in accessing and formalizing the designer’s knowledge, which may be relatively informal initially.

The means by which design expert systems may contain this knowledge will be presented in a subsequent chapter. The utilization of basic design rules dealing with color, symmetry, proximity, etc. are quite sufficient, and seem to produce perfectly satisfactory results in PackIT. Furthermore, there is no reason why with further investigation and investment, this “satisfactory” label cannot be upgraded to the status of remarkable.

Overall it seems that we should judge an expert system by its ease of use and the results it generates above all else. As Bruce Buchanan states in relation to computer learning:

Expert systems currently do not improve their own behavior based on experience: does that mean that they are not ‘really’ expert? This is a definitional question on which one

may take a dogmatic stand. However, if we use a performance-based definition of expertise and not a dispositional one, then we may be less dogmatic and say that a person, or a program, is an expert by virtue of excellent performance, regardless of how he, she, or it gained his/her/its knowledge in the first place (and kept it current). (Buchanan 1986, 39)

The typical criticism of whether expert systems truly function like human experts is not really an issue here. (Many people strongly believe that “expert” system is a misnomer.) What is important is PackIT’s performance in assisting designers at work.

*Design as an  
Application Domain*

The number of application areas for knowledge-based systems is quite extensive, ranging from medical diagnosis to geological exploration. These areas are considered to be the typical realms of knowledge engineering. Absent from virtually every AI article and text, however, are discussions of graphic design as an application area. It seems the only layout issues commonly described are those relating to the configuration of VAX computer workstations or the design of integrated circuitry. It is, in fact, quite rare to find a graphics workstation which uses an expert system as a design tool.

This chapter attempts to explain why this is so. The first section describes the criteria for choosing an expert system application area and states whether a design environment passes or fails in each case. The second section discusses some of the differences between conventional expert systems and expert systems with graphic design as a domain.

### Criteria For Choosing an Expert System

For our purposes, we might define artificial expertise as human expertise which has been entered into a knowledge-based system. The following descriptive lists present some advantages in using artificial expertise over human expertise:

Human Expertise

Perishable

Difficult to transfer

Difficult to document

Unpredictable

Expensive

(Waterman 1986, 12)

Artificial Expertise

Permanent

Easy to transfer

Easy to document

Consistent

Affordable

Artificial expertise has such advantages as being more permanent and reliable, whereas human knowledge may be inaccurate or transient. Great care must be taken, however, when deciding whether or not an area is suitable for the development of an expert system. It is not uncommon to find situations where, for one reason or another, expert system development is not appropriate. Three of the most important criteria specify that a domain is appropriate for implementation as an expert system when:

conventional algorithmic techniques are unsatisfactory,  
the domain requires the knowledge of an expert,  
and the knowledge may be clearly defined.

Below is a description of these three criteria that an acceptable application domain must satisfy, stating whether the design application passes or fails each test.

- **Conventional algorithmic techniques are unsatisfactory.**

An algorithmic technique is used when no inferencing is required, the solution to a problem can be generalized for all cases of that type, and the results are expected to be fully accurate. Problems which may be solved with such a step-by-step procedural approach are generally not good areas for an expert system. In such cases, an expert system, which excels in analyzing a situation and choosing the correct path to reach a goal, is not necessary. In fact, it would probably take longer to reach a solution here since the most efficient path is already known from the start, thus making any inferencing extraneous. Donald Waterman, in *A Guide To Expert Systems*, gives an example of the difference between an algorithmic and a heuristic solution (as might be used by an expert system) for the problem of skyjacking on commercial airlines:

*ALGORITHMIC: Strip search every person given access to the airliner and search all luggage. This includes passengers, flight crews and mechanics.*

*HEURISTIC: Search only those passengers that set off a metal detector or match a predetermined skyjacker profile (based on certain characteristics such as age, dress, behavior, etc.). (Waterman 1986, 17)*

That is not to say that experts do not employ algorithmic techniques in their work – they do, of course. Many of their sub-tasks are most effectively solved this way; however, their overall goals should require inferencing and application of rules. This is precisely what PackIT does. While its inference engine is used to decide where a particular graphic object should be placed, for example, it is a simple algorithm which determines whether this placement has resulted in any overlap of objects. Graphic designers certainly do not merely follow algorithms alone when working, and so this criterion is met.

- **The domain requires the knowledge of an expert.**

In general, knowledge bases are effective at simulating intelligence in highly-defined, specific areas, and not in areas where more a broad range of knowledge is required.

One feature of expertness is that it usually comes in *narrow, specialized domains*. For example, it is easy to imagine an expert in almost any technical field (say, mass spectrometry or protein crystallography), but not in such everyday activities as understanding natural language or visual scenes. Specialization in expertness seems to reflect a trade-off between depth and breadth of knowledge: one can know a great deal about only a small number of things. Specialization and expertness go hand in hand. (Brachman et al. 1983, 42)

Unfortunately, while a graphic designer's knowledge is, for the most part, highly specialized, it also covers other broad areas including human vision and knowledge of aesthetics. These domains are very difficult to represent symbolically, primarily because such a knowledge base attempting to rival the intelligence of a human would have to be immense. This is besides the fact that most people do not believe that knowledge of aesthetics can be adequately represented by a computer at all.

- **The knowledge should be clearly defined.**

It is of great importance that the knowledge of the chosen domain be clearly defined. The heuristics involved must be adequately describable and consistent from one expert to another. Expert systems are most effective when the rules they use are widely accepted and are not controversial or debatable. The purpose of this is to partially assure that the results generated by the system will be comparable to that of a human expert. If the rulebase is dominated by unconventional ideas, the results might not be reliable.

The pertinent issue for the development of an expert system with a design domain is

whether there are explicit rules of design that may be utilized here. Most graphic artists would probably hesitate when asked to write down a list of strict rules that should be consistently obeyed. Some would even state that there are no strict rules at all, or if there are they should be abandoned for the sake of originality. Many designers might also refer to this list with the use of some euphemism like “guidelines.”

It is evident, however, that there must be some rules that should be followed. In his book Grid Systems in Graphic Design, Josef Müller-Brockmann states the following rule quite unambiguously: “Margins of the same size can never result in an interesting page design; they always create an impression of indecision and dullness.” (Müller-Brockmann 1981, 41) To take an even more extreme case, who would dispute that a randomly generated layout would most likely be quite poor? Such a process should produce exactly the same results as could the most qualified design expert if he or she had no access to any rules at all! Indeed, anyone who critiques a particular design is simultaneously referring to rules. It is not sufficient to belittle a work simply because “it doesn’t look good.” Rather, the important task here is to describe why it is no good. The reasons why designers discriminate between choices must be uncovered. It turns out that perhaps the most critical aspect of PackIT’s development was to analyze and formalize the processes utilized by expert graphic designers. The means by which this was executed will be discussed later in this thesis.

It must be accepted that there are, in fact, describable rules to guide the design process. This is not the problem, however. A major criterion for developing a successful knowledge base is not only that rules exist, but that experts generally agree on their content – this is wherein the difficulty lies. It is simple enough to produce a rule of medicine, for example, and state it with a certainty factor of 0.8 (certainty factors are discussed in appendix A):

IF: The therapy under consideration is tetracycline,  
and the age of the patient is less than 13,  
THEN: There is strongly suggestive evidence (CF 0.8) that tetracycline  
is not a potential therapy for use against the organism.  
(Shortliffe 1984, 125)

Such rules are not so clear-cut in a design environment. While some physicians may disagree on the certainty factor by a few points, there is nothing to stop one designer from discarding with impunity another designer's rule altogether. There are no statistics to disprove the ideas of any particular designer.

One might argue, however, that design expert systems are quite different from other expert systems. The nature of design is such that complete consistency of rules may not be necessary – two designers might apply conflicting rules and still solve the same problem quite successfully.

Furthermore, we can initially bypass this difficulty (to some extent) by developing a knowledge base consisting primarily of those techniques which are universally accepted. These tend to be of the type taught in beginning and intermediate design courses. While one may accept that even these elementary rules are not fully concurrent throughout the design world, it is also true that expert systems are particularly adept at giving advice in environments where strict yes/no answers are quite uncommon. In fact, this is the type of challenge in which they are most able to display their power. Certainty factors are quite useful in specifying the strength of a rule or the validity of an assertion. In addition, a *generate and test* system like DENDRAL (see appendix A) can make a group of hypotheses which are suspected to be valid for a particular problem. Although the ultimate solution may not be attainable, DENDRAL eliminates the candidates one by one until only a small number of likely solutions remain.



## **Differences Between Conventional and Design Expert Systems**

### **Rule Credibility**

There are a number of significant differences between conventional expert system and those with design as an application domain. One of the most distinguishing characteristics of the former is the confidence with which rules may be formed. The system, upon request, must be able to provide justification for its final result as well as any inferencing decisions it has made in the process. In most cases, the knowledge can be based on empirical evidence, statistical analysis, or the like. If, for example, 60% of those patients who suffer from symptoms *a* and *b* die within one year of the symptoms' initial manifestation, the resulting rule may be formulated quite simply. Designs rules such as this, however, may rarely be stated with such assurity, simply because rigid data is not readily available in the design world. Can one be fully confident in saying, for example,

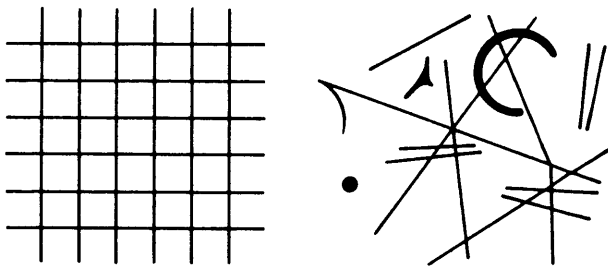
IF: Graphic object *a* is placed next to graphic object *b*,  
THEN: The layout is active (CF 0.8)?

Beside the fact that this rule is overly general for illustrative purposes, there is no indisputable evidence that it is completely (or even somewhat) accurate. While the certainty factor helps to alleviate some of the burden by providing an indicator of the rule's strictness, there may be some experts who would discard this rule altogether.

There are two straightforward ways to avoid this problem. The first is to consider rules more widely acceptable such as:

IF: Graphic object *a* is on the left side of the page,  
and graphic object *b* is on the left side of the page,  
and no other objects appear on the page  
THEN: The page is asymmetrical.

We may even formulate rules providable by experts which are much more subtle and are still considered universally acceptable. For example, Donis Dondis explains how the simple ordering of graphic elements projects a feeling of repose, while elements placed in various unexpected positions project stress (Dondis 1973, 25).



**Figure 3.1** *Two arrangements of elements projecting different feelings.*

This might be represented by providing rules of balance, or rules requiring constant (or varied) margins lengths between graphic objects, for example. Similarly, Müller-Brockmann bluntly offers the rule: “The fewer the differences in the size of the illustrations, the quieter the impression created by the design.” (Müller-Brockmann 1981, 11)

The second means by which the problem of rule credibility may be handled is (rather than through numerous supporting statistics) through sufficient support of expert opinion. Granted, opinion is not nearly as convincing as fact, but if a number of designers concur on a particular idea, at some point we may consider that knowledge to

be usable in our system. In some cases, it is traditional for the knowledge engineer to support his decisions with references from respected, credible sources in the literature (rather than through the interviews of a single expert).

### **Goals and Strategies**

Along the same lines comes a second difference between conventional and design expert systems. The argument is that while conventional expert systems deal with the processing of facts to arrive at inevitable conclusions, the tasks of design expert systems are much more nebulous. A human designer is required to sit back in the middle of a session and ask himself “does this really work visually?” If he does not think that it does, he must alter it. Although many times he might not even know why it is not satisfactory, there is formal knowledge residing beneath the surface of his talent. It is the job of the knowledge engineer to carefully observe and and analyze the activities of the expert in order to bring that knowledge forth. No action should be taken for granted; every choice made by the designer must be justified.

Paul Rand states that one cannot merely push things around on a page until it looks nice, as a novice might do (Rand 1970, 11). Such strategy would reflect an inefficient lack of direction and, therefore, an absence of the required knowledge. The knowledge engineer must now formalize the designer’s knowledge which has been, to a certain extent, somewhat more casual than is required for rule base development.

When considering the fact that design rules are somewhat less distinct than rules of other domains, we must keep in mind that expert systems are well-suited for situations where answers are not always clear-cut. “While conventional programs are designed to produce the correct answer every time, expert systems are designed to behave like experts, usually producing correct answers but sometimes producing incorrect ones.”

(Waterman 1986, 29) By the same token, perhaps we should accept that design expert systems may not always be quite as accurate as conventional ones. One might say that if the rules of a task are clear, the success rate may potentially be high; since design rules are not always completely clear, success may come less often. This might be why it is common for designers to make a number of sketches before deciding which approach works best.

### **Time Passage**

Another difference between these two classes of expert systems is the fact that the knowledge inherent in most conventional systems is relatively sheltered from disqualification resulting from the passage of time. In general, that which is true now will most likely be true in the near future. The following rule from the *PROSPECTOR* geological exploration system is not likely to change in the short run:

IF:     The igneous rocks in the region have a fine to medium grain size,  
          and they have a porphyritic texture,  
THEN: They have a texture suggestive of a hypabyssal regional environment.

The job of a graphic designer is to create a product which solves a particular set of problems by being both functional and what one might consider “good-looking.” The strategies and techniques used to accomplish this, however, evolve over time. An approach that was effective last year may not be quite as convincing today, perhaps due to changing tastes.

There does not appear to be any single strategy to avoid this problem. It might help to primarily incorporate those design conventions which have stood the test of time – if a technique has been used successfully for years, then it will probably remain valid for some time in the future. This might be done, for example, by adhering to the rules

provided by formalized styleguides such as corporate design manuals. Rules of corporate identity tend to remain relatively constant for individual companies. (Unfortunately, the body of knowledge provided by corporate ID manuals is inadequate by itself. Certainly, designers who use these manuals must also access other guidelines which are not quite as invariable.) It is also important to be aware that some rules are likely to change. Keep an eye open for these, and modify them when necessary.

### **Validity of Results**

Perhaps the most significant area of divergence between conventional and design expert systems pertains to the usefulness of their consultations. It is often the case that members of the former group solve problems by simply following the rules provided. If the knowledge base is accurate and the inference engine works well, then the advice given by the system will probably be quite valuable. The job of a graphic designer is also to solve problems; but this is often accomplished by devising new ideas or approaches. Are computer programs able to generate novel ideas? These ideas often come from a human's decision to break the current rules of design in order to obtain strikingly effective results. Surely one of the most significant characteristics of expert systems is their ability to perfectly adhere to the rules given them, unlike humans who sometimes make mistakes or forget. Do we want our design knowledge base to ignore or break rules under certain conditions? Is this even possible?

This problem is clearly one of the most difficult to satisfactorily resolve. Later on, some ideas on how this problem may be tackled will be presented. Some concepts include the use of particular design techniques and *metaknowledge* (knowledge about knowledge).

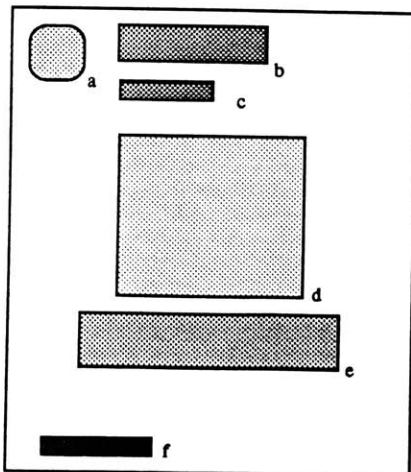
---

## *A Packaging Family Case Study*

Since PackIT's specific domain of expertise relates to the design of packaging families, it is useful to provide a case study describing the relationships that exist between members of these families. This chapter will describe the development of a line of related products for a hypothetical company.

Let us suppose that there is a company called Packo Industries which has a product that is already on the market. Packo decides it wants to create a line of similar products using the package of the original product as a prototype. The graphic objects of the package (which happens to be a box) are arranged as illustrated in figure 4.1.

### **Family Member #1: *Prototype***



**Figure 4.1**

*One of the most important concepts here is the relationship between individual objects on the package. Such relationships have a great impact on the possible adjusting of object placement and content when a new package is being developed.*

The new members of the family may be designed as shown in the following diagrams.

### Family Member #2: *Horizontal Orientation 1*

The company decides to introduce a version whose box is somewhat shorter than is found in the vertical orientation of the first box. The simplest means of redoing the layout would be to simply squash each of the objects on the package while keeping their positions constant; this is clearly unacceptable, however. While it might be permissible to alter the dimensions of a particular object on certain occasions (such as if the object is a simple geometric shape: a rectangle, ellipse, etc.), most objects may not be distorted as such. A human figure, for example, should remain in its proper dimensions. There are alternative means to compose the package, however (see figure 4.2).

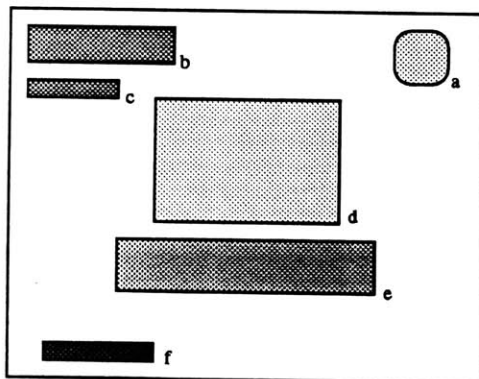


Figure 4.2

*In order to make room in the Y dimension (vertically), object a is moved right, and objects b & c are moved left; thus providing space in the upper center of the box. Object d (which has been cropped in order to make room) is now shifted up into this space. Objects e & f generally remain the same.*

Thus, the two means of creating space in figure 4.2 are position shifting and cropping in the y dimension. A number of points are of great importance here. First, note that the position of object *f* has remained constant; this is because it so happens that Packo's corporate design manual has restricted such a movement. The reasons for this

may be simple: *f* is an object which should remain in one constant position throughout the product line in order to create a visual company identity. The reason may also be more complex, dealing with aesthetic issues such as visual order or balance.

Second, while objects *b* and *c* have been shifted, the positional relationship between them has remained the same: *c* is still right justified below *b*. The reasons for this may be numerous, dealing with brand identification, legibility (if the objects are textual, they may be meant to be read together), or any number of design reasons. I will describe later that the power of PackIT is in its ability to make such judgements through chains of reasoning.

Third, object *c* was cropped at its bottom in order to make room. Packo provides strict rules as to which objects (or types of objects) may be cropped, and where they may not be cropped. The picture of the product may be cropped under certain conditions, but the logo may never be cropped, nor may text (although some items within a region of text may be shifted or deleted).

### Family Member #3: *Horizontal Orientation 2*

Packo now decides to introduce a member which is even more horizontally oriented (perhaps it is meant to be stored in a glove compartment or the drawer of a night table). Further changes must be made to compensate, as is illustrated in figure 4.3.

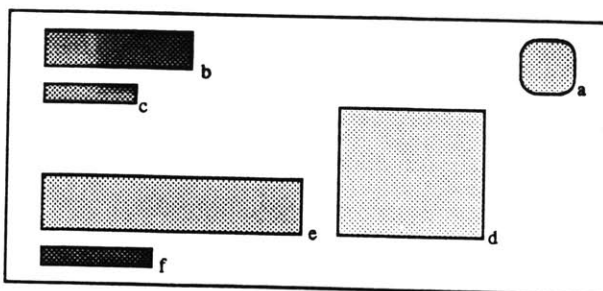


Figure 4.3

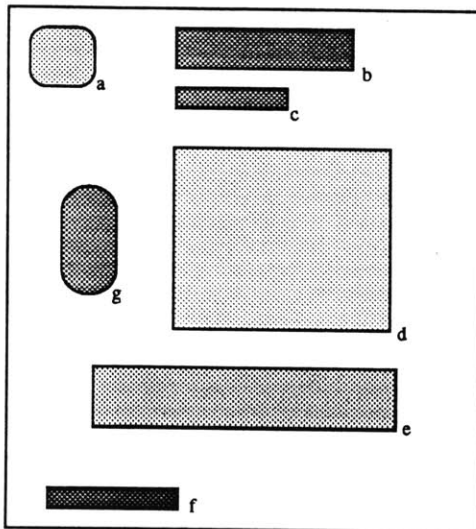
Object *e* is moved up and to the left side. *d* is uncropped, but is now scaled down. Other relationships remain the same.



Packo's design rules state that it is acceptable to relocate object *e* as shown, although only under rare circumstances may *a* and *e* appear juxtaposed. Note that *d* may now be uncropped again (in this family it is always desirable to display the objects in full) but it had to be scaled down in two dimensions.

**Family member #4: Large size**

Figure 4.4 shows Packo's large economy size box. This box looks more like the original with a few alterations.

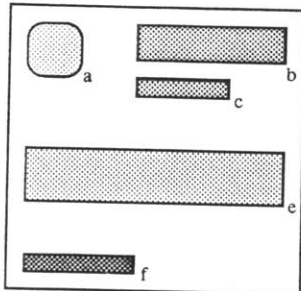


**Figure 4.4**  
*Similar to prototype. Objects are scaled up. d is moved right (now justified with e) in order to make room for g which is a new object (perhaps textual) indicating to the consumer that this is the jumbo size.*

**Family Member #5: Small Size**

In order to fit the objects on a smaller box (figure 4.5), the margins between them and the edges have been reduced. It was also determined that one of the objects had to be removed. While most objects must necessarily remain, such as the product and company names, there are times when it is acceptable to remove certain objects which are determined to be somewhat less vital. Such objects often include pictures of the product

or descriptive text.



**Figure 4.5**

*The margins on the left and right are now reduced. It is decided that one of the objects (in this case, object d) must be removed. Other objects, such as the company logo, product name, etc. are too important to remove.*

Steps like these are commonly found in actual packages on the market today, not only with different shaped containers, but also with different versions of the same container. Figure 4.6 shows two Polaroid film cartons: one is the standard box and the other is a specially priced version.



**Figure 4.6** *These two family members display strong stylistic relationships. Many of the design steps between the two are clearly evident.*

A number of objects had to be rearranged here. In order to make space for the *\$1.00 off* label, the company name had to be shifted upwards. So that the name would fit next to the *600* film type text, they both had to be reduced in size. The "Bright rich colors" text had to be removed altogether, as did the text describing the package as containing two rolls of film. Many of the steps followed are immediately identifiable.

While the new design worked quite well visually, it turned out that the product did not sell as well on the market as it should have. Since the box no longer informed the consumer that it was a *2 Pack*, they were not sure what they were getting for their money. This is an example where a rule stating that such text was necessary could easily be entered into the knowledge base of an assisting expert system.

Now let us suppose Packo wants to extend its family into other marketing areas (e.g. professional or industrial products) each with different levels of quality. This adds another dimension to the classes of rules and the number of possible package types. When types of family members are combined, the family may grow even more than indicated in the examples above. A company may want to have any of the following products:

Small, professional, high-quality.

Large, mass-marketed, economy.

Horizontally-oriented, industrial, standard-quality....

One potential family tree is illustrated in figure 4.7.

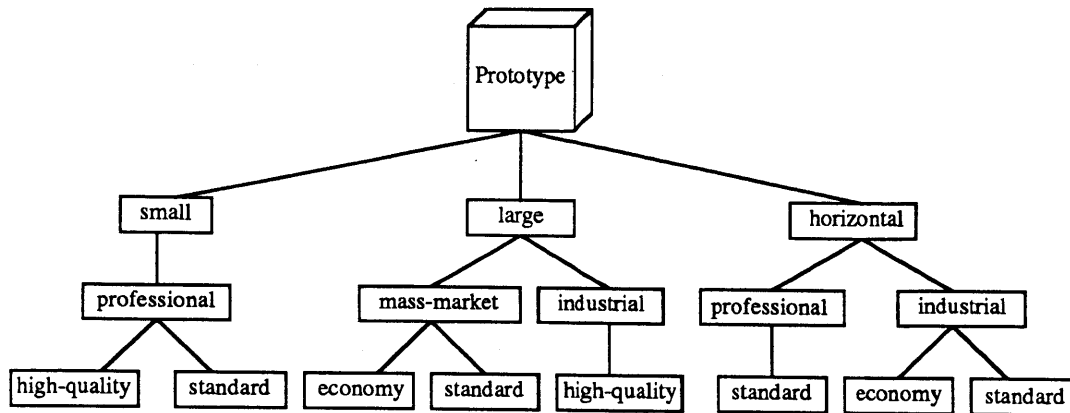


Figure 4.7

*The size of a packaging family can grow quite large even when only a small number of characteristics are considered. Illustrated here are eight family members developed from the same prototype using the three characteristics: shape/size, marketing area, and projected quality. The leftmost example is a small, professional, high-quality version of the prototype package.*

Although in developing PackIT I have concentrated on the creation of boxes as packages, many of the issues are similar when considering cans, bottles, jars, etc. The addition of these containers into the family enlarges the tree and the rulebase even moreso. Naturally, many of the rules would have to vary depending on the materials of the container.

The discussion above describes how Packco Industries has taken one prototype package, and developed an entire family of packages. The important point here is not that the changes shown are so complicated visually – they are often not. What is significant is that there may be an extensive set of rules governing this process; some of these rules are strict, some are not, some change as the layout develops, and some have effects on others. When carefully organized, they form an extensive body of knowledge which may produce complex lines of reasoning.

## *Knowledge Acquisition For Design Expert Systems*

One of the major steps in developing an expert system is the gathering of expertise which is to serve as the system's knowledge base. It is the task of the knowledge engineer to obtain this information from the domain expert.

The field under consideration for PackIT is certainly not one of the traditional areas of application and, not surprisingly, it requires a somewhat tailored protocol for the building of its knowledge base. This is mainly because, in general, the transmission of design technique from master to student is somewhat less formalized in this field than in others. Classes in design are often studio-type sessions where even beginning students learn by trial and error to some extent: the instructor gives an assignment, the student prepares a composition, the instructor critiques the work and relates how he would improve it, and the cycle starts over again. This is highly unlike the field of medicine, for example, where the student must study for years before applying his knowledge. Naturally, this is primarily due to the fact that the price of error is far greater in medicine than in graphic design. The important point is, however, that mastering design techniques is more of a learn-by-doing process than is found in most other disciplines.

What does this mean for the knowledge engineer? The answer is clear: since the principal method of learning graphic design is through observing and doing, *the primary means for knowledge acquisition should be through observation.* It is of great importance for the engineer to analyze the process of design and not merely interview the expert. In his book How Designers Think, Bryan Lawson describes the *Beaux Arts*

school of design where the most important aspect of design was believed to be the nature of the final product, rather than the process involved in attaining that product.

The inevitable reaction to such a philosophy generated a movement which sought more understanding of the design process itself. It was argued that the end product of design was too important a commodity for the process to remain such a neglected, hit and miss affair; society had a right to expect its designers to be responsible and accountable and have more control over their processes. (Lawson 1980, 2)

While question/answer sessions are common practice for other domains of expert system development, employing this technique alone is quite inadequate for design expert systems. This is true not only for the reasons stated above, but also because in many cases the designer simply does not know (or thinks he does not know) why he does certain things. If the processes underlying behavior are unconscious, then they may be only incompletely available for verbal reporting, if available at all. Furthermore, there is the objection of behavioral psychologists known as the *epiphenomenality* claim, which states that the processes which generate verbal reports are completely independent of those that generate our actions. (Ericsson and Pressler 1984, 109-10)

Many design rules are traditionally not as formalized as is required for computer implementation – it is the job of the knowledge engineer to overcome this. The knowledge lives within the designer's mind just as distinctly as does the knowledge of medicine within the mind of a physician. It is not some mysterious power granted at birth, nor is it the result of spontaneous enlightenment. Rather, it is (to some extent) an assemblage of heuristics which, when formalized, can be conveyed and utilized just like any other knowledge.

### **Knowledge Acquisition For PackIT**

A number of different methods were utilized for the development of PackIT's knowledge base. (See appendix C for some general guidelines for knowledge acquisition.) The following list describes some of the activities which directly or indirectly influenced its construction and revision:

- Observation and questioning of designers at work.
- Comparison and analysis of packages found in packaging design books.
- Review of corporate ID manuals.
- In-class exercise to elicit and formalize students' design procedures.
- Feedback from experts (and laymen) who tested PackIT itself.

These techniques utilize both styles of analysis referenced in Lawson's product versus process discussion above. The second and third techniques concentrate on examining the final product of design, while the first and fourth deal primarily with an analysis of the process of design.

These five methodologies are described below.

#### **An Expert at Work**

On November 20, 1986, a visit was made to MIT's Visible Language Workshop (VLW) by Joan Musgrave, Art Director of Technical Publications at IBM, Thornwood, New York. The purpose of this visit was to observe her during a design session in order to gain understanding of the process of design. This observation was specifically aimed at assisting another graduate student at the VLW, Alka Badshah, in the development of her system *GRID* (1987). *GRID* is capable of assisting in the process of page layout,

and has actually been used to help design the cover of a recent issue of IBM's journal, *Perspectives in Computing* (vol. 7, no. 1, 1987).

In order to get the most complete coverage of the session, we made use of a video-tape recorder in addition to the standard taking of notes. This ensured that most of the details were captured for later reference. Naturally, this is vital for the analysis of a visual design session.

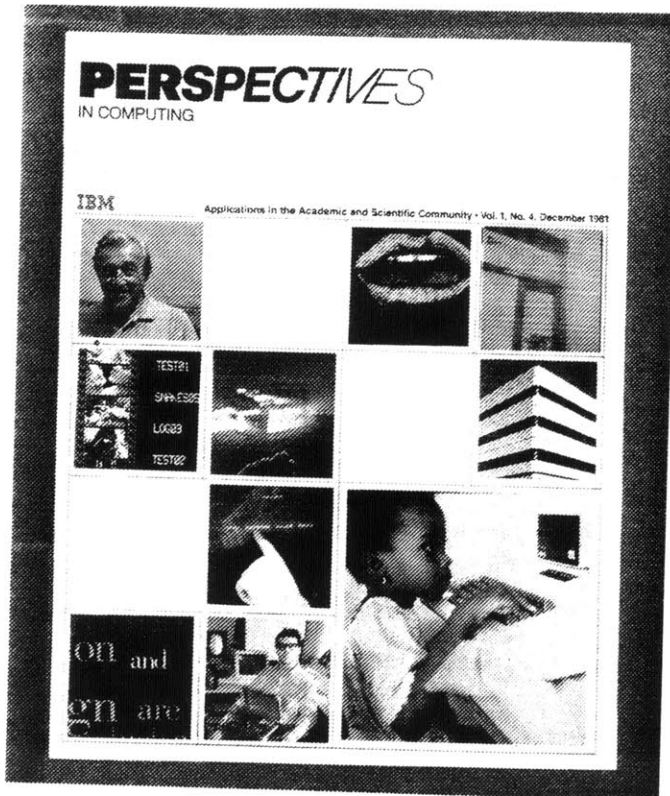
Musgrave's task was to create a magazine cover using a number of photographs provided to her. The observing group consisted of a few graduate students, professors, and myself. During the session we took note of actions which appeared to illustrate her expertise. We also asked questions pertaining to the reasons she did certain things in a particular fashion. It is interesting to note that quite often she was not immediately able to justify her acts. While this is not entirely uncommon in expert interviews, it became evident that such a response was related to the nature of her field. It is generally understood that experts, who have been performing their tasks for many years, tend to work without consciously retrieving the rules of their trade. Advanced beginners, on the other hand, are quite adept at freely accessing various rules in support of their actions.

Thus Musgrave, who is an accomplished designer, normally worked "without thinking," and found it difficult to supply us with information at times. One must bear in mind, however, that these are not acts of intuition, as they may appear. Rather, they are the result of an expert performing her task for many years, thus liberating her from actively retrieving guidelines as would a novice.

It is also important to note that while justification for her acts did not always come freely, it was our task as observers to carefully interrogate and extract this knowledge from her. It did turn out, in fact, that upon introspection she was able to give specific explanations for virtually everything she did. Whether or not these explanations were



actually valid is another story. We generally attempted to prompt elaboration on the responses that were not fully convincing, but this was not always successful. The importance of using a number of different knowledge acquisition methods is illustrated by such information gaps.



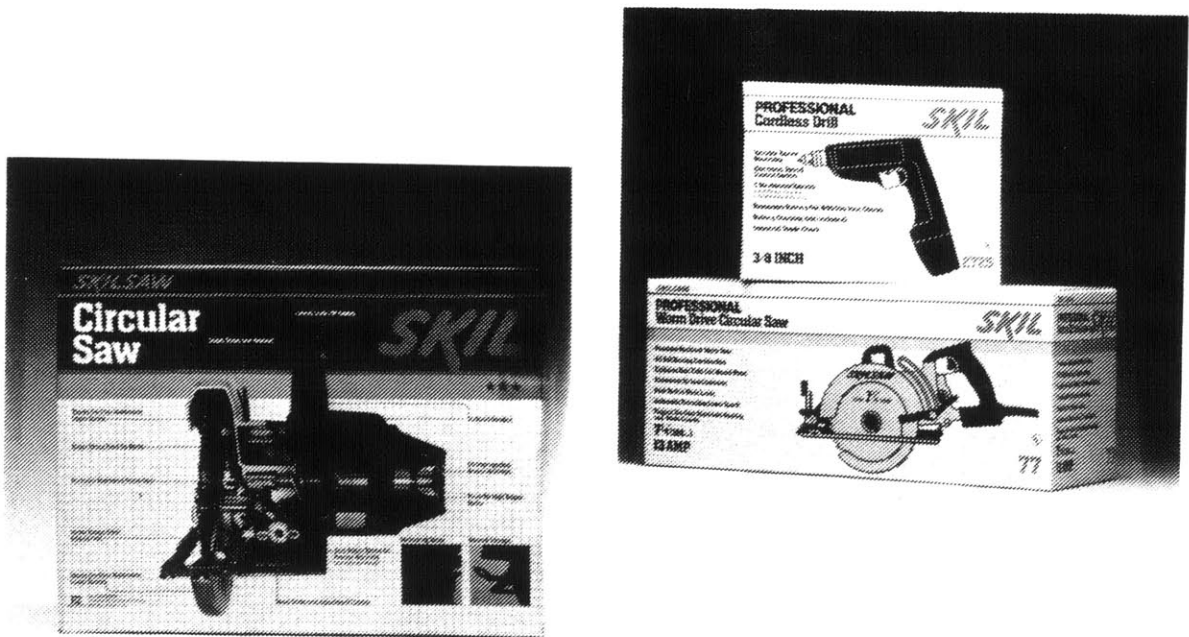
**Figure 5.1** *A sample layout designed in the knowledge acquisition session.*

### **Comparison of Commercial Packages**

One of the most revealing methods by which information was acquired for PackIT's knowledge base was by a thorough analysis and comparison of packages recently on the market. The source was a collection of packaging design books with full-color photographs of products (Japan Creators' Association, 1985; Japan Package Design

Association, 1985; Reese, 1986; Schmitt, 1985). The similarities and differences between individual members of product lines were clearly evident in these books, thus enabling me to formulate many packaging design rules. While one might argue that this is not a first-hand source of information, it did force me to analyze the relationships between packages much more fully since rules were not merely presented to me by an expert.

Figure 5.2 is an example of the family relationships displayed in one of these packaging books. These two photographs compare power tools meant for the professional and the average consumer. The professional package has a simple drawing of the product with subdued colors, while the other one displays a color photograph. Even though the designs are unique, the overall company identity is clearly identifiable.



**Figure 5.2** *A comparison between packages in normal and professional markets.*

### **Corporate Design Standards**

Specific guidelines and strict rules were conveniently provided by a number of corporate ID manuals, including those of IBM, Citibank, Honeywell, and others. These manuals are typically utilized by designers within a firm in order to clarify the various design criteria of that firm. Rules such as *the company logo should never appear anywhere but as far right as possible*, are quite unambiguous, and may often be entered into the knowledge base with only slight modification.

### **Class Experiments**

The *Computer Graphics Workshop* is a seminar offered at the VLW which in the recent past has devoted much of its subject matter toward the study of graphics in knowledge-based environments. One particular assignment required pairs of students to design business cards for a hypothetical company. One member was to observe his partner (with little verbal interaction) while creating cards on an Apple Macintosh using *MacDraw* software, after which the roles were reversed. The observer's task was to note his partner's design techniques with regard to what was done, while also attempting to infer why such techniques were employed.

This proved to be a rather illuminating experiment since it forced both designer and observer to think more consciously about their work. Many of the students employed similar techniques in producing their work, which seemed to support the idea that there are formalities and conventions in the design process.

### **User feedback**

The final means by which the knowledge base of PackIT was developed and revised was through feedback of people who have tested or observed the system. These users included a number of people, ranging from non-designers to experienced graphic artists. Most reactions were positive during development, and many were sources of suggestion for further improvements I have since incorporated. One such suggestion was for PackIT to display the layouts it has rejected during the inference process as being unacceptable. PackIT runs by creating a basic layout, then testing to see if the layout breaks any rules, and redoing it if necessary until an acceptable composition is made. By displaying each of these rejects, a graphical trace of the inference is made, thus revealing the system's internal processes and permitting the user to become more of a participant in the package's design.

In conclusion, it is clear that the process of knowledge acquisition is critical in the development of an accurate expert system, especially one in the design domain. It is also evident that observation is highly important here. The methods discussed above have proven to be of great help in the development of PackIT, and those who are attempting to develop such a knowledge base should most assuredly employ similar procedures.

*Rules For  
Design Expert Systems*

**Techniques of Art and Design as Rules**

In order to develop a knowledge base for a design expert system, one must study and incorporate some of the standard rules of design. This section describes a number of techniques of art and design which may be formalized and entered into a rule base. PackIT utilizes some of these rules, in addition to specific packaging techniques.

As previously discussed, the expertise exhibited by artists and designers must be formalized before entering it into a knowledge base. Joan and Russell Kirsch (in a forum at MIT's Media Laboratory, March 1987) suggested that when we are able to recognize a painting as being the work of a specific artist, we are aware of the visual "syntactic qualities" inherent in that artist's work. They propose "syntax as a formal grammar" for describing an artist's use of color, lines, mood, etc. This is a means by which an artist's style and technique may be formalized.

Their initial work dealt simply with lines and the relationships between lines, and was based on the study of paintings by Richard Diebenkorn. The grammar takes the form of production rules. A generalized example is,

IF: A large, empty, rectangular region exists on the canvas,

THEN: It may be divided into sub-regions.

It was important for them to extract the qualities which characterize Diebenkorn's

works and describe how they were used.

A vast number of descriptors may characterize any composition. The following composition attributes may be useful to the designer:

<i><u>Contrast</u></i>	<i><u>Harmony</u></i>
Asymmetry	Symmetry
Boldness	Subtlety
Angularity	Roundness
Economy	Intricacy
Verticality	Horizontalness
Depth	Flatness
Fragmentation	Unity
Exaggeration	Understatement

(Dondis 1973, 16)

While these concepts might initially appear to be too abstract to be entered into a formal rule base, they may be broken down into entities which are more computable. In terms of a rule-based system, they may be thought of as the attribute values of an object. For example:

```
(package-1 :weight) = ASYMMETRIC,  
(package-2 :cohesion) = FRAGMENTED.
```

These simple lines of code state that the weight of package-1 is unordered or unbalanced, and the objects of package-2 are not unified. Thus two vague concepts have been formalized as attributes of the graphic objects, and may be computed through analysis of the objects' placements and relationships, respectively. The knowledge engineer must be aware of what makes a layout "fragmented", "bold", or "angular" so that rules dealing with these characteristics may be encoded in the knowledge base.

An even simpler example from this list deals with the orientation of a composition. A package may be declared horizontally-oriented if its objects are laid out side-by-side rather than above or below each other. Package orientation is, in fact, one of PackIT's fundamental concepts, and will be discussed in depth later in this thesis.

Some of the more basic areas of visual study and design include:

- color
- balance
- contrast
- proximity

I will now present brief descriptions of how some of these factors are used by designers, and how they may be incorporated into a design expert system.

### **Color**

In my study and comparison of existing packaging families, I have noted that color is commonly used to distinguish individual members within families. The use of particular colors for specific situations helps the designer to project the product image he so desires.

Sacharow describes the emotions that certain colors may evoke in us:

*Red* is a powerful color because it stimulates the digestive system and the circulation of the blood, arouses the forces of self-preservation, and signifies strength and virility.

*Orange*, more subtle than red, is often used on packages associated with the physical because it expresses action.

*Yellow* denotes light gaiety and warmth, and it is cheerful and bright.

*Green*, quiet and refreshing, is associated with youth, growth, and hope.

*Blue* is cool and subdued.

(Sacharow 1982, 47-48)

He also presents more specific examples: containers of furniture polish are usually shades of brown, while milk products often come in dark blue, pale blue, or white cartons.

In addition, colors are perhaps the most widely-used means of differentiating between different packages in a product line; it is quite common to see packages within a line display different discriminating colors. At the same time each of them usually retain certain graphical elements of the same color for brand identification. Use of color in packaging also requires the additional consideration of production expenses.

Kepes describes the effects which result from the placement of certain colors next to each other.

Hue, brightness, and saturation of a surface are modified by the adjacent surfaces. The contrast effect is always in the direction of the greatest opposition of colors. If a red and a green surface are juxtaposed on the same picture surface, the red appears redder than it would be if viewed in a color background of closer hue. Similarly, green appears greener when viewed in a yellow, a blue or a brown background. (Kepes 1961, 139)

The human visual system is able to distinguish between a great number of shades of a particular color. In graphic arts, however, the more commonly used gray scales have a small number of steps, which are distinguishable by us. This small quantity can be greatly augmented when techniques of juxtaposition are applied. A particular color may appear relatively light when placed next to a darker color, and dark when placed next to a light one (Dondis 1973, 48).

The use of color saturation is another important technique for projecting particular package identities:

Saturated color is simple, almost primitive, and always given preference by folk artists and children....The less saturated colors reach toward neutrality of color, even noncolor,



and are subtle and restful. The more intense or saturated the coloration of a visual object or event, the more highly charged it is with expression and emotion....saturation or its absence as a meaningful visual effect is the difference between the dentist's office and the Electric Circus. (Dondis 1973, 51)

PackIT has utilities which calculate the brightness and saturation of graphic objects by first calculating those of each individual point on the object, and then taking the average of all these values. (These points are small dots of light called *pixels*, which make up the computer screen.) It does this with the use of the following formulas:

$$\text{Brightness} = (R * 0.3) + (G * 0.59) + (B * 0.11),$$

$$\text{Saturation} = (\text{MAX}(R, G, B) - \text{MIN}(R, G, B)) / \text{MAX}(R, G, B),$$

where R, G and B specify the strengths of the pixel's red, green and blue signals respectively. Once the inference engine has these values (which are held as part of each instance of the *graphic-object* class, described in appendix D) it is able to more accurately make future decisions about placement, color choice, etc.

### **Balance**

Various arrangements of objects on a composition are widely used by designers in order to project different qualities. In general, a design whose components are balanced or symmetrically placed on a page tend to emit the qualities of stability or composure. Unbalanced compositions tend to be somewhat more active or stressful. Paul Rand states that this feeling of unbalanced activity is often quite effective and desirable.

With asymmetric balance, he is able to achieve greater reader interest. Bilateral symmetry offers the spectator too simple and too obvious a statement. It offers him little or no intellectual pleasure, no challenge. For the pleasure derived from observing asymmetric arrangements lies partly in overcoming resistances which, consciously or

not, the spectator has in his own mind, thus acquiring some sort of esthetic satisfaction.  
(Rand 1970, 80)

While the balance of a composition is not specifically considered by PackIT, the stability described by Rand could be controlled by analyzing the positional relationships of the objects in order to determine the placement of weight on a package.

### **Contrast**

The term “contrast” is quite general, covering the areas of color, texture, shape, etc. Contrast of objects is used not only to alter perception by forming new relationships between objects, but also to alter the objects themselves. Why do we don summertime clothing in Early Spring when the temperature rises to 55 degrees? Why does the same temperature seem extraordinarily cold in the early Autumn? We generally do not experience stimuli as distinct and absolute entities. As Dondis states:

Contrast is the sharpener of all meaning; it is the basic definer of ideas. There is so much richer an understanding of happiness when it is thought of juxtaposed with sadness; love with hate; affection with hostility; motivation with passivity; involvement with loneliness. (Dondis 1973, 96)

In the packaging design world, many companies forbid the placement of a graphic object near their logotype if they both have similar visual qualities. This helps to reduce the blending in of the logo into its surroundings, and makes it more forceful.

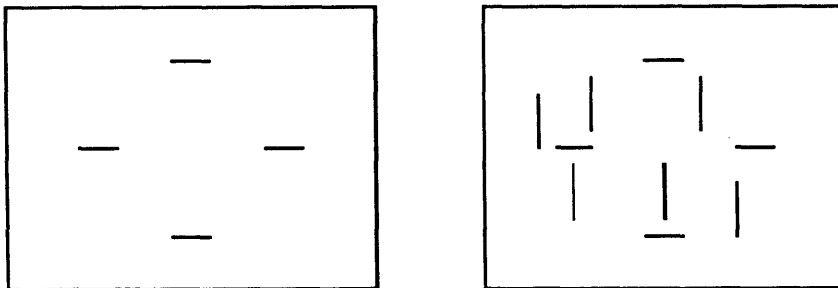
### Proximity

The importance of proximity is illustrated quite often in corporate ID manuals where rules commonly state strict requirements, such as:

*The company logo may not appear in other than a dominant position.*

*The motto must always appear centered at pagebottom.*

The ancestry of such rules is clearly documented in texts of visual design. One such usable set of rules states that various objects establishing a powerful unity due to their proximity may be broken up through the placement of similar objects at a distance; while units formed by similarity may be broken up by close proximity of dissimilar objects (Kepes 1961, 47).



**Figure 6.1**

*The first box shows four similar objects at a distance from each other forming one unit. The second box illustrates how this unity can be broken up by the nearby placement of objects of a different nature.*

---

One of the fundamental concepts upon which PackIT is based is the notion of positional relationships between graphic objects. The position of an object on a package is specified, not in absolute terms, but in relation to various objects surrounding it.

Such well-defined kinships permit a number of interesting design rules to come into play which deal with intensity conflicts, alignment restrictions, etc. These will be fully discussed later.

### **Packaging Rules Extracted From Texts and Manuals**

My primary sources for acquiring specific rules for packaging have been corporate ID manuals and packaging design books with illustrative photographs. The two, although quite different in approach, have both been extremely helpful in the knowledge acquisition process. They differ in that the packaging books present photographs of final designs with little commentary. This forced me to concentrate fully on the analysis of what gives a certain package its own unique flavor. Illustrations of packaging families were remarkably useful in this respect. The ID manuals, on the other hand, left no room for ambiguity – they state the design techniques that are permitted, and those which are strictly forbidden. That is not to say that those forbidden reflect poor design, but simply that (for whatever reason) they are inappropriate for that particular company.

This section is appropriately divided into two sub-sections – one dealing with rules extracted from the packaging books, and the other concerning the ID manuals.

#### **Packaging Book Rules**

Perhaps the most common means by which packages in a product line are made distinguishable from each other is through the use of color. Most package designers seem to agree that it is appropriate to put, for example, a company's *minty-fresh* version of their toothpaste in a green *minty-fresh* box. This notion also holds for companies

which use color as a means of identifying their products in general – the orange and red packages of the Kodak company, for example, are immediately recognizable by us.

Not only is color used as an aid to specify what type of product is contained within a package, but it is also used to indicate more subtle characteristics such as level of quality. In certain circumstances companies may use more subdued colors to put forth a feeling of quality, for example.

In packaging there is another color consideration relating to product quality, namely production cost. The producers are subject to price constraints and so, naturally, multi-colored attractive packages are not always desirable. The most obvious example is the so-called “generic” line of products, where colors (and graphics in general) are used minimally.

A second source of production rules for PackIt deals with position of individual graphic objects. Clearly this is just as important here as the use of color. Situations where this is a factor include product lines, production of different sized containers of the same product, use of containers with varying material types, etc. The designer must determine where graphic objects are to be placed on the package under these differing circumstances, while still retaining the company’s visual identity. For example, the introduction of a smaller *handy trial size* of an existing product may initially prompt the designer to shift the placement of the company’s name on the new box. It must be considered, however, that this may alter the traditional layout (and consequently the identity) found in the company’s other products.

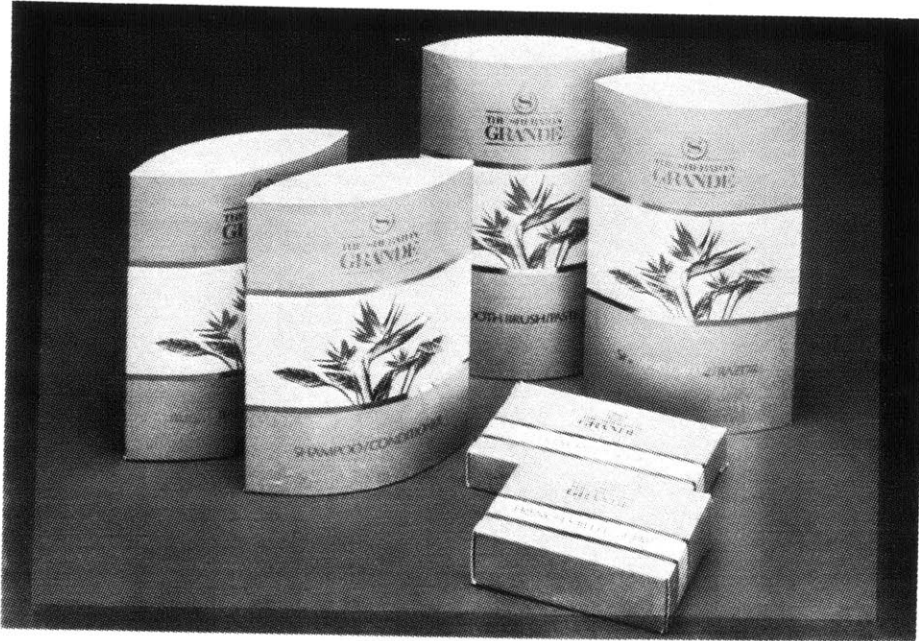
Fortunately the designer's options are numerous at this point. Rather than shift the position of an object, it is also possible to resize or crop it.



Figure 6.2

Note in figure 6.2 that the quart, half-gallon, and gallon milk containers all display the same picture cropped differently.

It is also acceptable to remove particular objects altogether, under certain well-defined conditions, as shown in figure 6.3.



**Figure 6.3**

These types of decisions are handled quite well by PackIT, and there are also rules stating when and by how much certain objects may be resized or cropped. The handling of these decisions is one of PackIT's major features and will be discussed later.

In addition to color, the clever use of typefaces can also be used to project certain feelings. A slim, sans serif typeface is ideal for products such as microwave dinners where the producer would like to evoke a clean, modern feeling. Or perhaps a highly ornate typeface can be used for the more "elegant" gourmet dinners.

Even animals are concerned with the use of graphics and typography. The space-age typeface illustrated in figure 6.4 is clearly aimed at the contemporary hound, who is ready to take on the future fueled by this nutritious nuclear chow.

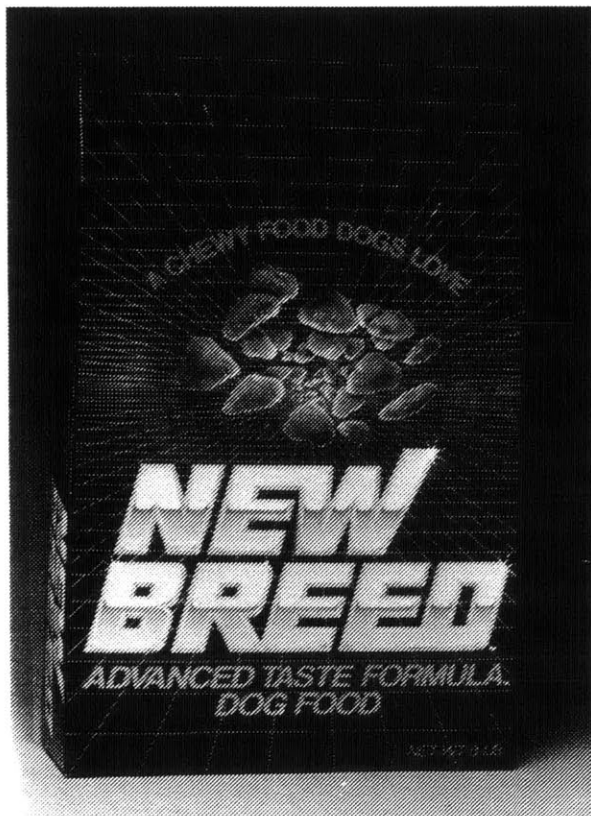


Figure 6.4



That which I refer to as “auxilliary graphics” is also used to more accurately characterize products. *Low-Calorie* or *diet* products commonly exhibit thin, vertical lines on their containers in order to represent lightness and slimness.



Figure 6.5

Sloping lines tend to project a feeling of activity. This is also evident in the designs of the Coca-Cola Company (clearly a marketing giant). Rather than reusing the solid background color on their *Classic Coke* cans, they choose thin diagonal stripes for cans of their new *Diet Coke*.

There is, in addition, an entire class of rules related to more strict standards or requirements. A decision which has become a standard rule for many cereal producers is to place the product’s weight at the bottom of the box in a simple typeface. This is surely not required, but the consumer expects it to be as such and so it is rarely found otherwise.

To take this one step further, one may consider rules which are required and

unbreakable, such as those imposed by governmental regulation. The Surgeon General's warning on cigarette packages immediately comes to mind. Also, the use of the UPC symbol is a commercial standard found on virtually all packages.

The purpose of this discussion was to present some of the guidelines that I have observed during my analysis of contemporary packages. Note, however, that these are but a few examples of my entire compilation which is, in turn, a minute sampling of all packaging design rules currently in use.

#### **Corporate Identification Manual Rules**

The information I acquired through the corporate ID manuals was much more directly presented than that previously discussed – these manuals clearly state what designers for a particular company may and may not do. Most of them cover the design of letterheads, envelopes, business cards, and the like. Some even extend to advertising, signs for corporate offices, and graphics displayed on company vehicles. Given the specificity and strictness of the standards set in these manuals, it is clearly evident that rule-based expert systems could be particularly useful as aids for overall corporate design in addition to packaging.

I will present below some of the rules I have extracted from the ID manuals of IBM, Citibank, and Honeywell.

IBM

- *Text of lesser importance should not draw attention away from text of greater importance.*

In this manual, IBM expresses their desire to refrain from drawing attention away from the selling point, even if it comes at the expense of their logotype. This is illustrated in figure 6.6 where the IBM name has been reduced from four occurrences to two, with the name at the bottom deemphasized so as not to detract from the title.

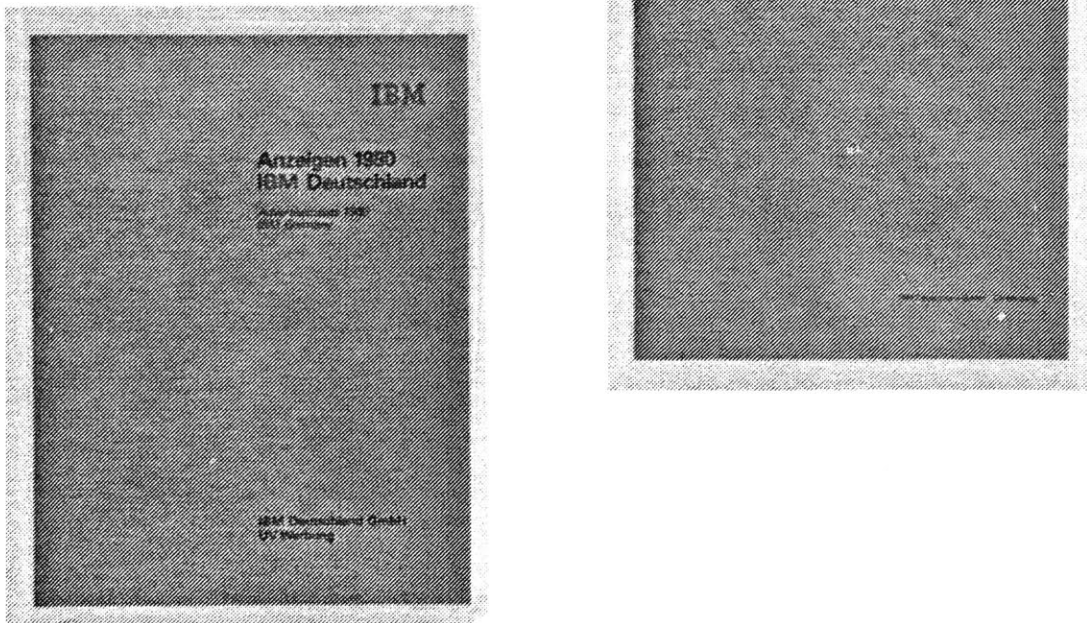


Figure 6.6

- *The IBM logotype may be presented vertically when useful.*

This is an example of how the dominance of one object (the logo) may be reduced, while still keeping it present. In this case, the flow and readability of the other text is retained.

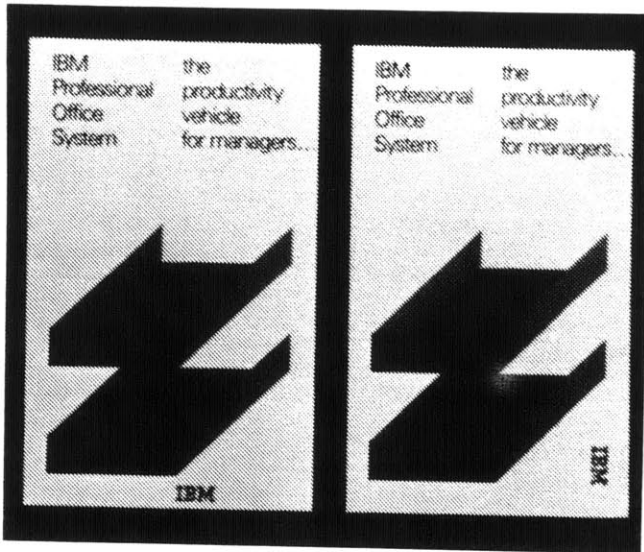


Figure 6.7

- *The alteration of the logo is acceptable when used as a selling aid.*



Figure 6.8

Variations on the basic logo design may be made in order to produce a dynamic effect. The cover of this manual, in fact, shows a number of IBM logos rotated at random in various colors.

*Question:* Does altering the logotype tend to reduce the company's visibility, since the consumer expects to find certain graphic elements constant throughout the company's products?

*Answer:* Perhaps. But it also might make the design more unexpected and exciting, and therefore more visible.

*Note:* Although IBM is commonly thought of as projecting a relatively consistent and conservative image, this manual seems to promote many interesting techniques which are not quite as conventional as those of other large corporations.

### Citibank

- *Company name always appears "as far to the right as possible" on a given page.*

The Citibank manual stresses that these rules apply in exactly the same way in all other media.

- *"A Citicorp Company" may not appear in type smaller than 9 point.*
- *Company name may not be displayed other than horizontally.*

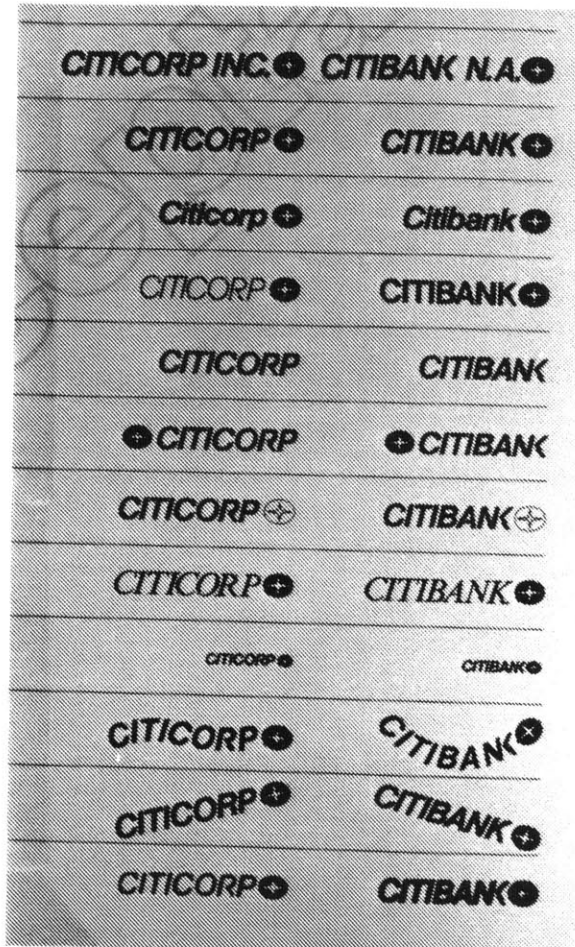


Figure 6.9 Examples of unacceptable Citibank logo designs.

Honeywell

- Company name may never be displayed with anything else directly adjacent to it.
- No overpowering objects should appear in the same area as the company name.

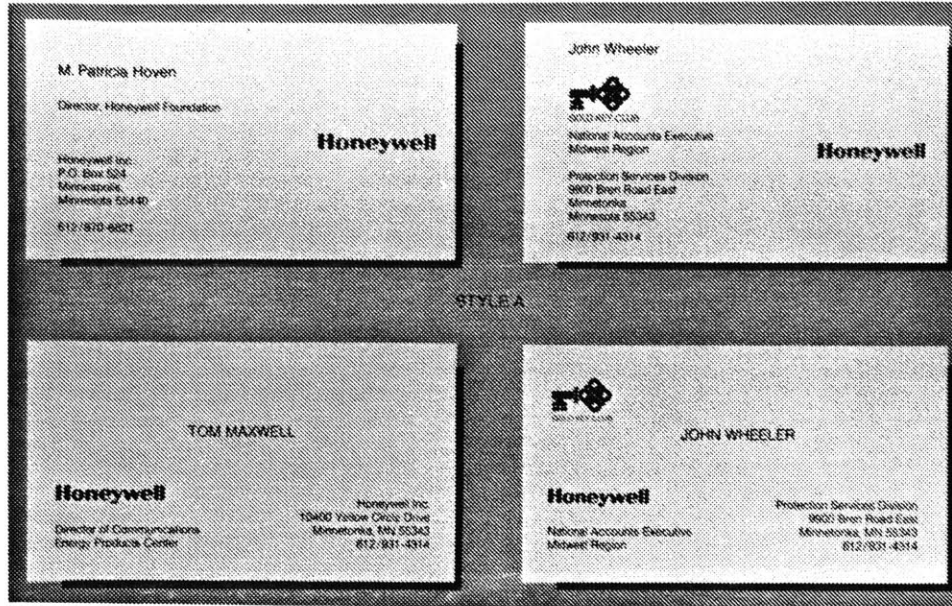


Figure 6.10

These rules are illustrated in the business cards of figure 6.10 where the Honeywell logotype is either positioned by itself or near to more subtle text.

Although most of the rules stated in this section are based on underlying rules of design, it should be noted that they are specific to particular companies and therefore do not always reflect the standards of graphic design as a whole. The knowledge base of PackIT provides a kind of case study, where some of these company-specific rules have been employed.

CHAPTER 7

---

*How Strict are the  
Rules of Design?*

While this paper has proposed the use of packaging and graphic design rules, there are, in fact, some classes of design technique that are less strictly rule-based. Some objections to the use of strict rules stem from the monotony resulting from constant adherence to such rules, and others simply concern the restraints of stylistic freedom. This chapter discusses these criticisms and attempts to present some potential solutions.

A few months ago, I came across a particular book entitled Forget all the rules you ever learned about graphic design (1981). At once I knew I had found a person who would probably question the validity of my work. In this book the author, Bob Gill, discusses the problem of originality by describing a job he once had for designing the title card of a television series.

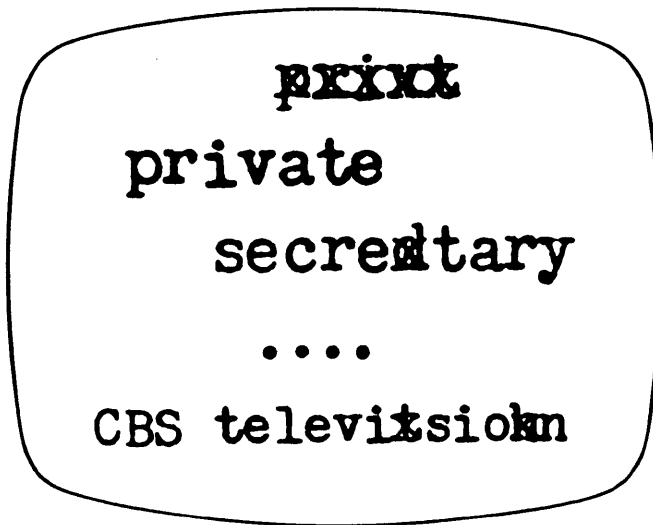


Figure 7.1



“I wanted to do something that was original. But I kept thinking of ideas based on images I had already seen. Then I realized that it was inevitable that my ideas *had* to be based on previous experiences. What else could possibly be in my consciousness *but* previous experiences?” (Gill 1981, 7) It was at this point he realized that redefining the design problem in a unique manner would result in a successful, exciting design.

The book, in general, stresses the importance of new ideas. The designer’s pitfall is said to be conformity – if one has a unique solution, take it as far as it will go.

But despite the title of his book, Gill is most likely not condemning the use of design rules. In fact, the text is filled with rules; the title itself is a rule! What he really states is that the approach to design is crucial and that graphic compositions should grab the viewer’s attention.

Fortunately, interesting designs can be attained in a most formal (rule-based) manner. One technique involves the balance of a composition. At times bilateral symmetry is too simple and obvious, and offers little intellectual pleasure or challenge. Asymmetry often incites much more viewer interest. (Rand 1970, 80) Balance (in terms of weight, color, shape, etc.) is quite easily handled in PackIT with the use of such things as the rules of position, and the image brightness/saturation facilities previously mentioned.

Dondis explains the importance of stimulating the viewer. “Boredom is as much a threat in visual design as it is elsewhere in art and communication. The mind and the eye demand stimulation and surprise, and an approach to design that functions boldly and successfully suggests the need for sharpening of the structure and message.” (Dondis 1973, 93-94) She describes how Rembrandt, in his *chiaroscuro* technique, overcame this. In many of his works, Rembrandt played with lightness and darkness by diminishing his use of middle tones, thus resulting in a painting of greater contrast and a

striking “theatrical look.” The removal of light where we normally expect it in daily life (from above us), and replacement behind or to the side of the painting’s subject also creates a “dynamic spatial effect.” (Kepes 1961, 145)

Although the question of a computer’s ability to actually “create” new design (even in the most lauded “intelligent” AI systems) is certainly a valid one, it is at least possible for the computer to compose interesting works by employing rules like those above. It is not acceptable to simply discard the rules of design for the sake of innovative creativity. Graphic objects in a layout cannot simply be pushed around on a page until a pleasant design is generated.

An erroneous conception of the graphic designer’s function is to imagine that in order to produce a “good layout” all he need do is make a pleasing arrangement of miscellaneous elements. What is implied is that this may be accomplished simply by pushing these elements around, until something happens. At best, this procedure involves the time-consuming uncertainties of trial and error, and at worst, and indifference to plan, order or discipline. (Rand 1970, 11)

A knowledge-based system that produced in this semi-random fashion would either:

- a) have a tiny knowledge base (since objects are randomly arranged) and would require a human to rerun it hundreds of times until an acceptable layout were created.
- b) not require a human to judge, by instead having a huge knowledge base which could judge the final result itself (in which case, the judging knowledge might as well be employed in the layout routines in the first place to produce an acceptable result in one iteration).

At any rate, it seems that the more rules are employed which govern a process, the more fit computers are to solve it. At first, this may appear to be so because less rules in the process means more “intuition” or “natural creativity” is required. This may be why

domains with many strict rules (such as geology and medicine) have produced the first and most widely-used expert systems. However, as I stated in an earlier section, “design intuition” is not as mysterious as one might think. It is based on knowledge and experience – the very things that expert systems handle so well.

In conclusion, I believe that by utilizing the techniques of viewer stimulation described above, computers can effectively handle designs which are to be exciting or pleasant or even boring! (At times certain graphic elements should be boring; specifically when attention is not to be grabbed away from other important aspects of the composition [Gill 1981, 13].) And even if Gill objects to the use of formal rules in design, he does, in fact, use them himself.

### **When a Designer Chooses to Break a Rule**

There are certain instances in the design process when it becomes necessary to outstep conventional bounds. Perhaps an idea is inexpressible within these constraints; maybe a striking, unique look is required. Whatever the reason may be, the breaking of rules is clearly more common in design than in most other fields. Human experts are quite capable of deciding when such actions are appropriate. How may this situation be handled in a knowledge-based system?

Before attempting to tackle this problem, let us consider some specific situations where rules may be broken with impunity. An example may be found in the Citibank corporate ID manual. As previously mentioned, Citibank makes it quite clear to their design department that the company’s logotype must never appear in any orientation other

than a horizontal one. Yet on page 54 one finds a photograph of a Citibank tie proudly displaying a number of their logos in a diagonal formation. Evidently, even the seemingly most strict rules may be overruled at times.



**Figure 7.2**

While one might counter that a tie is in a different class than the other subjects of this manual and therefore requires different rules, it is also true that the division between classes is not always so distinct. Furthermore, humans are generally able to judge when it is appropriate to override rules.

In order to determine whether design expert systems can handle such situations, we must analyze exactly what conditions exist when rules are broken in the first place. In these situations one may conclude:

*A designer is aware of the appropriate times to break rules – computers cannot do this, therefore expert systems cannot adequately handle design knowledge.*

This may not be the case, in fact. What if we conclude the following instead:

*When a design rule is broken,*

- a) the rule was not valid to begin with and should be discarded.*
- b) the rule was once valid, but is now obsolete and should be discarded.*
- c) the rule may be valid under different conditions.*
- d) there are good reasons for breaking the rule. Formalize these conditions and enter them into the knowledge base.*

Situations *a* and *b* may be quite common in expert systems, and are also relatively easy to handle. If a rule is overridden by the user often enough, it is not difficult to simply update the rule or discard it altogether. Stepping through the rule base as such and filtering out those which are deemed invalid is a debugging process. If the system has good justification/explanation facilities, these rules can be pin-pointed relatively quickly.

Situation *c* states that certain rules may be valid only under certain circumstances. These conditions should be added to the rules themselves. Situation *d* also provides a simple solution. If a rule is still valid but should be overridden under certain conditions, why not simply formalize these conditions? Entering this newly acquired information will make the knowledge base that much more powerful.

There are still other means to manage the problem of dealing with a rule base which needs to be updated. The alteration of a rule's certainty factor is one common solution. A much more complex solution is the incorporation of metaknowledge – programming the system to know when to alter its rule base, for example. This might be done if the system updates its own database which holds information of the firing or the user-induced bypassing of rules. If a rule is rarely fired or commonly violated, then an update should be made.

Lenat et al. (1983, 229) describe a chemical spill management rule that could never be fired, and a metarule which could automatically detect it:

*FAULTY RULE*

IF: Acid must be neutralized, and lye is available,  
and no basic material is available,

THEN: Use lye.

*METARULE*

IF: Over the course of many runs, a rule never fires,

THEN: Ask the expert if the condition is for some (semantic)  
reason unsatisfiable, and request that it be rephrased.

The metarule above is be able to detect a rule which was never fired due to faulty logic. A similar rule might be used to detect other rules which (because of taste) a graphic designer has chosen to override many times.

Overall, it appears that the breaking of rules in order to achieve more unique and interesting compositions does not present great problems for design systems, providing there is at least minimal human intervention in the process (as in PackIT). Furthermore, the dilemma is substantially reduced when the intervention of the user increases, as one would expect.

---

*PackIT's Rules*

This section describes some of the rules found in the knowledge base of PackIT which represent the design specifications for a hypothetical company. These rules may be broken down into the following categories:

- color
- object importance
- cropping
- repositioning
- margin size
- resizing

**Rules of color**

I have previously discussed color as a useful means by which individual members of a product line may remain distinguishable, or by which entire lines may remain identifiable as being that of a particular company. PackIT uses color to differentiate packages with different marketing areas and projected images:

IF: Marketing area is professional,  
THEN: Set background color to medium gray.

IF: Projected image is economy,  
THEN: Set background color to white.

The first rule specifies the package color to be used for professional products (e.g. tools for carpenters, electricians, mechanics, etc.). Such products are usually of somewhat

higher quality than their counterparts (which are meant for normal usage) and tend to have more subtle or refined packages. The second rule sets the color for a low-cost product – white containers are less costly to produce. Note that these simple rules are particular to the hypothetical company represented by PackIT; a professional-style package might just as well be something other than gray for a different company.

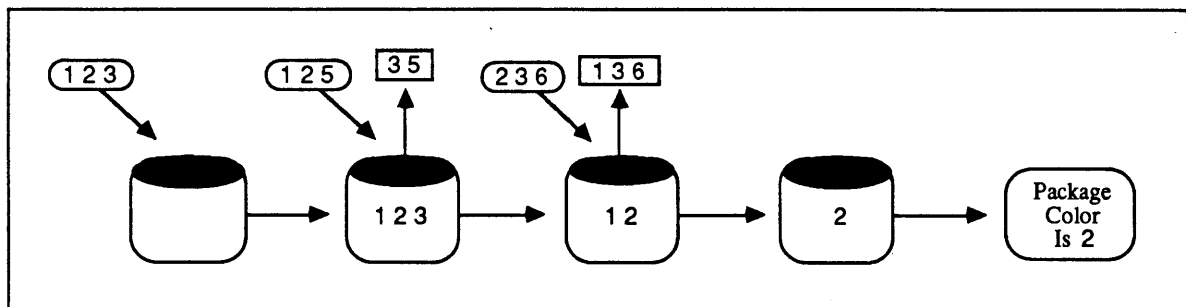


**Figure 8.1** *The use of colors on “generic” packages is always kept to a minimum.*

In addition, both of these rules would have to be more intricate in order to handle other conditions that affect color. They are both quite inflexible and seem to rule out any further change. What if another rule suggested using a different color given the present



conditions? Which would take precedence? Some systems might assign a certainty factor to these assertions so that the more dominant one would prevail. PackIT, on the other hand, assigns colors by accessing a *color bucket*. Instead of assigning one particular color, it presents all the acceptable colors (under the current circumstances) to this bucket, and then checks to see if any of the new entries match those already in the bucket. If there are two or more colors in the final bucket, the first in the list will be chosen. Figure 8.2 illustrates this filtering process.



**Figure 8.2**

*A package color is chosen by intersecting the currently suggested colors (as determined by the active rule) with the colors previously determined to be acceptable. Colors 1, 2, and 3 are initially entered into the bucket. A second rule suggests 1, 2, and 5, with 3 and 5 immediately being discarded since they do not have a match in both groups. 2, 3, and 6 are suggested by a third rule leaving only 2 as the final package color.*

### **Rules dealing with object importance levels**

An importance factor is associated with each graphic object that may appear on the package. This indicator, which ranges from 0.0 to 1.0, specifies how important it is for the final composition to include the object. A particular object may be necessary in one instance, and expendible (or even unwanted) in another. If it is decided that due to problems of space, for example, an object must be removed if possible, these importance factors will be accessed.

IF: Product picture appears on the package,  
THEN: Lower importance factor of product description text.

IF: Marketing area is mass-market,  
THEN: Raise the importance factor of the product description.

Many packages have text describing their products. The first rule states that if there is a picture of the product on its package, then some of this text may be less vital and may be removed or reduced in size. Descriptions of product color or style are often unnecessary if a picture provided. The second rule states that if the product is being mass-marketed, then the text describing its features are important. This is so because such products are often meant to induce quick sales. If information describing the benefits of a product is easily accessible by the consumer, then he will be more likely to make a purchase. It also reduces the need for a salesman to explain the product. (See figure 8.3.)

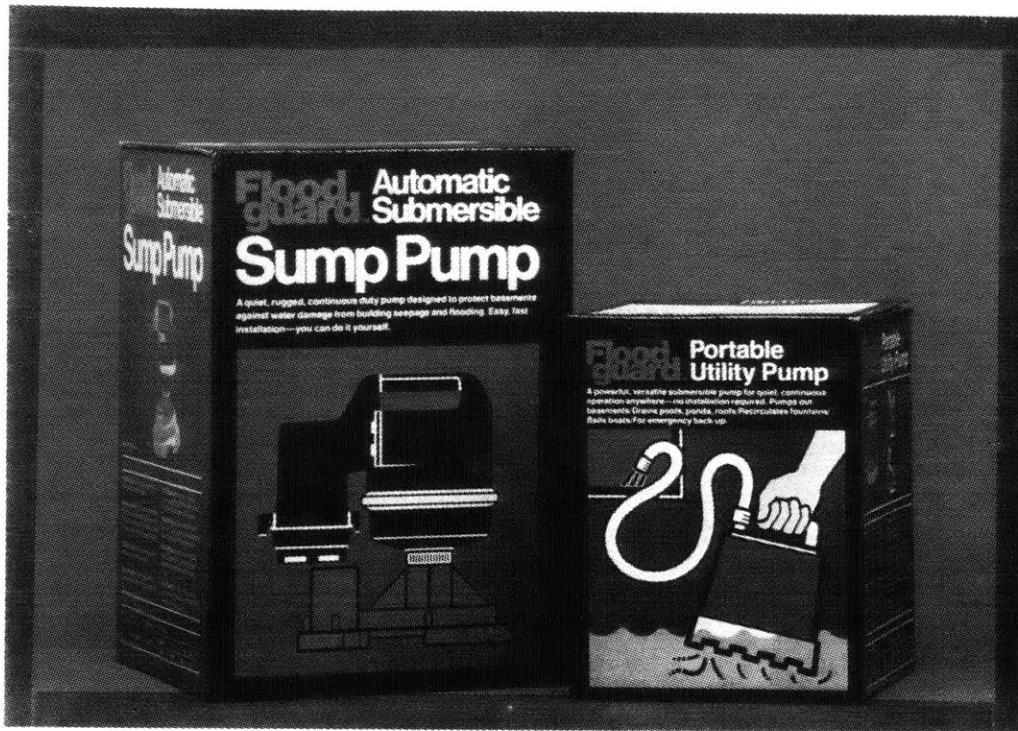


Figure 8.3 *This package with its bright colors and descriptive text was meant to grab attention away from its competitors, and reduce the need for a salesman.*

At times certain graphic objects may be removed altogether:

IF: Marketing area is industrial,

THEN: Remove the color photograph of the product.

Many products, such as medical equipment, are not meant to induce quick sales, and so expensive and colorful graphics are not needed. The rule above states that a full color photograph may not be necessary for some package types; another rule might substitute a simple illustration in its place (see figure 8.4).



**Figure 8.4** *Colorful photos have been replaced by a simple illustration for this medical product.*

### **Rules dealing with cropping**

One of the most powerful features of PackIT is its ability to determine and remedy situations where the shape of the package is being altered and the graphic objects no longer fit. This was illustrated in the case study described in chapter 4. If a package is shaped in such a way that the objects can not all fit satisfactorily, some of them may be cropped. The rules specifying which are cropable are strict. For example, it may be acceptable to crop the outdoor scene of a mountain with adjoining fields, but it may not be desirable to crop too much of a human figure.

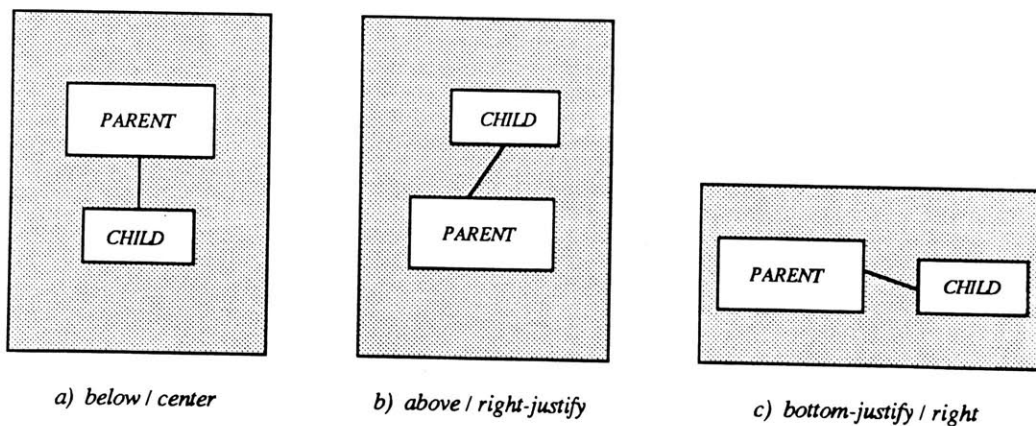
### Rules dealing with repositioning

Another way to adjust an object which does not fit well is to shift its position.

IF: An object can not fit vertically,

THEN: Shift it upward and to the side of its parent object.

PackIT's database holds information describing object position in a hierarchical tree structure. The position of each object is specified not in absolute terms, but in relation to its parent and children. A child may found *centered below* its parent, for example (Figure 8.5). Note that although a child may be *physically* positioned above its parent, it is still found below the parent in the hierarchy.



**Figure 8.5**

*Relationships between graphic objects. The objects are arranged in a tree structure with relationships between them specified by verticle/horizontal pairs. Example a shows the child centered below its parent. The child in example b is above and right-justified with its parent (note that parents do not have to appear physically above their children). The child in c is bottom-justified to the right of its parent.*

The rule above says that if an object cannot fit vertically, then it may be moved up to another acceptable area. PackIT automatically removes it from its position in the tree and

reattaches it in a nearby area where there is room. This move should not alter the nature of the design too drastically, and so the object will be reattached to its parent in some other manner. If there is nowhere else to fit the object, the other rules will attempt to tackle the problem.

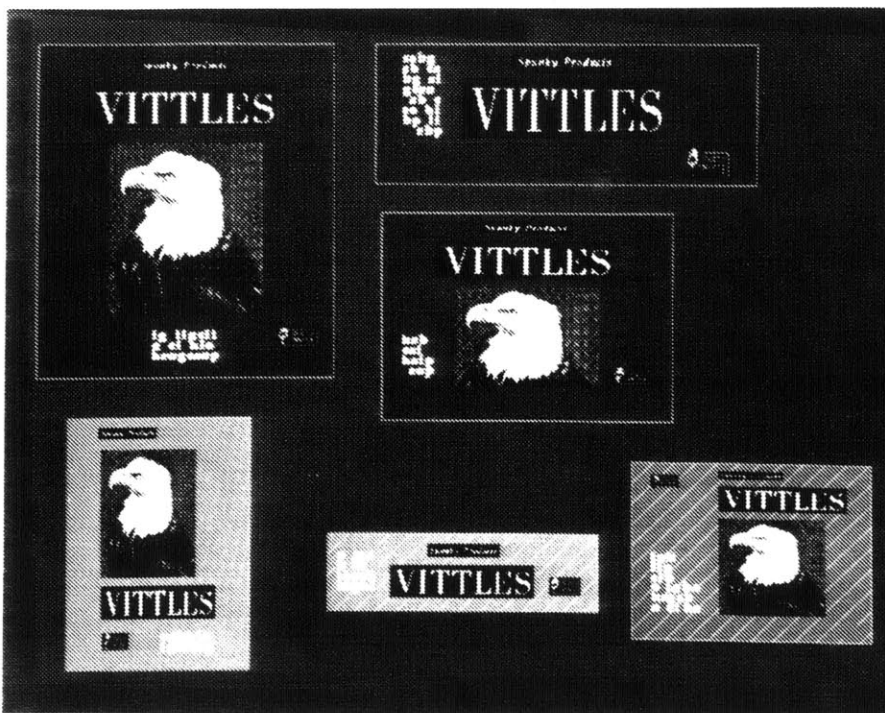


Figure 8.6 *PackIT* assisted in the generation of these packages using the rules of packaging and graphic design.

### Other rules

There are additional rules which assist in the arrangement of graphic objects when there is too little or too much space in the composition. There is a pre-set margin size specifying the minimum permissible distance between container the edge and the

outermost objects, as well as one for distance between objects themselves. These sizes may be altered by rules in order to compensate for space limitations. Another set of rules deals with the resizing of individual objects (while keeping their dimensions constant). Certain objects may be resized relatively freely, while others may not. Text, for example, may not be greatly reduced so as to render it illegible.

---

*The PackIT Expert System*

After considering that PackIT's domain was quite unique and might therefore require special software for its implementation, I decided to develop a specialized expert system building tool rather than use one of the tools already available. This was done to avoid the dead-ends that the developers of these tools could not envision. For example, I had previously worked with the *Knowledge Engineering System* (KES) in the development of a simple predecessor of PackIT dealing with the design of business cards. KES is also a rule-based, backward chaining environment. I eventually discovered that the version of KES I was using did not allow complete freedom when dealing with numbers – attributes of the type integer were not permitted in the inference process. While KES made the implementation of the simple business card program quite easy, the above restriction was unacceptable in dealing with some of PackIT's major issues, such as scaling and relocation calculations of graphic objects on a computer screen.

In developing my own tool I was able to custom design any low-level function I needed without having to get around the limitations of pre-existing ones. Naturally, this also meant that most of the sophisticated facilities commonly found in other expert system building tools were not available to me; but after studying those functions in other tools, I was able to adequately duplicate many of them, such as rule searching, tracing, and justification.



## Programming Environment

PackIT was implemented on a Hewlett-Packard 9000 series 300 computer (commonly known as “Bobcat”) running under the HP-UX operating system, a UNIX System 5 port. It has 7 megabytes of core memory and a 68020 processor and 68881 math coprocessor. Both the expert system development software and the expert system itself were written in Common LISP in Hewlett-Packard’s NMODE programming environment, while most of the graphics routines were written in the C programming language. The Bobcat provides a *Starbase* graphics library with hardware support of 2D and 3D graphics primitives, and a 768 x 1024 x 8 bit frame buffer.

PackIT’s backward-chaining inference engine was implemented with the use of Hewlett-Packard’s object-oriented programming facilities. In this thesis the individual graphic elements of a package have been referred to as *objects* – this has more significance than one might expect. In object-oriented computer programs, an object is a data-structure representing a concept, situation, or thing which contains information about its current state. (See appendix D for a description of PackIT’s graphic-object class.) For example, a graphic object, such as a company logo, may have the attribute *width*. The width of the object may be accessed by sending a *message* to it asking it to return the value of that attribute. An object also has information on how it may be manipulated. A message such as *display-object* requests the graphic object to display itself on the screen. The power of message passing is that since the manipulation operations are all specified in the objects themselves, then the programmer may simply give general instructions without worrying about specific details. Although an action like displaying an illustration may require a number of different low-level commands than

would be required to display text, the differences are contained within the objects and not in the function calls.

### The Structure of Rules

The structure of an object is specified in the definition of its *class*. The class states all the attributes which may be associated with a particular object (which is said to be an *instance* of the class). The class rule is defined in PackIT as follows:

```
(define-type rule
  (:var sets)           ; what values does it set?
  (:var requires)      ; what values does it need to test?
  (:var test)          ; the antecedent test condition.
  (:var action)        ; the consequent actions.
  (:var other)         ; actions if antecedent is false.
  (:var active)        ; is rule currently active?
  (:var fired)         ; has rule previously been fired?
  (:var help))         ; description of rule.
```

The *sets* attribute states what values may be altered by the firing of this rule. It may change the attribute value of an object, or the value of a global variable. The values which the rule needs in order to be fired are specified by the attribute *requires*. *Test* is the condition or conditions under investigation in this rule; if the test is true, then the *action* statements will be executed, otherwise the *other* statements will be performed. The *active* attribute specifies whether the rule is being utilized at the present time, and *fired* states whether it has ever been utilized during the entire inferencing session. Finally, *help* provides an english explanation of the rule's functions and purpose. The roles of each of these attributes will be described below in relation to the inference engine.

The following is an example of the core of a typical PackIT rule dealing the importance factors of an object when the package is meant for a professional market.

```
profess-obj-impor
:sets      `(=> descript-obj :importance)
:requires  `(=> packagel :marketing-area)
:test      `(equal `profess (=> packagel :marketing-area))
:action    `(=> descript-obj :raise-obj-importance 0.2)
:other     `()
```

`sets` specifies that the rule could change the importance of having the descriptive text on the package. This is the text that describes the features of the product ; if the package was meant to be sold in a professional market, such text is very important. `requires` states that in order for the rule to be fired, it needs to know the marketing area of the package. If the marketing area is professional, then the `test` is true and the importance level of the descriptive text object will be raised by 20%, as specified by `action`. This particular rule has no `other` clause, which would normally be executed when the test is false. (The `active`, `fired`, and `help` values are not shown here.)

### The Inference Engine

PackIT's inference engine is backward-chaining; it is given goals to achieve (i.e. attribute values to find) and will search for any values it needs to attain those goals. It may be given a goal using the function `obtain`. `obtain` takes one argument – the attribute of a particular object whose value is desired. The attribute is then sent to the function `rule-search`, which will step through the rule base looking for a rule which contains the attribute in its `sets` specifier. When it finds such a rule, the rule is activated and its `active` flag is set to true. If all the attributes in its `requires`

specifier have values, then the rule has all the information it needs to be fired. This rule has the potential to set the attribute value depending on whether its antecedent phrase is true or false. If it is true, the `action` statements will be executed; if not, the `other` statements will be executed. After firing the rule, the inference engine checks to see whether the attribute has actually been assigned (it may be the case that it is set only by `action` or only by `other` – thus its success depends on which was executed). If a value has not been assigned, the search will begin again starting from the rule after the one just activated.

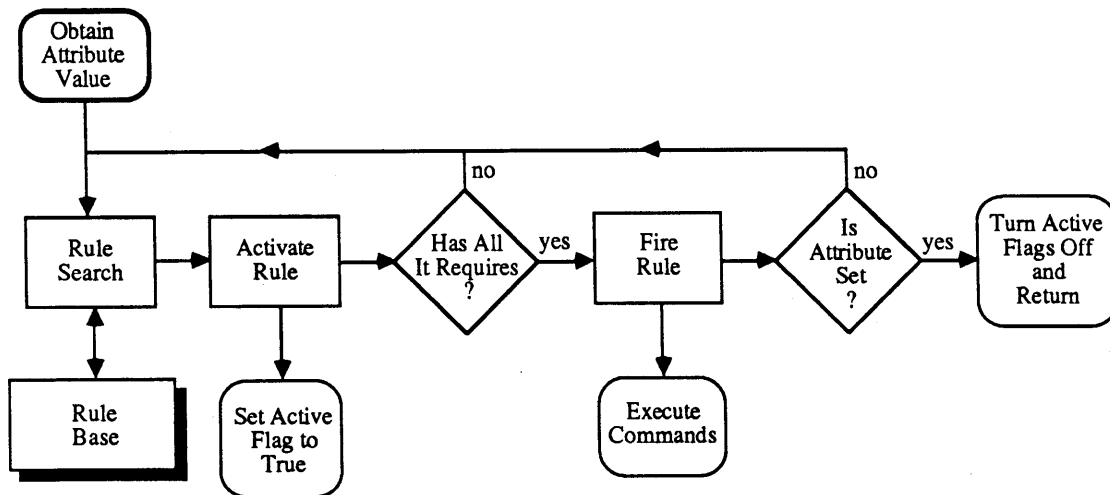
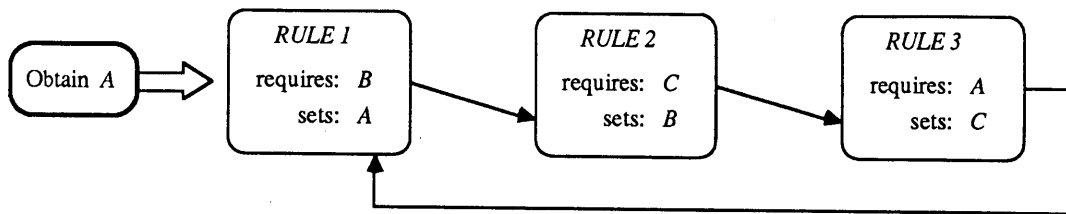


Figure 9.1

*PackIT's inference engine is asked to find an attribute value. It accesses the rule base in search of a rule which may set the value. The rule is activated and then fired if it has all the data it needs; if not this data is searched for. The rule is then fired. If the attribute has been set as a result, the active flags are turned off and the process ends; otherwise a search begins for another rule which might set the attribute.*

When a rule is activated, the inference engine checks to see if each of the attributes in its `requires` space has a value. If not, then it will do another rule search to try to set that value, which may, in turn, prompt an additional rule search. This is how the multiple levels of inferencing are produced. The purpose of the `activate` flag is to

ensure that no single rule is activated more than once during a single call to obtain, which might otherwise cause an infinite loop as illustrated in figure 9.2. The activate flags of all rules are turned off upon exiting the inference engine's top-level obtain function.



**Figure 9.2**

*An example of infinite looping caused by absence of "active flag." The inference engine is asked to obtain attribute A. Rule 1 sets A, but also requires attribute B in its test. It searches for a rule which sets B. Rule 2 sets B, but requires C. Rule 3 sets C, but requires A. Since there is no flag to indicate that Rule 1 is already active, Rule 3 finds A here. There is nothing to halt the loop. An "active flag" would prevent this by forcing Rule 3 to search for a different rule which sets A.*

There are two related ways in which obtaining an attribute might fail. One is if the first rule search fails to find any place where the desired attribute is set. Should this happen, PackIT will produce a warning message describing the details of the situation so that the rule base may be updated if so desired. A second source of failure is if the search is unsuccessful on any of the sub-level searches, thus resulting in a dead-end. Some other systems are designed to test a number of paths of logic which may actually lead to such dead-ends, and so they provide facilities for backtracking where the inference engine will go back and choose a different path than it had previously taken. Although PackIT has no such facility, it is designed to take the most promising path available, and thus its search failure rate is low.

Its inference engine provides a tracing feature which displays each rule fired allowing the user to follow the internal paths of logic taken. In addition, information justifying each rule's validity is held in the `help` attribute of that rule. This information is displayed for the rules fired when tracing is turned on, or for any rule upon request by the user.

### **Levels of Inferencing**

One of the major powers of expert systems lies in their ability search for the information needed to solve a problem – the knowledge engineer provides general rules which will be accessed by the inference engine for specific situations. For PackIT's backward chaining mechanism, this means that an overall goal is presented to the system whose completion requires the gathering of bits of information – each of which (now goals as well) requiring further information searches themselves. Thus, each goal is broken down into a number of sub-goals to form intricate inference chains.

To take an example, suppose we want to know whether the picture of a product is going to appear on its package. Figure 9.3 illustrates that a number of sub-tasks are assigned in order to answer the question.

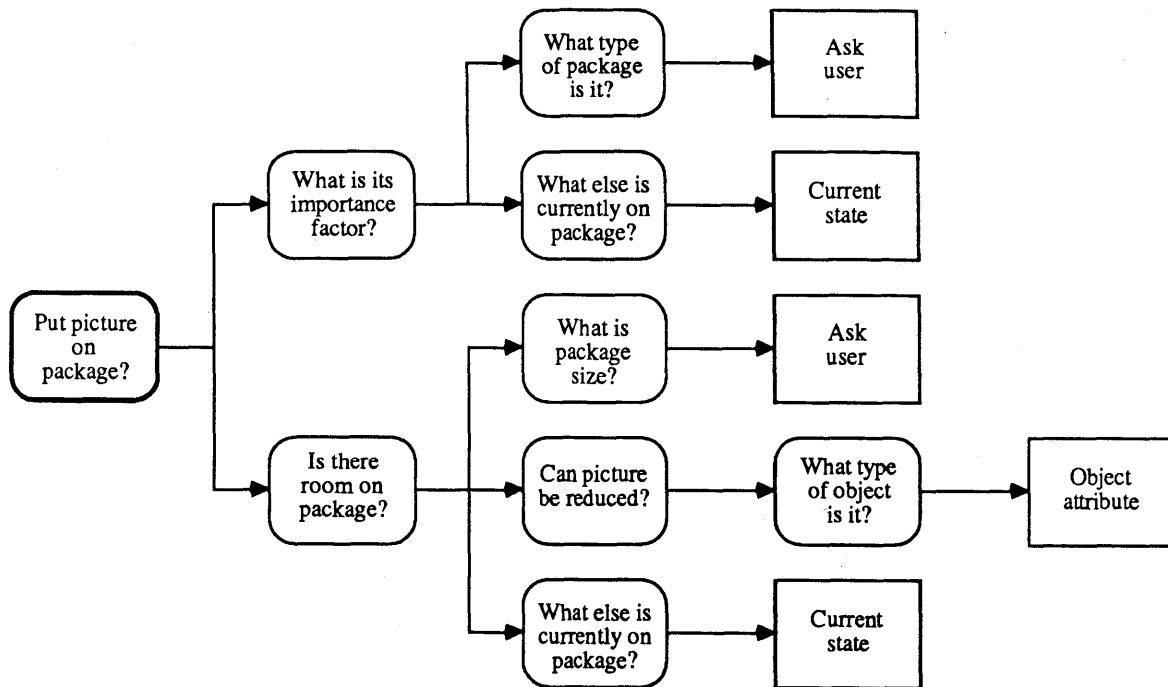


Figure 9.3

*Levels of inferencing will required when one asks if the product picture should appear on the package. This may be answered by finding out how important the picture is to the composition and by deciding if there is room to fit it. These two questions are further broken down until one of the three ultimate response sources provides an answer to each line of questioning.*

First the system must decide what the importance factor is for the object and whether there is room for it – they must both be true for the picture to appear. These questions are further broken down into other questions which may be divided themselves. They will ultimately be answered by accessing attribute values of objects, examining the current state of the environment, or asking the user for additional information. One line of reasoning in the example states that the object may appear if there is room for it; in order for there to be room it must be reduced; whether it is reducible depends on the type of object it is (if its a company logo, for example, it may not be reducible past a certain size); its object type is an object attribute and is readily available. Since this object type

may be reduced, the answer is yes for this line of questioning. The other lines must now be answered in order to make the ultimate decision.

Thus, the inference engine must go through four levels of inferencing in order to achieve the original goal. This is a typical example of PackIT's obtain mechanism; since its rule base is relatively modest, deeper levels of inferencing are not common. In fact, many of its goals are attained in fewer levels than are described here.



---

## *Features of PackIT*

This chapter describes the procedures involved in a typical PackIT session. Many of the facilities provided by PackIT are described here in the order they are most commonly employed at the session's start, although it is also common to access any of them individually later in the session.

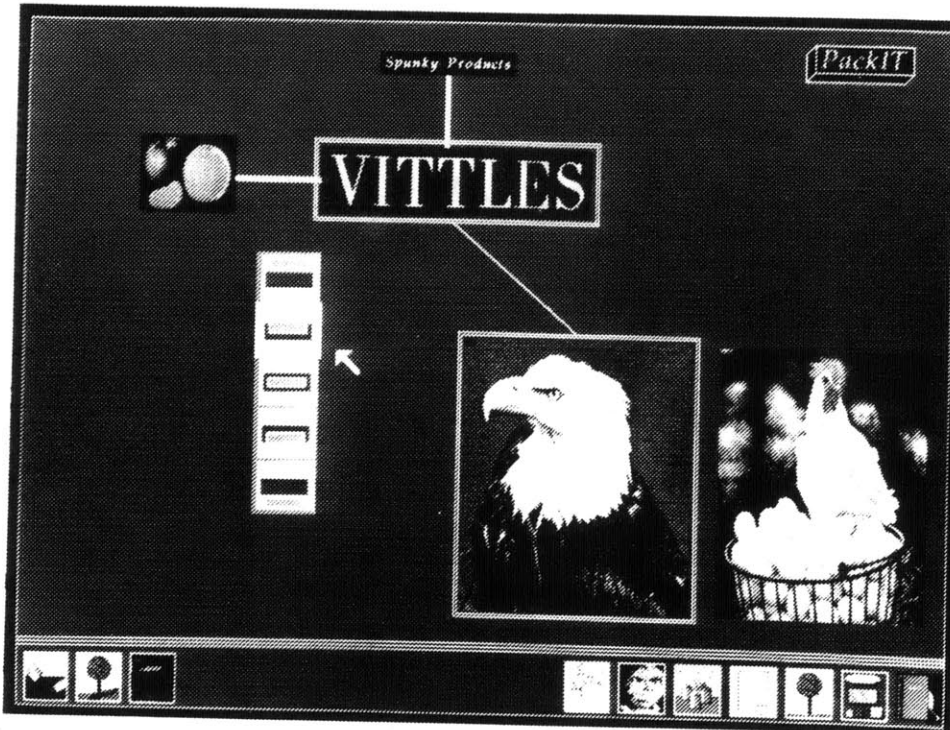
The first step involves the creation of the individual graphic objects which are to appear on the package. These objects may either be digitized images, text objects, or objects painted using the paintbox system described in chapter 1. The paintbox may also be used to change any of the objects through resizing, retouching, color modification, etc.

After the designer is satisfied with the individual objects, they may be sent over to PackIT. Naturally, PackIT can not tell the difference between a company logotype and a product picture, so the user must specify the category of each object at this point in time. This is performed using a graphical interface, after which PackIT's main facilities are ready to use.

### **Position Tree**

Once the objects created in the paintbox are sent to PackIT and labelled, the positional relationships between them must be specified. The user/designer builds a *position tree* by defining the relationships objects have between each other under ideal circumstances (as described in chapter 8). This tree contains the object hierarchy

information, such as *object A* is below and right justified with *object B*, and is created interactively using a tablet and pen as an input device.



**Figure 10.1** *In this picture of PackIT's tree creation interface, the user has already put three objects into the tree. He is now adding a fourth by declaring the "Vittles" text as a parent and the eagle as a child. The positional relationship between them is specified with the pop-up menu near the arrow cursor.*

The objects are initially presented near the bottom of the screen with the root of the tree (the company-name object, in the case of PackIT's rule base) appearing at the screen top. The user may choose a parent object by "clicking" on it with the pen, connect it with a child object, and indicate the relationship between them with the use of a pop-up menu which appears. Once a link is made, the child is moved from the bottom of the screen to the appropriate area near its parent with a branch drawn between them. Other objects are similarly chosen until the tree is completed to the user's satisfaction. This tree

represents the ideal positional structure that the user wants the package to take on – it may be considered the standard or prototype upon which the other family members should be based. The relationships may have to be altered by rules during the inference process depending upon the conditions set by the user.

PackIT makes sure that only objects which are already in the tree may be chosen as parents, and only those which are not may be chosen as children. Thus, there are no loops in the tree. The user may also choose to use the default tree structure which has been declared in the rule base as a prototype.

### **Cropping Factors**

One of the options of the rule base is to crop certain objects if needed. The graphic-object class has as part of its definition the attributes `xcrop` and `ycrop` which specify the cropping dimensions of the objects. They each contain a pair of numbers stating the percent distances from the left (for `xcrop`) and top (for `ycrop`) of the object that may be cut out. For example, the following values:

```
(object-1 :xcrop) = (0.1 0.9)
(object-1 :ycrop) = (0.2 0.6)
```

state that in the x dimension the left 10% and the right 10% of the object may be cropped if necessary, and the top 20% and the bottom 40% may be cropped in the y dimension. While these factors are pre-set by default for each object, they should naturally be changed if the default pictures are not used and instead pictures are sent over from the paintbox environment during run time. To set them, the user calls the `set-cropping` function which lines up the objects and displays a rectangle within each one to indicate its current cropping dimensions. He may then draw a new rectangle to reset it if desired.

### **Package Attributes**

The package attributes, such as expected marketing area and projected quality level, may be changed at any time during the session. These attributes are accessed by the rule base and have a number of effects on the design of the package as discussed in chapter 8. Package layouts may be generated under these varying conditions, and then displayed side by side for comparison.

### **Tracing**

PackIT's inference engine provides a tracing facility to illuminate its internal strategies and processes to the user. If the user clicks on the tracing icon, each rule will be described as it is fired during inferencing. Provided are the rule's name, an explanation of its test conditions and actions, and a statement of whether these actions were taken or not. The tracing function illustrates how the system goes about solving the design problems and is also a valuable debugging tool.

### **Rejected Layouts**

Another valuable feature is the `save-and-show-rejects` function. PackIT creates a design by first attempting to layout the objects in the locations indicated by the position tree hierarchy. If one or more of the rules state that there has been a violation and that something must be changed, the composition will be appropriately altered and retested. Normally this happens internally, with only the final layout displayed at the end. When the rejects flag is turned on by the user, however, each package declared unacceptable and rejected by the rule base will be displayed while it is being analyzed. For each reject a small icon appears on the screen; the rejects have all been saved and may

be viewed by the user by clicking on the icons at a later time in the session.

This feature is, in a sense, a graphical counterpart to the tracing facility described above. It provides information for the user who is now able to see what is actually happening internally when the rule base goes to work. The two features when activated simultaneously are a powerful combination which makes PackIT's designing methods almost self-explanatory.

### **Redoing the Layout**

There are times when the rule base will create a layout which is less than satisfactory – perhaps the package dimensions provided by the user were unreasonable (much too narrow or small, for example) or the design simply does not seem please the user. This is to be expected at times since PackIT does not consider the huge number of design criteria that a human designer would. After the inference engine is finished and the suggested layout is generated, the user may “grab” the objects on the package and reposition them on the package as he pleases using the tablet and pen, or he may alter the package attributes or dimensions slightly and request another layout.

---

*Future Work*

It is clear that the complexity of PackIT's nature brings forth an entire realm of interesting issues which could be addressed in the future. Its development was a significant endeavor that took place in a relatively short period of time. Naturally, such a job could be continued almost indefinitely – it seems that for every question I addressed, many more questions arose. This chapter describes some of the additional research which could be done to both improve the features of PackIT and to tackle the issues raised by its conception.

**Extension of Rule Base**

The rules currently employed by PackIT are relatively few in number (less than 50) and are adequate in the generation of only simple layouts. In order for the system to become more powerful and generally usable, the rule base must surely be extended. This would increase the range of possible solutions for any given layout, as well as improve the quality of its results. Two ways to accomplish this are to supply additional rules into PackIT's existing categories (rules of color, scaling, cropping, etc.) or to add new categories of rules.

To address the former suggestion, the class of rules dealing with the scaling of objects, for example, only considers individual objects at the present time. Scaling would be more effective if rules dealing with relationships between objects were used – perhaps certain groups of objects would require to be scaled only in unison, never individually. Another improvement would be to add more rules dealing with the repositioning of objects. Presently, if an object must be shifted it will only be moved to

some other location in relation to its parent; if no space is available near to the parent, it will be returned to its initial spot. Rules dealing with repositioning near to other objects would be helpful.

The second suggestion is to add new categories of rules, such as rules of space or balance. The balance of a package is fairly easy to calculate by traversing the positional relationship tree to see the formation of the objects. A rule could be entered stating that if the package is unbalanced or if there is too much open space in an area, then some of the objects should be shifted. (See figure 11.1.) This is one way the system might avoid generating a layout that the domain expert might consider a poor design.

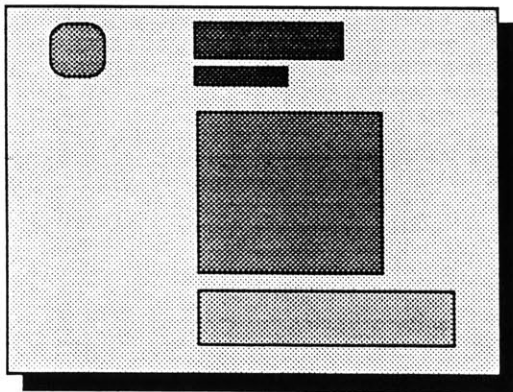


Figure 11.1

*The layout shown here is unbalanced with a large region of space. The system could potentially traverse the position tree to discover that most of the objects' weight appears to the right of the central axis. It could also be determined that the object on the left is the solitary object on that side. If rules state that such conditions are unacceptable, the objects would be repositioned.*

The rule base could also be improved if the graphic object class were extended to handle additional user-input package attributes. The primary attributes in use now are the *marketing area* and the *projected product quality*, which account for a large percentage of the most powerful rules in the system.

### **Rulebase Access During Run-Time**

One of the most common requests I receive when demonstrating PackIT is to display the knowledge base which is governing the layout process. While this is not a problem in itself, most people (including programmers) are unable to immediately understand what the rules actually mean, since they are in a highly stylized programming language. It would be of great value to the user if he were able to view any of the rules in graphical form during run-time. Displaying them graphically makes their meanings more transparent and would facilitate the representation of the structure of rules, rule class groupings, relationships between individual rules, etc. It would also permit the use of other graphic aids such as color and typefaces of various styles and point sizes.

Once this is available, a natural extension is to allow the user to actually alter these rules during a layout session. This is especially useful in the domain of graphic design since the rules are not standard, but vary somewhat from expert to expert. If after a package has been generated the user (who we are assuming to be a designer himself) says "I like this layout, but I would have done a few things differently," it would be desirable if he could access the pertinent rules and alter them as he sees fit. The alterations would then be saved as the user's personal rule base sub-set, and could be activated in subsequent sessions. Thus, personalized work environments tailored to the specific tastes of the designer would be possible.

Facilitating the alteration of the rule base is important not only if the designer's tastes or the company's standards change, but even moreso when the company is first using the system. If their own rule base must be developed from scratch, adding and fine-tuning rules must be easy for such a system to be worthwhile at all.



### Graphical Explanation/Justification Facilities

Most expert systems have facilities for explanation (a description of how the system arrived at a conclusion) and justification (supporting evidence stating why a rule is valid). By convention, they take the form of textual question/answer sessions. PackIT's domain seems to warrant a different approach – since the solution to the problem is graphical, it would be appropriate to interrogate the system using a graphical interface. Rather than typing in a request at the keyboard to justify rule 22 or explain conclusion 4, the user could point to a particular part of the layout and request information about the rules which affected this area.

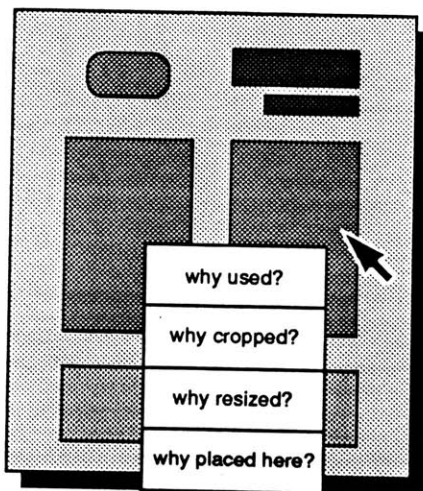


Figure 11.2

*Should a user want an explanation from the system, he would point to the object or area in question using the mouse, and a number of possible topics would pop up. No keyboard input is necessary; the user doesn't have to inquire in a programming language, or even in natural language – graphical questions are asked graphically.*

For example, if the user points to the area between two objects, a pop-up menu would appear asking him what he wants to know about the decision such as: Why do the objects appear together? Why are they left-justified? Why do they appear at the bottom of the package? The type of questions appearing in the menu would depend on the type of area pointed to by the user. Answers to the questions could be retrieved in a manner similar to that used by the `obtain` function described in chapter 9.

Allowing the user to access information in this fashion would have the following advantages:

- The user would not have to be immersed in screens-full of esoteric, unintelligible rules.
- The prompts of the system would help the user organize his questions.
- Questions (about graphics) asked graphically would be much easier to convey.

### **Metaknowledge**

In addition to possessing knowledge of packaging and design, it would also be useful if PackIT had some high-level knowledge about what it knows. For example, since the rules of design are much more likely to be wilfully overridden than those of other domains, such rules should be monitored and updated if necessary. Conversely, if a rule is accessed frequently, this might be a signal to add other similar rules which may be just as valuable. Finally, there could be metarules that specify the conditions under which certain other rules may be overridden. These are all situations that might not normally be handled otherwise.

### **Setting User Constraints**

After PackIT has presented a suggested layout, the user is able to reposition any of the objects as he sees fit. At the present time this manual alteration is a final step in the layout process – once done, PackIT can no longer take over. Although the user should naturally have the final say in the composition, he may want to take the suggested layout, change one of its features, and ask PackIT to continue its analysis. If the user should decide to shift an object from its suggested position, for example, a number of internal conditions are changed: the position hierarchy is altered and some rules may have been violated. It would be of great value if PackIT could register these changes, which may

have put the package into a state of disequilibrium, and once again activate its inference engine using the user's change as an unalterable constraint. If the change desired by the user turns out to be unacceptable, PackIT could explain why. This would allow the user to take a greater role in the design process while being certain that the package conforms to the standards set by the company.

### **An Educational Tool**

PackIT works by taking the specifications set by the user and accessing its rule base in order to generate a satisfactory layout. These rules, however, could just as easily be used as a basis for analyzing the work produced by a student of design. To do this, the process would be reversed: rather than using rules to construct a layout, the layout designed by the student would be presented to the system, which would critique the composition by testing to see if any of its rules were broken. The system would then describe the reasons why it felt the student's work was or was not acceptable. Naturally, this implies the need for sophisticated explanation/justification facilities – whereas before they were merely desirable features, they would now become an indispensable part of the system.

Such a feature could also be used as an alternate approach for debugging the knowledge base. An expert could present a layout to the system and observe whether or not its critiques were valid. If the system's suggestions do not seem plausible, the rules which prompted them could be tracked down and altered.

### **The Third Dimension**

An important packaging consideration which is conspicuously absent from this thesis is design in three dimensions. While I have generally been concerned with the layout of

a box's front side, most of them have at least six sides each with the same types of rules and constraints I have previously described. In addition, there are strong relationships existing between sides that also must be considered by the package designer. This is clearly illustrated in figure 11.3.

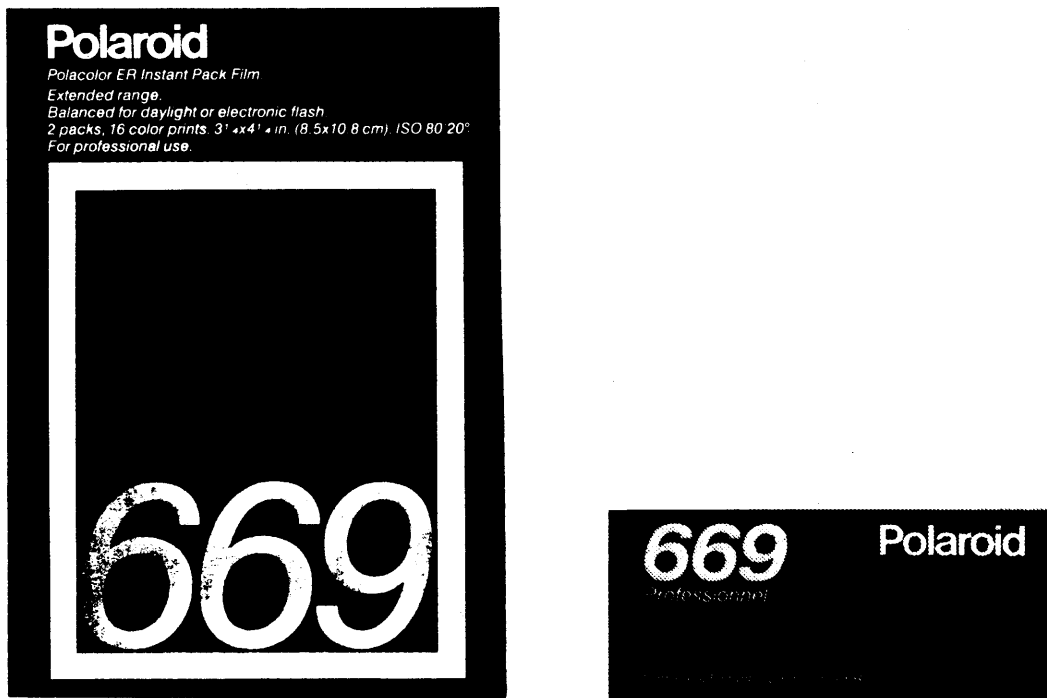


Figure 11.3 A package where the third dimension plays an important role.

This Polaroid film box was designed so that it could be displayed either standing upright (with the large side facing toward the consumer) or laying on its back. These two sides are analagous to two different members of a package family; both sides provide the same information – company name, product name, and product description – in the same style, but with a different format. They are both equally effective selling agents.

Certainly packaging is a 3D design problem, not just a 2D one. An obvious area for continuation of this project is to extend the rule base to consider these 3D aspects of

design. While a number of new rules would have to be added, many of them would turn out to be quite similar to those already existing in the rule base.

## *Conclusion*

This paper has described PackIT, a rule-based system which assists designers in the layout of package families. PackIT's rule base contains various rules of graphic design as well as the packaging rules of a hypothetical company. It takes these rules and, given a number of specifications provided by the user/designer, generates a layout appropriate for that company's image. The specifications may be altered in order to create entire families of packages. PackIT is intended to present options to the designer by following a simple set of rules, as would an apprentice designer, and is thus a visualization tool which in no way claims to substitute for the design expert. In the past, computer graphics has been used by a number of expert systems as a user interface aid. This paper has illustrated that these roles can be reversed, with an expert system serving as a tool within a larger design workstation.

PackIT is part of an ongoing study at the Visible Language Workshop and was intended to continue investigation into the intersection of graphics with design knowledge and artificial intelligence. It lays the groundwork for further exploration into methods of user/machine interaction with rules, graphic representation in the user interface, and non-verbal knowledge acquisition in the world of print and electronic visual communication.

I have discussed a number of rules used by PackIT as well as some more general rules that could be incorporated in the future. I have also concluded that knowledge acquisition in the design domain requires great consideration. Careful analysis of the design process, because of its complex and somewhat informal nature, is highly important. While I have described some of the unique knowledge acquisition methods

used for PackIT's development, much more work is needed to capture the more aesthetic-oriented rules. Once formalized I believe they are as usable as the rules of any other existing expert system.

I have shown that such a knowledge base is particularly useful in the design of package families where rules generally remain constant throughout the product line. Companies are concerned with both creating unique designs for each of their products and at the same time retaining their company identity overall. This is clearly illustrated by the corporate identification manuals where the relationships between family members are well-defined. By using a rule-based packaging design system, the designer is liberated from memorizing the numerous low-level rules which hold true for specific companies, and may concentrate on solving the overall design problem at hand. Although PackIT's knowledge base is relatively small and simple, I believe that a larger, more comprehensive one is possible and could be quite useful.

The development of PackIT was an enormous undertaking and there were many obstacles in implementing the concepts described in this paper. The first is that many of the ideas of design are difficult to formalize. Many of the reasons why designers do things are unknown to them – it is not always an easy task to determine why one composition works and another does not.

Another weakness with PackIT in particular is that since its rulebase is somewhat limited, its designs tend to be limited as well. This is simply because when given a problem it has only a small number of options from which to choose, and therefore only a small number of suggestions to offer. In fact, after building and working with the knowledge base for a number of months, I can almost predict what the final layout will look like in many cases.

A third problem with the system is that since the knowledge base is small, this means

that some of the individual decisions it makes are not going to be the optimal ones. For example, if the inference engine has determined that one of the objects must be repositioned due to space limitations, it might be repositioned in an area where there is room without regard to other important factors such as the change in symmetry or balance. It may be the case that the object would work even better elsewhere – but this might not be considered.

In addition, although I have generally projected an optimistic view of how rules of design are quite similar to rules of other domains, the fact that no comprehensive design expert system exists seems to question this idea. The most important step in expert system design is to extract and formalize knowledge provided by experts. While I am convinced that the formalization of design rules is possible by utilizing the knowledge acquisition techniques described earlier, this process may not be as simple as one would like. The problem of design formalization is one of the many research areas that should be explored more fully in order to facilitate the development of design expert systems. Other suggestions like those presented in chapter 11 would also be of great value.

While many important concepts were presented here, I feel that I have merely scratched the surface of this huge realm of inquiry. Indeed, many of the individual topics covered could each become the source of large-scale investigations themselves. The use of computers as qualified delegates for human experts, and the dubious computer representation of creativity (or the definition of creativity, for that matter) are but two examples of major areas for future research that were only touched upon in this paper. I have discussed the idea that it is unacceptable to discard a design expert system simply because one doesn't believe that computers can possess the genuine intelligence of a human – we must instead judge the system by the quality of its results. I believe that



PackIT, representing a modest but significant entry into this complex world, produces admirable results that should be seriously considered by both optimists and pessimists alike. Perhaps if this idea is recognized, many of the skeptics might agree that the computer as knowledgeable assistant represents an entirely new class of graphic design tools which may soon be of inconceivable value to the world of design – possibly as significant as any other design tool known to man.

---

*Expert Systems*

The term artificial intelligence (AI) may be defined as that area of computer science concerned with the development of computer programs which are considered to be “intelligent.” When we say intelligent systems, we are referring to systems that are able to simulate human activities through the formalization and symbolic representation of human skills or knowledge. AI is divided into a number of categories such as robotics, natural language processing, and knowledge-based expert systems, the topic of this thesis.

In the 1960s, AI scientists attempted to develop general problem-solving systems which could address broad classes of problems. This proved to be ineffective since the human processes that were being mimicked were much too complex and wide-ranging and, as a result, the programs could not handle lower-level sub-tasks of the domain very well. They decided to remedy this by devising more efficient search techniques (which accessed internal information faster) and better methods of representation (which permitted more effective formalization and retrieval of information). While this improved the situation to some extent, it was eventually realized that such a system would be much more powerful if it were highly specific and contained a greater amount of knowledge of a particular domain. (Waterman 1986, 3-4)

This idea led to the birth of expert systems, which are designed to assist humans in problem analysis or decision making of limited, highly-specific domains. S. Jerrold Kaplan gives a concise comparative explanation of exactly what knowledge engineering (the process of developing an expert system) is:

Knowledge engineering is the representation and use of symbolic knowledge in electronic form to solve problems that normally require human intelligence or attention. The primary difference between data processing and knowledge engineering is that data processing deals with the representation and use of data, whereas knowledge engineering deals with the representation and use of knowledge. Data processing is concerned with algorithmic, repetitive processes: knowledge engineering is concerned with judgmental and inferential processes. Data processing is focused on the efficient handling of large data bases and large numbers of transactions, but knowledge engineering is concerned with the effective handling of knowledge bases and decisions. (Kaplan 1984, 52)

An expert system provides an interactive environment where the user describes details of his problem to the system which then uses this information and its knowledge base to reach a solution. These individual pieces of user-input information are usually requested automatically by the system. Typical domains include circuit design, chemical analysis and geological exploration. These limited domains are well-suited for implementation as expert systems. In fact, at times even some of the most successful systems produce incomplete analyses because their knowledge bases need to be quite large. The knowledge-based system, *INTERNIST*, for example, is not quite as accurate as one would like, simply due to the fact that its domain, internal medicine, is so extensive.

### **Successful Systems**

Some examples of successful expert systems are described here.

*DENDRAL*, which was developed at Stanford University, performs analysis of chemical compounds by examining the physical spectra of a molecule under investigation (Feigenbaum et al. 1971). It attempts to find the composition of this unknown substance

by generating a list of all known substances which are consistent with the data provided by the user. As additional information about the substance is entered, DENDRAL eliminates more and more of these candidates until a conclusion is reached by process of elimination. This is known as a *Generate and Test* technique.

*MYCIN* is a rule-based system which assists in the diagnosis and treatment of infectious blood diseases. Its knowledge base consists of approximately 400 IF-THEN rules (known as *production rules*), each of which may be accessed when needed (Hayes-Roth et al. 1983, 9-10). A production rule consists of an antecedent clause (a set of conditions which are tested) and a consequent clause (a set of actions performed if the value of the antecedent clause is determined to be "true"). When a rule is activated, the data needed for determining the antecedent's value is either requested from the user or inferred through the firing of other rules. Thus, complex chains of inferencing are formed during run time. Problem-solving techniques similar to those of MYCIN have been utilized in PackIT and will be described at length in a later section.

*MACSYMA*, developed at the Massachusetts Institute of Technology, is a system which performs symbolic manipulation of algebraic expressions. Mathematical expertise is incorporated into rules which reduce complex equations into their simplified equivalents. These reduced expressions produced by MACSYMA are much more easily utilized by the user. (Moses 1984)

*R1* (also known as *XCON*) was developed by a joint effort between Carnegie-Mellon University and Digital Equipment Corporation (DEC). It is used by DEC to configure VAX computer systems. R1 is given a customer's purchase order consisting of the

computer hardware desired, and calculates an internal layout of the environment based on the relationships between each of the components. By doing this it determines what components must be exchanged, added to, or deleted from the purchase order for the customer's system to be complete. (McDermott 1981)

### **Structure of Expert Systems**

The two primary elements of an expert system are the *knowledge base* (the body of knowledge which represents human expertise) and the *inference engine* (which determines how and when this knowledge will be accessed).

### **The Knowledge Base**

The knowledge is often represented in the form of rules, frames, or semantic nets. A semantic net consists of a network of nodes connected by arcs representing the relationships between nodes. Two common relationships are the *isa* and *has-part* arcs, such as *Tom's duck isa bird* and *Fred has-part leg*. A frame is an entity which provides the description of an object or concept. Each frame contains attributes (known as *slots*) and their values which describe each of the objects. Groups of frames are linked together to permit slot inheritance between each other.

This thesis is concerned with a rule-based approach, where knowledge is represented by a number of production rules. A production rule is a formalized piece of information about a particular domain which is useful for solving specific problems. Rules often take the form of *heuristics* – guidelines or rules of thumb which are commonly used by experts to guide them toward the correct path of a solution. Since most assertions cannot be

declared as entirely true or false, rules are often associated with a *certainty factor* (CF) as is used by MYCIN. For example,

IF:     The stain of the organism is grampos,  
           and the morphology of the organism is coccus,  
           and the growth conformation of the organism is chains,  
 THEN: There is suggestive evidence (CF 0.7) that the identity of the organism  
        is streptococcus.

Certainty factors are used to state the strength of belief that a certain situation exists or that a particular rule is plausible. In MYCIN, a certainty factor ranges between -1 (suggesting a false hypothesis) and +1 (suggesting a correct hypothesis).

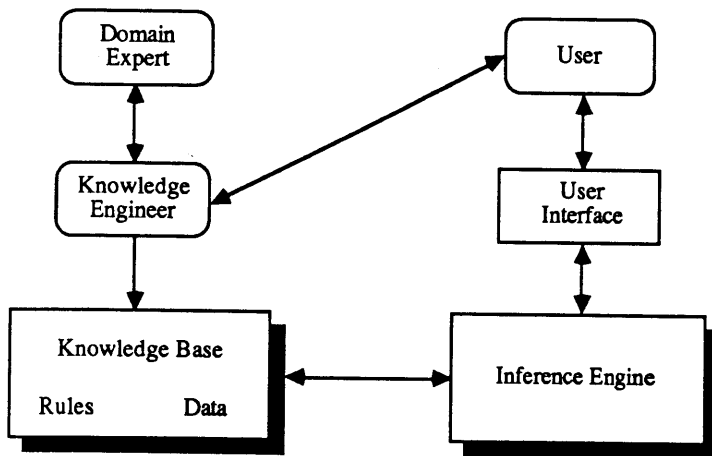


Figure A.1

*An expert system environment. The knowledge engineer interviews the domain expert and encodes this information into the knowledge base. The user (through the interface) interacts with the inference engine which accesses rules and data stored in the knowledge base. Additional comments and suggestions may be offered to the engineer by the user.*

### **The Inference Engine**

The inference engine is used to guide the system as it fires rules, asks questions of the user, and reaches conclusions. Whereas the rule base contains the knowledge of a particular domain, the inference engine contains the knowledge of how to access this information in order to solve problems.

Harmon and King describe the function of the inference engine:

Notice that the inference engine stands between the user and the knowledge base. The inference engine performs two major tasks. First, it examines facts and rules, and adds new facts when possible. Second, it decides the order in which inferences are made. In doing so, the inference engine conducts the consultation with the user. (Harmon and King 1985, 49)

An inference engine can either be *forward chaining* (data-driven) or *backward chaining* (goal-directed). Forward chaining inferencing might be desirable when the goal is not actually known (perhaps because of a large number of possible outcomes) and the system must construct it. In this type of strategy, the system takes the information known and applies it to rules, determining the path that is to be taken. (See figure A.2.)

The inference engine of PackIT is a backward chaining one. In this type of inferencing method the system is given a goal to achieve, and the engine finds any value it needs to reach that goal. (See chapter 9 for a description of PackIT's inference engine.)

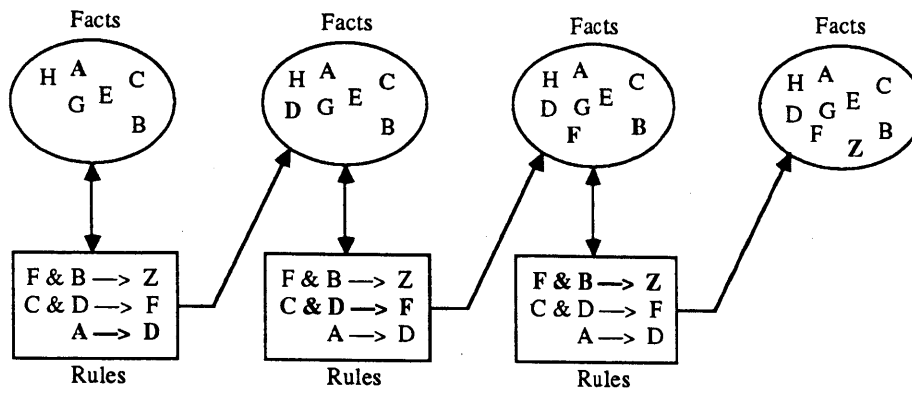


Figure A.2

An example of forward chaining with rules and facts known about the environment. Since one of the rules states that *A* implies *D*, and *A* is a known fact, *D* may be added to the list of facts. As a result, the rule *C & D* implies *F* may be fired, adding *F* to the fact list, etc. Thus, this system is driven by rules applied to facts. (Waterman 1986, 66)

Some inference engines can perform *backtracking* in cases where the system has hit a dead end and has not yet reached a solution. If, at any point during inferencing, the system had been in a situation where it had a choice of two or more paths to take, it will search backwards from the dead end to find the latest occurrence of this choice. It will then restart its inferencing on one of the paths it had not chosen previously.

### Expert System Utilities

Most expert systems provide other functions which assist the user. Explanation and justification facilities are commonly used to aid in the running and debugging of a system.

When the user wants to know how the system arrived at a particular conclusion, the explanation facility should be able to provide that information. It usually does this by tracing the chain of rules that were fired in the inferencing process. For example, the



user may want to know why conclusion  $z$  was reached. The system might state that  $z$  is true because both  $x$  and  $y$  are true. If the user wants to know why  $x$  and  $y$  are true, this is available as well. The system should also be able to explain why a particular question was asked of the user during inferencing. Some explanation facilities provide *hypothetical reasoning*, where the system describes what might have happened had a certain rule or user-provided fact been different, and *counterfactual reasoning*, where the system explains why an expected result was not attained (Waterman 1986, 91).

While explanation facilities may explain why particular rules were fired, justification describes why a rule is valid in the first place. For example, a knowledge base dealing with chemical spill management might have the following rule and justification:

Rule:            If the spill is sulfuric acid, then use lime.

Justification:  Lime neutralizes acid and the compound that forms is insoluble and hence will precipitate out.

Note that explanation/justification facilities are extremely useful when attempting to track down fallacious rules or lines of reasoning in the knowledge base. PackIT has, as part of its rule structure, an attribute variable for the provision of justification information. The user may ask for a rule's provenance at any time during the session.

*Tracing* is also an important feature for debugging rules or explaining conclusions. The system provides an ordered listing of all the rules fired for that inferencing session so that the user may trace its line of reasoning. PackIT provides such information by displaying a rule's name and actions taken when the rule is fired.

### Developing an expert system

The main participants involved in the implementation of an expert system are the domain expert, the knowledge engineer and (to some extent) the end user. The domain expert is the source of expertise which is to be incorporated into the system. It is important that he express the various shortcuts and heuristics that he has used throughout the years of performing his job. The task of the knowledge engineer is to extract and formalize this expertise primarily through a series of interviews. This process of protocol analysis is called *knowledge acquisition*. The engineer is familiar with the tools and processes involved in building a knowledge base. While in many cases he does not have much experience in the domain under study, he must eventually absorb much of the information provided by the domain expert; thus it is a natural consequence of the knowledge acquisition process that he ultimately become somewhat of an expert himself.

Once the system is at a workable stage, a number of users may be asked to experiment with it. They may reveal errors in the knowledge base and provide suggestions to the system designers.

The system is built using a developmental tool which serves as the programming environment. Most of these tools are high-level languages specifically designed for the purpose of expert system development. It is not uncommon, however, to also find systems that are built using more conventional programming languages such as LISP. While LISP allows more programming freedom, it comes at the expense of the facilities normally provided by the high-level expert system tools.

---

*The Bad News*

In the past, the advent of any new and powerful technological advancement has generally been accompanied by criticism and questioning of its implications on mankind. The impact of AI on society has naturally evoked great concern as well. Perhaps one of the most wide-ranging criticisms posed is, “How can we successfully simulate something that is so mystifying to us?” If we do not really understand how the human mind works, then it seems to be an impossible endeavor to try to symbolically represent it.

As Beau Sheil muses:

The essence of the AI programming task can be made clear simply by reflecting on the meaning of the phrase “artificial intelligence.” “Artificial” is easy enough, that means using machines.... “Intelligence” is somewhat harder. I rather like Minsky’s definition of “intelligence” as a property we ascribe to intellectual behavior that we admire but do not understand. Iconoclastic, but effective.

The implication for AI programming of Minsky’s unsettling definition is that, if you are building an AI program, you are, *by definition*, building a program that you don’t understand....If you *do* know what you are doing (or if you find out), it isn’t AI anymore, it’s something else, because it can’t possibly involve “intelligence.” (Sheil 1984, 288)

This section describes some of the areas where expert systems sometimes have difficulty simulating human intelligence.

## Weak Points of Expert Systems

- **Commonsense reasoning**

Waterman describes a complaint of knowledge-based systems which has also been held by many programmers and users of conventional software. The problem concerns the lack of common sense displayed by computers. He describes the problem as follows:

Commonsense knowledge includes knowing what you don't know as well as what you do know. For example, if you were asked to recall the phone number of your previous residence, you would search your memory, trying to retrieve the information. If you were asked to give the phone number of England's prime minister, you would know immediately that you didn't know the answer and not even try a retrieval. If you were asked the phone number of Shakespeare, you would know at once that no answer exists, since telephones weren't around in Shakespeare's time. When an expert system is given questions it can't answer or for which no answer exists, it doesn't have the common sense to give up. Instead, it may waste much time searching through its data and rules for the solution. Even worse, when the solution isn't found, it may think it's because its knowledge is incomplete and ask for additional information to complete the knowledge base. (Waterman 1986, 15)

While there has been much successful research in the speeding up of information search routines, the previous example is clearly a case where virtually no search should occur at all. Humans are immediately able to recognize certain tasks which are overly time-consuming, fruitless, or simply absurd, and discard them with describable justification. One might accept the idea that humans not only know individual facts, but also have some high-level knowledge of what information is accessible by each of us, and what is not – we are aware of the boundaries of our intelligence. This type of metaknowledge (knowledge about knowledge) is not automatically provided as a by-product of knowledge base development.

The countless bits of information, such as knowing that there were no telephones in the sixteenth century, are not unique to an expert's realm of knowledge, but are commonly known facts. Since they are so numerous and wide-ranging, it often becomes impossible to capture them all in a knowledge base. One must remember that the power of expert systems lies in their ability to represent domains which are highly specific; as the domain becomes more general or far-reaching, this power tends to deteriorate.

Although how and with what difficulty this type of knowledge is stored in the human brain is certainly disputable, this thesis does not pretend to formulate any ground-breaking theories of psychology. The important point is that expert systems are not particularly adept at commonsense reasoning, and metaknowledge can be useful in the attempt to make them more effective. Lenat et al. (1983, 234) illustrate this in the case study of a chemical spill management system:

IF: The substance spilled is biologically active,

THEN: The "Spill knowledge base" is inadequate for the task.

This rule contains knowledge about the system's knowledge. It helps to eliminate useless searches in situations where the system will inevitably fail or provide inaccurate solutions due to insufficient information.

In graphic design such reasoning is quite problematic at certain times, and is easily solved at others. For example, when composing the layout of a page (note that in this thesis when I refer to "page layout," I generally mean the layout of objects on a package side, pictures in an advertisement, text in a book, etc.) we would immediately observe that a collection of images totalling 200 square inches could not fit on a page 100 square inches in size without overlapping. Such overlap detection is trivial to implement in a

computer system with the use of simple geometric algorithms. In fact, algorithmic utilities do not constitute the value of expert systems. Their power is displayed by the way they infer what pictures must be reduced in size, cropped, or omitted altogether, for example. Thus, there are times when solutions which would normally lead to dead ends are easily detected by the computer.

Other problems are substantially less trivial in nature. For example, consider a situation where a number of graphic objects are to be put on a page, and two of them are particularly striking, with various bright colors. While both the designer and the expert system realize that (for this particular client) the nearby placement of such competing objects is forbidden, the system might continuously attempt to reposition the two when in some cases an acceptable resolution may be impossible. It knows that there is a problem, but it doesn't know that there is no immediate solution. The designer, on the other hand, might immediately realize that one of the objects will have to be altered somewhat or perhaps even removed. Although one may retort that this is not an insurmountable problem since we may simply assign another rule or two to prevent these futile searches, it must also be conceded that such rules are of a somewhat higher order, and thus require special consideration. They are of a higher order because they are not simple rules, but are rules which know about the presence of other rules. This type of metaknowledge is discussed in chapters 7 and 11.

- **Creativity**

There is also a problem of representing creativity. While humans often handle situations by reorganizing knowledge to develop problem-solving methods, expert systems behave in somewhat uninspired, routine fashions (Waterman 1986, 14). The question of whether design expert systems are truly able to create is clearly an important

issue which must eventually be addressed. Unfortunately, this topic covers a huge and complex realm of inquiry extending far beyond the scope of this thesis. It is important, however, to mention it here in this list of general problems. Let me merely suggest that its resolution might lie in the questioning of human creativity itself. What makes it so mysteriously unique? Is it unique at all? PackIT's knowledge base is comprised of rules commonly used by designers who are thought to possess creativity. Whether or not this makes it creative is anybody's guess.

- **Inaccurate representation of knowledge**

Another fault of expert systems lies in their sometimes inaccurate representation of human intelligence. Herbert and Stuart Dreyfus promote the idea that computers could never think like humans because human experts do not always act by referring to rules. "When things are proceeding normally, experts do not solve problems by reasoning; they do what normally works....Skilled outfielders do not take the time to figure out where a ball is going. Unlike novices, they simply run to the right spot." (Dreyfus and Dreyfus 1986, 46)

While the authors are quite correct in stating that humans do not stand around consciously applying rules, they claim that experts act "intuitively." They lead us to believe that this intuitive knowledge is somehow divorced from the rules experts once consciously applied. I feel, however, that this position is not valid. Let us not forget how the experts attained this ability to act without conscious effort. Every element of expertise was, at some earlier point in the expert's life, unknown to him. The reason he is able to execute these actions appropriately is because earlier in his career he applied rules to guide him. Granted, the outfielder doesn't have to think about running towards the ball; but this is because his rules have been applied so many times, that he no longer

must actively retrieve them. He is still unknowingly following those rules. The authors do not point out that when they speak of intuition, they are actually referring to rules reapplied until they become habits. A person from a baseball-free society would not know how to catch the ball either; but this doesn't mean he cannot learn through rule application.

They also mislead us by stating:

we cannot program computers for context. For instance, we cannot program a computer to know simply that a car is going "too fast." The machine must be programmed in a way free from interpretation -- we must stipulate that the car is going "20 miles an hour," for example. Also, computers know what to do only by reference to precise rules, such as "shift to second at 20 miles an hour." Computer programmers cannot use common-sense rules, such as "under normal conditions, shift to second at about 20 miles an hour." (Dreyfus and Dreyfus 1986, 51)

They can, of course, if the definition of "normal conditions" is specified. A person would not know what "too fast" meant out of context either. Driving 50 miles an hour on a highway may not be fast, but driving the same speed on your driveway where your children are playing *could be risky!* We must specify the conditions for a computer just as we would for a human.

• **Problems with current expert system tools**

The programming languages that are used to build expert systems also present some problems and limitations for the knowledge engineer. In general, most of them do not aid in the tedious jobs of knowledge acquisition and knowledge base refinement (Waterman 1986, 182). While some systems do have utilities for the entering of data (such as the *Knowledge Engineering Environment*, which provides a graphical, mouse-driven interface), they often require data to be entered in the stylized language of



the programming environment rather than in natural language. Furthermore, these knowledge bases often take quite a long time to build.

Jon Doyle (1984) believes we might be able to retain the benefits of expert systems and remove some of their drawbacks by employing the same techniques (of protocol analysis, knowledge formalization, etc.) in a non-computer-based environment. This would eliminate the necessity of using a programming tool which is not suited to the application. It would also allow the knowledge base to take a natural language form understandable by anyone, not merely by the elitist group of knowledge engineers. Furthermore, it would restrict us from putting undeserved faith in software which, unbeknownst to us, may not be completely bug-free. Doyle feels we should be more concerned with formalizing and conveying knowledge than with the computer implementation of knowledge manipulation algorithms.

## *Guidelines for Knowledge Acquisition*

Some of the guidelines for verbal knowledge acquisition are presented here as described in Anna Hart's Knowledge Acquisition for Expert Systems (1986, 49-52). I have augmented some of her general ideas with reference to issues related to the domain of design.

- *Be specific, not general, when asking questions.*

It is easier for the expert to recall and describe actions that are specific, interesting or unique. For our purposes, this can be taken one step further when we consider that it may be easier for design experts to access descriptive explanations of specific actions than to formulate general rules. For example, it is easier for a designer to explain why two particular photos look awkward together in the same composition than it is for him to state (out of context) why some photos do not look appropriate together in general.

- *Do not impose alien tools.*

The expert should be allowed to convey information in the medium most natural to him so that the description of his expertise will not be unnecessarily altered. This concern was reflected in a PackIT acquisition session where the expert was permitted to describe her processes in an environment somewhat similar to her own workspace. (While her actual work area would have been an ideal location, it was not available at the time.)

- *Do not interrupt.*

Interruptions are distracting to the expert who must concentrate not only on how he performs his job, but also on justifying his actions in verbal form.

- *Record information.*

Certain information that may appear trivial to the knowledge engineer at one point may prove to be of great value later. Ericsson and Simon (1984, 3-4) describe the difference between “hard” and “soft” data. In general, data which is hard is not subject to interpretation – its content is clear. Soft data, on the other hand, is sometimes ambiguous – different observers might extract different meanings when examining the same event. Soft data can be dangerous because it appears hard when formalized on paper, even though the analysis was highly subjective. The authors explain that before the use of tape recorders, experimenters paraphrased interviews omitting what they incorrectly thought was “unimportant.”

- *Listen to the way the expert uses knowledge.*

It is not only the rules themselves that are important, but also the way the expert employs them that is valuable.

- *Present questions carefully.*

The knowledge engineer should be careful about how questions are presented. The same question may produce vastly different responses depending upon its structure. Schuman and Presser (1981, 281) compiled a number of polls which reveal some intriguing insights into human nature. One particular question received surprising

responses by the simple replacement of one word:

*Do you think the US should forbid public speeches in favor of Communism?*

YES (forbid) 39.9%

NO (not forbid) 60.1%

*Do you think the US should allow public speeches in favor of Communism?*

NO (not allow) 56.3%

YES (allow) 43.8%

Note that although the actual meanings of the two sentences are identical, the difference between the responses is quite substantial. The authors suggest that, among other things, the word “forbid” sounds harsher than “allow.”

---

## *The Graphic-Object Class*

The class `graphic-object` is a large structure containing the information which describes each graphic object of the package. The class is defined as follows:

```
(define-type graphic-object
  (:var name      (:type simple-string))
  (:var xpos      (:type fixnum))
  (:var ypos      (:type fixnum))
  (:var width     (:type fixnum))
  (:var height    (:type fixnum))
  (:var parent    (:type list))
  (:var children  (:type list))
  (:var type      (:type simple-string))
  (:var category  (:type simple-string))
  (:var xcrop     (:type list))
  (:var ycrop     (:type list))
  (:var squash    (:type float))
  (:var brightness (:type float))
  (:var saturation (:type float))
  (:var importance (:type float)))
```

The first five entries are self-explanatory. The `parent` and `children` attributes indicate which other objects are directly connected to any particular one in the positional relationship tree. They are declared as lists in order to hold the required information:

```
(=> object-1 :parent) = `(object-2 below right-just 0.8)
(=> object-2 :children) = `((object-1 0.8) (object-3 1.0))
```

The first line states that `object-1` is positioned below and right-justified with its parent, `object-2`. The 0.8 (of a possible 0-1 range) indicates the importance of retaining this relationship in the layout. The second line says that `object-2` has two children, `object-1` (we already knew this) and `object-3`. The relationship

importance values are specified here as well. Although these values are not currently used by PackIT, they could be useful in telling the inference engine that if it needs to make room on the package, for example, then removing the parent/child link for some of the objects may not be an acceptable method to accomplish this.

The `type` attribute specifies whether the object is textual or strictly graphical in nature. `Category` tells us what kind of package entity it is: company name, product picture, company motto, etc.

`Squash` indicates the maximum amount that the object may be altered in shape (vertically or horizontally). Some objects, like simple geometric shapes, may be altered more than others. In general, however, most of the objects' default squash factors are set quite high in the PackIT system (i.e. little reshaping is permitted). The calculation of how much an object is squashed is primarily used to determine how much room is needed on the package, so that the inference engine will know to seek some other resolution.

`Xcrop` and `ycrop` each have pairs specifying the left/right and top/bottom cropping limits respectively, as described in chapter 10. The brightness and color saturation of an object are calculated by a pair of functions and are entered here. Finally, the `importance` attribute indicates how important it is for the object to appear on the package at all.

## Bibliography

Badshah, A. "GRID – Graphic Intelligence in Design: An Expert Layout System." M.S. thesis, MIT Media Laboratory, Jan. 1987.

Brachman, R. et al. "What Are Expert Systems?" In *Building Expert Systems*. eds. Frederick Hayes-Roth, et al. Reading, MA: Addison - Wesley Publishing Company, 1983, 31-57.

Buchanan, B. "Expert Systems: Working Systems and the Research Literature." *Expert Systems*, Jan. 1986, 32-51.

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

Citicorp. *Citicorp/Citibank Identification Standards*. New York.

Dondis, D. *A Primer of Visual Literacy*. Cambridge, MA: The MIT Press, 1973.

Doyle, J. "Expert Systems Without Computers Or Theory and Trust in Artificial Intelligence." *The AI Magazine*, Summer 1984, 59-63.

Dreyfus, H., and Dreyfus, S. "Why Computers May Never Think Like People." *Technology Review*, Jan. 1986, 42-61.

Ericsson, K. and Simon, H. *Protocol Analysis, Verbal Reports as Data*. Cambridge, MA: The MIT Press, 1984.

Feigenbaum, E., Buchanan, B., and Lederberg, J. "On Generality and Problem Solving: A Case Study Using the DENDRAL Program" In *Machine Intelligence 6*, eds. Bernard Meltzer and Donald Michie. New York: American Elsevier Publishing Company, Inc., 1971, 165-190.

Feiner, S. "APEX: An Experiment in the Creation of Pictorial Explanations." *IEEE Computer Graphics & Applications*, Nov. 1985, 29-37.

Gero, J.S. ed. *Knowledge Engineering in Computer Aided Design*. Amsterdam: Elsevier Science Publishers, 1985.

- Gill, B. *Forget all the rules you ever learned about graphic design*. New York: Watson-Guption Publications, 1981.
- Harmon, P. and King, D. *Expert Systems*. New York: John Wiley & Sons, Inc., 1985.
- Hart, A. *Knowledge Acquisition for Expert Systems*. New York: McGraw-Hill, 1986.
- Hayes, J.E., Michie, D., Y-H Pao, eds. Machine Intelligence vol.10 *Intelligent Systems: Practice and Perspective*. Chichester, England: Ellis Horwood Limited, 1982.
- Hayes-Roth, F., Waterman, D.A., Lenat, D.B., eds. *Building Expert Systems*. Reading, MA: Addison - Wesley Publishing Company, 1983.
- Honeywell Inc. *Honeywell Corporate Identification Manual*. 59-5057. Minneapolis, Aug. 1985.
- International Business Machines Corporation. *The IBM Logo, Its use in company identification*. FO4-0002. Armonk, New York.
- Japan Creators' Association. *Noah Directory of International Package Design*. Kanagawa, Japan, 1985.
- Japan Package Design Association, ed. *Package Design in Japan*, vol.1. Tokyo, Japan: Rikuyo-sha Publishing Inc., 1985.
- Kaplan, J. "The Industrialization of Artificial Intelligence: From By-Line to Bottom Line." *The AI Magazine*, Summer 1984, 51-57.
- Kepes, G. *Language of Vision*. Chicago: Paul Theobald and Co., 1961.
- Kirsch, J. and Kirsch, R. "Computers as Art Connoisseurs." The Media Lab Forum, MIT, Cambridge, MA. 11 Mar. 1987.
- Lawson, B. *How Designers Think*. Westfield, NJ: Eastview Editions, Inc., 1980.



- Lenat, D. et al. "Reasoning About Reasoning" In *Building Expert Systems*. eds. Frederick Hayes-Roth, et al. Reading, MA: Addison - Wesley Publishing Company, 1983, 219-39.
- McDermott, J. "R1 The Formative Years." *AI Magazine*, Summer 1981, 21-29.
- Moses, J. "MACSYMA – The Fifth Year." *Eurosam Conference, Stockholm*, Aug. 1984, 105-110.
- Moses, J. "Algebraic Simplification: A Guide for the Perplexed." *Communications of the ACM*, Aug. 1971, 527-37.
- Müller-Brockmann, J. *Grid Systems in Graphic Design*. New York: Hastings House Publishers, Inc., 1981.
- Nilsson, N. "Artificial Intelligence, Employment and Income." *The AI Magazine*, Summer 1984, 5-14.
- Prerau, D. "Selection of an Appropriate Domain for an Expert System." *The AI Magazine*, Summer 1985, 26-30.
- Purcell, P. A. et al. "A Strategy for Design Research." In *Basic Questions of Design Theory*. Amsterdam: North-Holland Publishing Company, 1974, 75-93.
- Rand, P. *Thoughts on Design*. New York: Van Nostrand Reinhold Company, 1970.
- Reese, T. *The Best in Packaging*. Bethesda, MD: RC Publications, Inc., 1986.
- Sacharow, S. *The Package as a Marketing Tool*. Radnor, PA: Chilton Book Company, 1982.
- Schmitt, P. *Packaging Design 2*. Locust Valley, NY: PBC International, Inc., 1985.
- Schuman, H., Presser, S. *Questions and Answers in Attitude Surveys*. New York: Academic Press, 1981.
- Sheil, B. "The Artificial Intelligence Tool Box." In *Artificial Intelligence Applications in Business*. ed. W. Reitman. Norwood, NJ: Ablex Publishing Corp., 1984, 287-95.

Shortliffe, E. "Details of the Consultation System." In *Rule-Based Expert Systems*. eds. B.G. Buchanan and E.H. Shortliffe. Reading, MA: Addison-Wesley, 1984, 78-132.

Waterman, D.A. *A Guide to Expert Systems*. Reading, MA: Addison - Wesley Publishing Company, 1986.