NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, tests publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30.

Canada

Development of A User-Friendly Menu Driven Intelligent Front-End System For Large Frame Based Knowledge Engineering Tools

Allan Gary Kowalski

A Thesis

in

The Department

of

Computer Science

Presented in Partial Fulfillment of the Requirements for the degree of Master of Computer Science at Concordia University Montréal, Québec, Canada

March 3, 1988.

ISBN 0-315-41650-5

ABSTRACT

Development of A User-Friendly Menu Driven Intelligent Front-End System For Large Frame Based Knowledge Engineering Tools

Allan Kowalski

As part of an effort to achieve a solution to the classic bottleneck problem related to the task of knowledge engineering, a front-end system was developed that would allow the development of a large, complex and dynamic set of taxonomies and related hierarchical structures, as a first step towards the development of full-scale expert systems by domain expert without a strong A.I. or knowledge enginering orientation.

Dedication:

To my parents, Chaim and Chana, for their steadfast and everlasting support.

Acknowledgements

I wish to express my appreciation to my co-supervisor Dr. Renato DeMori for his advice and encouragement and academic support through the various stages of development of this thesis.

In addition I wish to thank my co-supervisor, Dr. Paul Fazio for providing the professional and academic support that greatly facilitated the development and testing of various aspects of this thesis.

Finally I also wish to thank the Centre Recherche Informatique De Montreal for making available some of the crucial resources without which the successful development of this thesis would not have been possible.

Table of Contents

List of Figures

CHAPTER I - INTRODUCTION

1.1 Background

This report describes a knowledge engineering tool called ESCHER (Expertise Structure, Class and Hierarchy Engineering Resource) developed as part of a Master's Thesis in Computer Science. The tool development was part of a research effort at the Center for Building Studies, under the direction of Dr. Paul Fazio, to investigate the applicability of expert system technology in the area of building engineering.

1.2 Expert Systems in Building Engineering:

Expert systems tend to fall into one of ten basic categories which are:

1) Monitoring
2) Interpretation
3) Prediction
4) Diagnosis
5) Design
6) Planning
7) Control
8) Instruction
9) Repair
10) Debugging

It is now a widely held belief that the interest of the engineering community at large in the applicability of expert systems is extremely high [14].

Expert systems are currently being developed for many industrial applications in the Construction Industry. The applications range from representing fire regulations, (ACE), selection of an appropriate air-conditioning systems, advice on air-to-air heat recovery, and plant fault diagnostic routines, to name just a few.

In the field of Computer Aided Design, the U.S. NBS (National Bureau of Standards) is initiating standards for the interface of Coded Standards and (C.A.D.) Packages, while at Carnegie-Melon University work progresses steadily on Kadbase.

At the university of Sydney, a program to aid in optimizing the room lay out is being developed and, finally, Johnson Controls continues work on the JC/85/40 Building Automation Systems [8].

The development of ESCHER stemmed from an initial study of the present day strategies involved in the building of large scale real-life expert systems. Associated with these strategies are a set of problems which have been clearly identified as standing in the way of large-scale practical industrial development of expert systems.

ESCHER is an attempt to address some of the key problems and offer a practical first step towards a solution. Such a

solution is essential if expert systems are to become accepted as a common, practical and useful tool of information management and decision making.

The development of real-life expert systems is a complicated and involved endeavor that requires the transferring of knowledge from a human expert to an expert system. This transfer process is called **knowledge engineering** and, even for expertise which is already in a structured form [22] and well suited to such a process, it is a long and involved task.

The field of knowledge engineering has been around for more than a decade. In that time there have been many significant developments and improvements.

These improvements have, for the most part, been in two areas; **knowledge representation** and the **user interface** techniques. Improvements of both natures are important to the relevance and nature of the ESCHER package and therefore a brief review follows.

CHAPTER II - INTRODUCTION TO KNOWLEDGE ENGINEERING

2.1 The Evolution of Knowledge Representation Techniques

The element which characterizes an expert system (ie. which makes it "expert") is the expertise or knowledge of the information related to the field about which the system is an expert.

The issue of how best to represent this knowledge is the subject of much debate and study at the present time. It is because these sytems are centered around this knowledge that they are referred to as "knowledge-based systems" or "knowledge systems" . Not only is a great deal of information incorporated into this knowledge, but it is information of different types (ie. procedural, taxonomical, etc . .).

All of these kinds of information must somehow be stored in such a way as to form a *cohesive, logically interconnected* collection of available knowledge which represents as accurately as possible the domain of expertise in question. A collection of these various kinds of knowledge is called a **knowledge base.**

The first expert systems developed were not built through the use of specialized tools but rather, with the foremost AI language of the period, namely *lisp*. Using lisp, various modules were created to perform all of the tasks required. The various major modules are shown below in fig. 2.1 .

The key components are the inference engine and the knowledge base.

The *knowledge base* contains a set of structures or a combination of such sets which represent a model of expertise in some area and tend to be more or less static from one user to the next. The knowledge can be said to contain the rules of expertise of a particular domain.

The *data base* (not to be confused with a data base management system), is a file containing specific information relating to an individual case of the particular end-user making use of the expert-system and, of course, differ from one user to the next. The data base can be said to contain the facts of a particular case.

For example a knowledge base might contain information about the various attributes of wines and the rules about matching certain wines with certain meals, whereas a data-base would contain information about which wines the user has in his wine cellar and a description of the meal with which he wishes to add a wine.

The *inference engine* is that part of the system which processes all of the information in both the knowledge and data base thereby making it possible, for the expert system to draw its conclusions. This is done by matching the knowledge available against the facts of a particular case in order to determine which parts of the expertise are relevant and should be applied.

In addition there is often a *user interface* facility which is responsible for controlling the means by which the user communicates with the system (ie. inputs facts and questions and obtains responses).

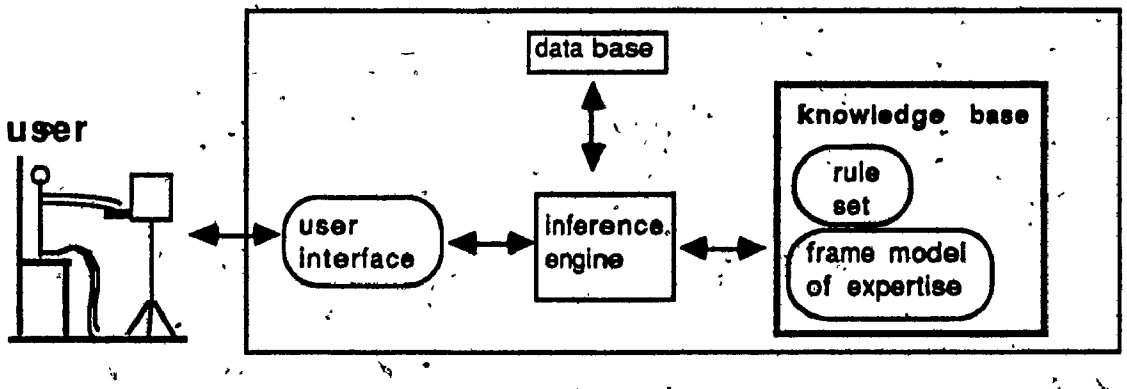


Fig. 2.1 Components of an Expert System

Development of these early expert systems revealed the usefulness of *specific data structures* built for the purpose of representing knowledge or expertise, in the database.

PREDICATE LOGIC:

One of the first such structures which was implemented as the unit of expert systems development was the predicate-logic statement. Just as sets of rules were used to define a knowledge base so, too, could a set of such statements be used. System based on such units developed into the PROLOG shells of today. Although Prolog is very useful in itself, and was initially selected as the tool of choice by the Japanese for the fifth generation project, it is now generally accepted to be lacking as a complete tool.

Prolog offers "facts" and "rules" where rules consist of fact based premises and logical connectives. The latest developments in PROLOG include;

- interfaces to real world data-bases
- concurrent processing compatible with parallel machine architecture

- increasingly sophisticated natural language capability

- incorporation of meta-level reasoning

- functional programming capability [20][21]

Prolog has been used as the basis for much work in A.I. since it has some very attractive features for the building of expert systems. Such features include its ability to work with facts and rules, its grammar formalism and of course its ability to "infer". Many systems, such as OSCAT,[12] have already been built around it.

In addition, there have been some recent attempts to develop new languages based on predicate logic, but none have yet gone beyond the research stage [11], [36].

Rules

Another of the first such structures was the so called if/then rule similar to the if/then statement in PASCAL. The rule consists of two parts. The first part is a premise (ie. the IF clause with which a set of one or more conditions are associated. The second part of the rule, the THEN clause, is the set of actions to be implemented if all of the conditions

in the premise are satisfied. The basic syntax of a rule is shown below in fig. 2.2 .

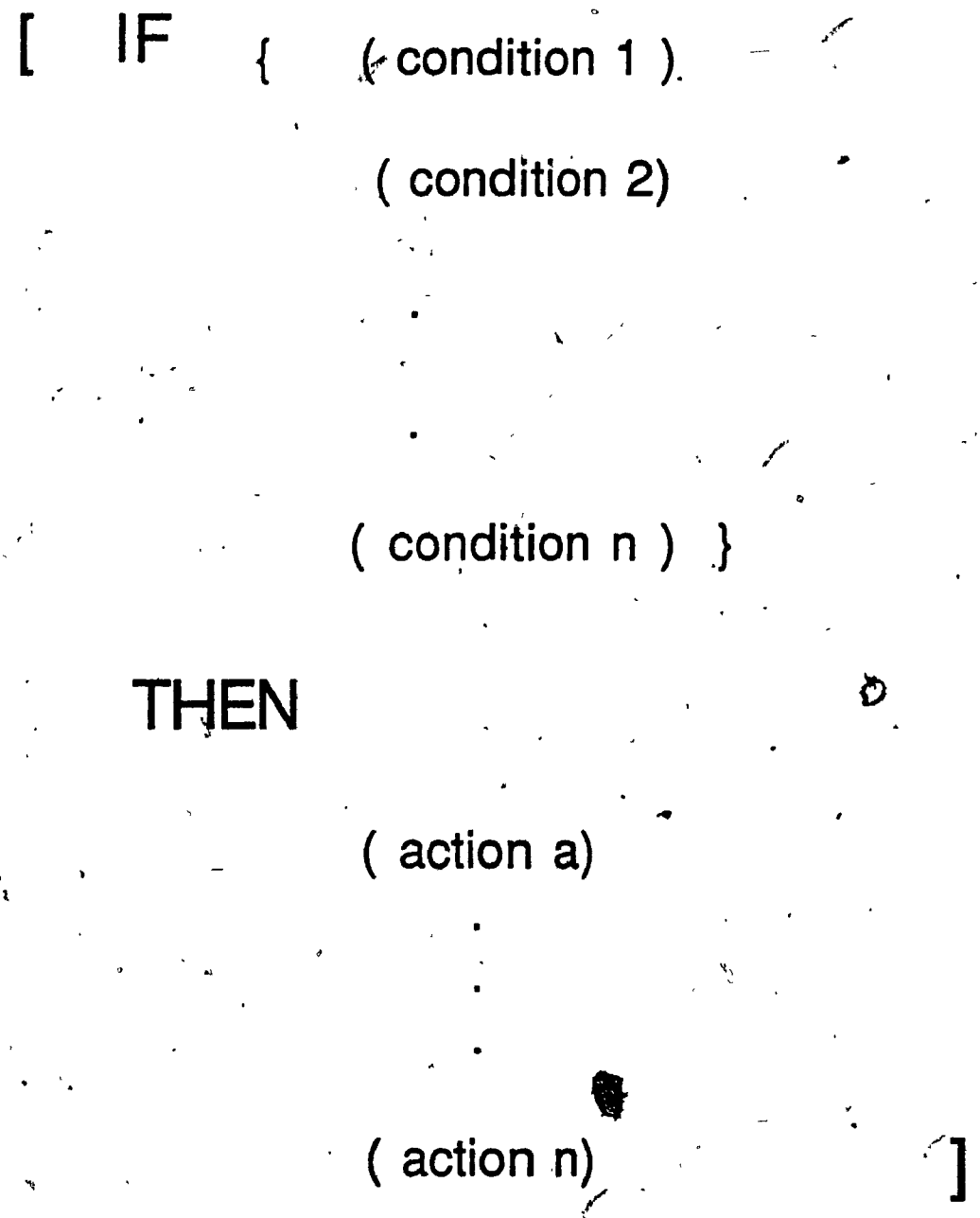


Fig. 2.2 Sample Rule Syntax

Once the usefulness of rules or a specific knowledge representation structure had been established, tools, especially geared to process such structures were developed as the first expert system development tools or shells. These contained the previously mentioned inference engines and user-interfaces and allowed the user to develop his own knowledge base.

Forward vs Backward Chaining :

The concepts of "forward chaining" and "backward chaining" are very crucial to rule-based expert systems and so a basic explanation of the two opposing strategies is called for.

In forward chaining, at the start of each cycle, a note is made of all facts that are known to be true. A check is then made through the list of rules to see which rules have conditions which are satisfied by the set of facts. Of all these *satisfied* rules, one and only one, is chosen as the rule whose actions will be executed (the selection of which of the satisfied rules will be executed usually involves several different criteria which depend on the particular system being used. This process of rule selection is referred

to as *conflict resolution* . It is important to note that such actions may result in new facts being true or some formerly true facts becoming untrue. After these actions are executed the cycle begins anew.

In contrast,backward chaining works in the opposite manner. It basically involves working backward from the desired goal to the starting point. At the start of each cycle, the system "matches" those rules which, when satisfied, will give the desired goal(ie. action). The conditions of **all** those rules, then, become the new goals (one at a time). This cycle repeats itself until the conditions can be satisfied (or a dead end is reached).

The advantages of the production rule system is that since it allows the user to specify control of actions, it is much better suited towards ease of user/knowledge manipulation (ie. it is much easier to make the system do what you specifically want it to do).

On the other hand the disadvantages are that, in contrast to frames, production rules cannot be related to one another in any way (ie. there is no inheritance).Thus, actual existing relationships between rules cannot be accurately represented. Furthermore, unlike frames,production rules do not allow for the description of domain objects.

The first such system was EMYCIN, a spin-off of the application MYCIN. It captured the essence of the MYCIN inference strategy (hence the name **E**ssential **MYCIN**). It

offered a package containing everything but the actual knowledge which was to be supplied by the developer who had purchased the product in the form of Mycin type rules. One of the earliest systems developed in this way was SACON [7],[26], an expert system that acts as an advisor to the user on the use of the MARC finite element code for structural analysis.

Development and improvement of rule based systems followed for over a decade resulting in such systems as SPERIL-I, SPERILL-II [23], and PROSPECTOR. In each case a different approaches was taken in designing the system specific to the needs of the individual application. As a result, the rule approach was analyzed quite thoroughly and was found to be very useful, perhaps even more so than predicate logic. It provided a simple mechanism for the generation of explanations and prompting the user [29]. However, it was also found lacking in several key areas, [1] [5] the most critical ones being:

1 - it was inconvenient for representing certain complex structures and objects

2 - it did not allow for the separation between control/procedural knowledge and domain expertise.

There is a definite limit to the amount of knowledge which can be conveniently represented in a single rule. This means that knowledge exceeding this amount must be represented by a set of rules which often leads to confusion or, with an incorrect implementation, erroneous results.

In addition, rules tend to impose on a knowledge base a structure revolving around a collection of small homogeneous "chunks of knowledge", rather than something more hierarchical and taxonomical in nature [1]. Aside from the inefficiency, it also allows a greater degree of freedom in modifying, adding, deleting or neglecting to add an individual piece of knowledge independently of all the other pieces. *This can be an invitation to disaster.*

An example of this is called "semantic conflict", and was noted as far back as the development of MYCIN. It involves, as an example, the implicit contradiction of two rules, one which concludes the site of the culture is sterile and one which concludes that the site of the culture is non sterile. As intuitive as the contradiction may be, it is very difficult to represent in a general manner which will handle every such occurrence.

In addition, there was a definite upper limit to the amount of information that could be represented as a collection of rules. The reason for this has to do with a part of the rule matching cycle called **conflict resolution**. Conflict resolution involves the selection of a rule to be

fired from a set of rules, all of which have been determined to be satisfied or *matched*. The set of rules which are matched is called a ***conflict resolution set***.

This selection follows a strategy that is particular to both the shell and the configuration of the shell. Strategies are available in varying degrees of complexity and flexibility.

As more and more rules get added, the average size of the conflict resolution set gets larger and larger. As the size of a conflict resolution set grows it becomes increasingly difficult to control, (ie.to correctly predict), for all cases, which rule will be selected for firing. When a point is reached at which a correct prediction can no longer be guaranteed, the expert system is no longer controlled and **is of no use**.

This becomes a critical issue in the fields where the process of production is especially dynamic, such as architectural or building design.

" *Knowledge Bases associated with architecture will inevitably be very large. The process of capturing knowledge will involve refinement and modification, even while the system is in use. An expert system must, therefore, be amenable to change and growth over time* " [27].

An example of the state-of-the-art, high-gloss, rule-based, pc-based expert system shell is Personal Consultant Plus, marketed by Texas Instruments Corp.

The basic structure of PC plus is the classic IF/THEN rule, but the development of these rules is aided by a sophisticated editing, debugging and primitive structuring capability. This, at least in theory, allows the domain expert to enter the substance of the rules without worrying about syntax or structure. Unfortunately, the reality does not bear this out [37].

While the entering of any particular rule is relatively easy, the resulting collection of rules does not usually end up producing the desired results. This is due largely to the fact that no structure to the rule set has been provided by the user and the system, using its own primitive mechanism has provided an inappropriate one.

In other words, despite its sophisticated graphics the system does not aid in knowledge acquisition but, rather, aids only in knowledge entry. **The need for knowledge engineering by qualified personnel still remains a critical element.**

Frames:

A partial solution to these problems (at least in so far as it reduces the number of rules required) came from the development of another knowledge representation structure called the **frame.**

The frame, itself, is a data-structure used to represent some object or concept. Each frame is associated with a set of descriptive components, usually called slots, that help to describe the object or concept associated with the frame. In turn, each slot may have associated with it one or more values that help to describe the particular value or values of the slot. Although the concept is very similar from one system to another, the generic concept of a frame may be associated with one of several names (ie., "concepts", "prototypes" "schema", etc . .)

An example of a typical frame is shown below in fig. 2.3. In this example the frame represents the concept of a building with which is associated the attributes of a certain number of walls, a certain number of windows, a certain colour(on the outside of the building), a certain number of stories and a certain number of roofs which is always 1. The frame was a significant improvement over the standard rule-based systems in that it allowed a method that is much more convenient in representing certain types of domain knowledge.

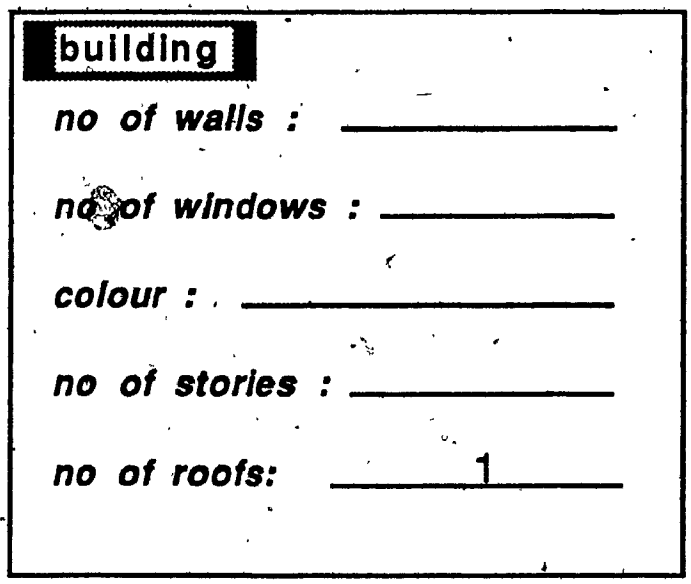


Fig. 2.3 Sample Frame

The frame, in conjunction with its associated slots, (and their associated values,) allowed for the representation and description of objects in a more convenient and descriptive form, while, in conjunction with frames, it allowed for a better and more clearly defined separation of the two distinct types of knowledge. The frame served as one of the basic representation-structure of such hi-key expert systems as XCON (originally named R1).

A variation of the frame structure, called the object-attribute-value tuple (or O.A.V), was sometimes used instead. The most well known examples of this are S.1 and Copernicus.

Another improvement provided is the capability of implementing constraints through such slots as " value-class " and "cardinality" . These constraints provide an automatic mechanism to guard against erroneous descriptions or values.

Clearly, though there remained some serious deficiencies, chief among them, the fact that while rules and frames allowed for the representation of individual pieces of knowledge they were clumsy means of providing a means of representing the nature of the connections between these individual pieces.

In addition, frames were ill-suited to the manipulation of knowledge by the user. Ad-hoc solutions of the past consisted mainly of writing commonly used procedures and

functions in more standard languages such as lisp, and then attaching them to slots in appropriate frames.

Frames were also inadequate from the point of view that, while capable of describing *necessary* attributes, they were incomplete for the task of describing *sufficient* attributes. For example, the frame, " elephant " may contain the attributes mammal and four-legged but not all four legged mammals are elephants.

It was, therefore, clear that the representational capabilites of frames, slots and values was not up to the task of modelling the complexity of real life situations.

The next generation of shells contained a series of improvements designed to address those issues.

Semantic Networks and Inheritance:

The next generation of shells contained representational capabilities that allowed for the implementation of semantic networks(ie. a set of units connected to each other). These networks were composed of frames that were connected to one or more other frames. These connections were referred to as relations and the two frames connected in such a manner are said to be related.

Relations provided the means by which pieces of knowledge could be tied together. In addition they provided the means by which a relation, by being treated as knowledge

itself, allowed for a more realistic representation of the knowledge of relations between certain objects. Relations also allowed for the feature of *inheritance*.

Inheritance is one of the most important features in large frame-based knowledge engineering tools.

Most, if not all, areas of expertise involve knowledge which is, in one form or another structured into taxonomies (ie. a form of hierarchical stucture). It is inheritance more than anything else, which allows the direct construction of taxonomies as well as other heirarchical structures. In addition, it allows the inter-level connection between various taxonomies so as to produce a representational model of expertise.

Inheritance is a mechanism by which a second frame, related to first frame, will "inherit" or automatically obtain the attributes of the first frame. In this way, certain properties can be passed down, so as to more accurately (realistically) represent the domain of knowledge. In specialized cases, customized relations can be built to also allow properties to be passed upwards, as required.

An example is shown below in fig. 2.4. in which the frame "little red school house" is related to the frame "building" by the **is-a** relation (ie. the little red school house is a building). As a result of this relation the little red school automatically inherits the attributes of

walls, windows, colour, stories and 1 roof. The value red associated with the slot colour is not inherited but *local*.

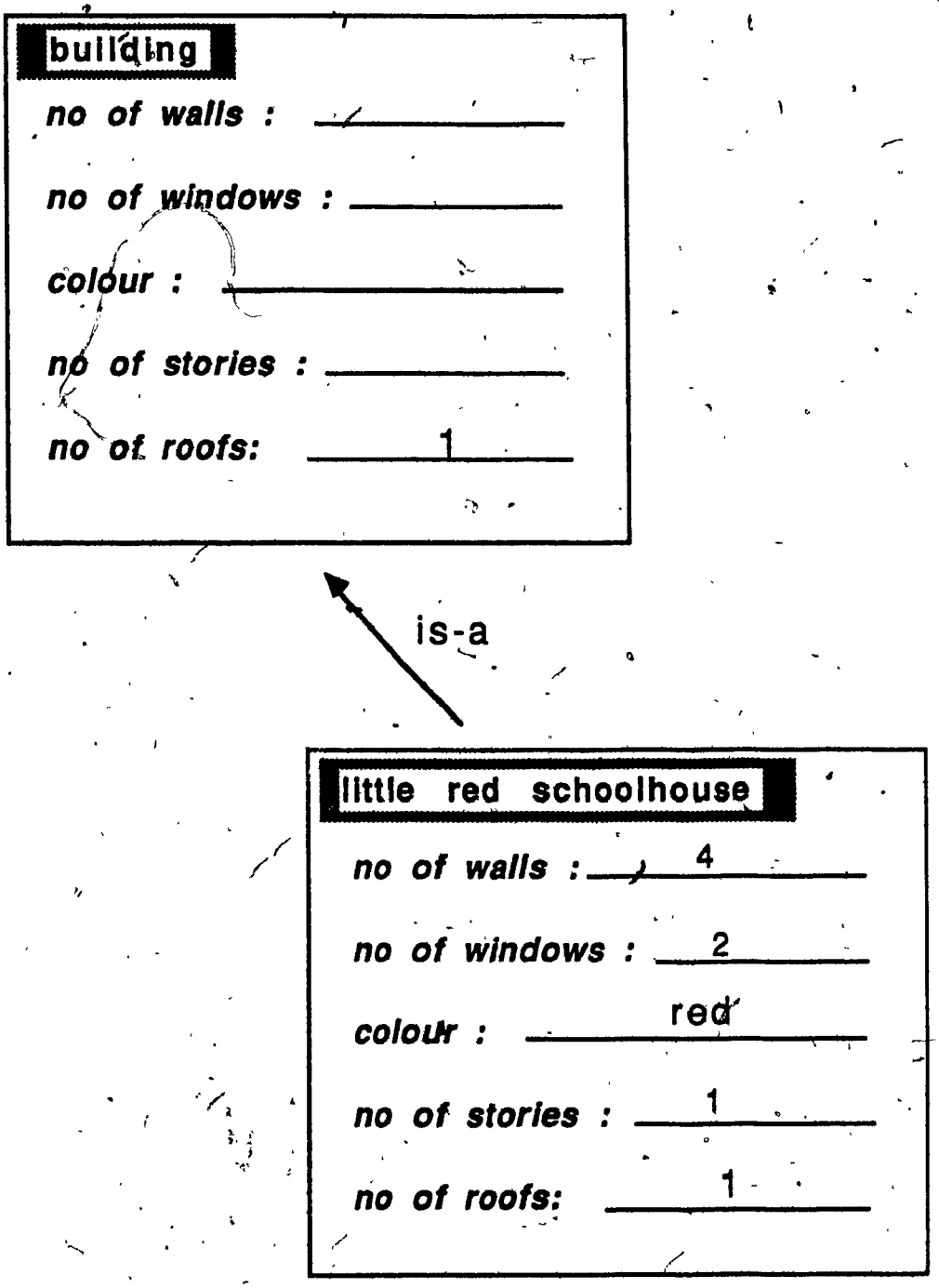


Fig. 2.4 Example of inheritance

To be sure, there has been some criticism of inheritance as being too absolute, in that it does not distinguish between cases in which inheritance is universal and cases in which it is not. In real-life, members of certain groups which may usually inherit all properties, may in certain cases not do so and such exceptions are an important part of many domains of expertise.

This produces the need for inefficient and messy ways to handle the distinctions and special cases. All in all however inheritance is generally considered to be a very useful feature.

All of these features which make possible the building of taxonomies and various other hierarchical structures were used in HI-RISE (in the area of structural engineering) [23], THINGLAB [24] and SKETCHPAD [25]. A collection of these taxonomies and inheriting related sets of frames is called a semantic network, because it expresses the semantics(ie. relationships between objects or ideas) of the knowledge. The largest. most sophisticated, shells incorporating this philosophy are ART [13] and Knowledge-Craft [2]..

In addition, an extention of this approach called object-oriented programming, became available leading to products such as KEE. [42] and LOOPS [29]. In object-oriented programming, frames can be used to represent actual procedures or even groups of procedures, and taxonomies of procedures can, thereby, be built up. In this way, an entire

set of procedural calls can be developed as part of the knowledge base.

This representation of **relations** is important for 2 reasons:

1 - It allows for the building of taxonomies and other hierarchical structures to represent large overall concepts. *In other words makes possible the* ***represention of the relation*** *of certain objects to each other.*

2 - It allows for the representing of knowledge in the **form of relations**. That is to say that the specifications of varying type of relations is in itself knowledge that can now be represented. ***It is this second factor that helps to differentiate the more powerful shells from the rest.***

The ability to build customized relations allows for the representation of knowledge to such a high degree of complexity that it becomes feasible to represent enough knowledge in a knowledge base to *make real-life applications feasible.* For example, implementing a design system requires the ability to represent very dynamic models, the state of which are related to various factors involved in the design process itself [31].

This accomplished, the latest state-of-the art frame-based shells now boast such sophisticated capabilities as:

- customizable slots
- customizable relations
- customizable facets

Customizable slots allow for the specification of demons, inheritance behaviour and various restrictions. For example a slot called " material " could be customized so that the only legal values which could be assigned to it would be those items which had been already described, within the knowledge base, as materials.

Customized relations allow frames to be connected to each other in a variety of ways. A customized relation is described by a frame containing all of the specifications relating to the relation. For example the relation " adjacent " could be created to describe structures in a building that were positioned next to each other. The fact that these parts are " adjacent " would mean that they share some features such as a certain space but not others such as materials from which they are composed.

Customized facets revolve around the nature of the information represented in a value of a slot. The information can be passive or active and if active it can be customized to suit the particular needs. For example a facet could be created to implement load constraints on a structure so as to automatically trigger a flag if the value of the load exceeds a certain critical amount.

The structures used in building models of expertise, while referred to by unique terminology, have many common features and employ common priniciples, to a high degree. This is especially true for the larger frame based or object-attribute-value based systems.

It is inheritance more, than anything else which allows the direct construction of taxonomies and hierarchical structures. In addition, it allows for the inter-level connection between various taxonomies which produces a representational model of expertise.

Hybrid Structures:

Given that the two general knowledge representation strategies have strengths and weaknesses which are highly complementary, it is logical to expect that some sort of combination of the two might have the benefits of both combined in one. Such an attempt has been made several times and often, with significant success (eg. ART and CENTAUR).

In such systems, frames allow for full description of objects which are referred to in the rules. In addition, the ability of frames to incorporate hierarchical structures makes it possible to structure the rules in such a way as to facilitate the accessing of appropriate rules. In other words the systems can be set up in such a way so as to easily

control which rules are brought into play at which stage (ie. under which conditions).

In such hybrid systems, the production rules may be represented either in standard condition/action form or in the form of frames. There are several advantages to such frame-rules:

- the rules can be easily grouped into appropriate classes and sub-classes

- the rules can be described more fully by means of descriptive slots

- functions or constraints can be attached to particular slots to act as checks or alarms which indicate special problem cases or situations.

These features become more important as the number of production rules grow. Rule classification can greatly aid in control of rule behaviour. Rules can be organized into a logical organization based on their functionality.

The structures used in building models of expertise, while referred to by unique terminology have many common features and employ common principles, to a high degree. This is especially true for the larger frame based or object-attribute-value based systems.

Clearly, although there are some expert system projects in which more general languages such as LISP or PROLOG are

still being considered as the language with which to build [30], commercial shells now offer a state of power and flexibility which approaches much more closely the requirements to model real-life situations. This is especially true in the case of modelling design functions where the semantic network capability becomes especially crucial [31]. This, same, previously mentioned power and flexibility of today's state-of-the art shells is at the same time, an achilles heel because it involves factors related to the methodology of building and expert system.

2.2 Tools :

In this sub-section we shall briefly discuss some specific examples of expert knowledge representation tools encompassing the various *methods* discussed previously. As these methods have already been discussed, we will focus our attention (in this section) on those aspects unique to each particular system.

Knowledge-Craft/SRL :

Knowledge-Craft was developed at the Carnegie Mellon Institute. SRL (Schema Representation Language) is the purely frame-based knowledge representation system which, with OPS5, constitutes the inferencing mechanism of Knowledge-Craft. Of course, it has all of the features that typify such systems but, in addition, it has the following special features and characteristics worth noting:

- frames are referred to as "schema"

- "the approach taken in SRL is that all relations are slots through which information can be transferred bi-directionally" (ie. if frame "a" is described as is-a "letter" then letter will be automatically described as is-a+inv "a" and vice-versa)

- all effort was made to make SRL as user "definable" as possible (ie. user can define relations and associated inheritance), slots, facets and meta-values

- a slot can be assigned any value (within the user defined constraints) where value can be any lisp expression

- any part of a schema may have *meta-information* associated with it (it is said to be "attached") in the form of a "meta-schema" (a schema containing only meta-information)

- the entire store of knowledge is structured in a hierachical tree of" mini-databases" called contexts. Every schema must be stored in a context.This hierarchy allows each context to inherit from its parent context

Relations :

Relations are represented as slots. An example is shown below

```
{{ dog
     Is-A: Mammal }}
```

Translated from SRL this means that "dog is a mammal". This entitles dog to inherit all of the attributes (ie. slots and their values) associated with mammal.

Functions :

- Standard functions are supplied by SRL to perform the commonly required tasks. Some examples are:

schemac - standard function used to create schemata

schemad - checks to see if a particular schema exists

mschema-p- returns the meta-schema associated with the specified schema if one exists

mschemad - checks to see if a meta schema exists which is associated with a specified schema

slotc - creates the specified slot in the specified schema

slot - lists all slots defined in the specified schema

slotd - deletes a specified slot from a specified schema

Other functions provide various types of access to slots,meta-schema,meta-slots,etc.... so as to allow vitually any kind of schema manipulation a user might require.

Contexts :

SRL stores all schemata in what is called "contexts". "Each context is like a separate database in which schemata are stored". The "context mechanism" provided in SRL allows

the user the means to separate the various schemata into various groups and/or sub-groups according to the users specifications. One of the key reasons in setting up such a feature, was to allow two unique schemata with the same name to exist in two different contexts.

In addition, to the features described above SRL offers DB. "DB is a multi-process database system especially designed for SRL". Use of this database for storage and recall of schemata also greatly facilitates work for the user.

OPS5 :

OPS5, while perhaps not a totally rule-based system, (it does use attribute describing data-structures resembling primitive frames), is certainly prototypical of them. It is typical of the rule-based systems previously discussed and has the following noteworthy points:

- it is strictly a forward chaining system
- it employs the concept of a "working memory" in which all facts are stored so as to be readily accessible for matching with the rules
- it offers a variety of *conflict resolution* strategies depending upon the needs of the user
- it makes various logical operators available for use in conditional statements

- lisp functions can be called from OPS5

- A recent update of OPS5 called **OPS83**, which greatly facilitates procedural calls, is now available.

Despite the fact that OPS5 is one of the oldest rule-based shells still in use, it is still selected as the tool of choice for many sophisticated contemporary projects [18].

KEE :

KEE, the Knowledge Engineering Environment, was developed by Intellicorp Inc (originally Intelligenetics) in 1983 [19]. The "Knowledge Engineering Environment" (KEE) is a hybrid system. It has most of the features which characterize large frame-based shells. Its features especially worth noting are :

- there are properties associated with each slot called *facets*, which permit the user to specify the **constraints** to which a slot is subject to both in terms of the number of values a slot can have and the type of values they must be. *CardinalityMin* and *CardinalityMax* specify the minimum and maximum number of values a slot is permitted to have. *ValueClass* specifies the class which a slot value must belong to

- there are two methods available to implement *procedural attachment* (usually LISP); "methods" and "active values", each being appropriate for somewhat different needs

- frames are used to represent procedural rules themselves

ART :

ART (Automated Reasoning Tool), is a system produced by Inference Corporation. It is neither a rule nor frame based system but a combination of both. The key features are as follows :

- the data structures are schemata, "facts" and "rules"

- aside from standard rules, there are special hypothetical", **"constraint"** and "belief" rules (see below)

- it provides mechanisms for implementing both forward and backward chaining

- it provides the "viewpoint mechanism" which allows the user to consider various alternative models

- it provides the feature of "inheritance" (as in SRL)

Facts :

In ART, "facts" are used to represent general "background knowledge". Each "fact" consists of two parts:

Proposition - the specific piece of knowledge itself such as "the car is red"

Extent - this is all and any specific circumstances related to the proposition. A typical extent of the above mentioned proposition might be; "at this specific time".

Schema:

In ART, schema are frames in which the slots are filled by facts, and the "schema type" represent **a class of objects** that share certain characteristics which makes ART very conducive to hierarchies and taxonomies.

Viewpoint "Mechanism":

The "Viewpoint Mechanism" involves the method in which ART represents the particular "view of the world" in a given state. It allows the user to trace a changing scenario. As mentioned previously, this feature allows the user to propose various "hypothetical models. Basically, it involves conjecture, whereby the outcomes result from two or more

possible paths from which one has to be chosen. ART will determine, as much as its knowledge allows, the outcomes of the various routes, noting which paths lead to dead ends. This is achieved by producing a treelike-structure of connected "viewpoints" each one representing the situation after a hypothetical step. Such a strategy allows for inheritance, when required.

Rule Base:

The rules representing the "procedural information" (ie. the expert decision-making strategy) are kept in a collection known as a rule base. Each rule takes the basic (if/then) form described previously. As mentioned before, in additon to standard rules of expertise there are two other kinds of rules possible in ART. They are; "hypothetical rules" and "belief rules".

Hypothetical Rule:

A hypothetical rule is similar in syntax to the previously described If/Then rule, the difference being that its action is one of hypothesizing something. Such a rule can be very useful in checking for danger by associating

conditions of malfunctions with all possible interpretations and checking for leads to the possibility of serious danger.

Constraint Rule:

A "Constraint Rule" is used in conjunction with a hypothesis to allow the user to specify non-permissible situations. It matches unacceptable situations to actions specifying abandoning a hypothesis (this is called "poisoning a plan").

Implementing Constraints in Art:

As previously mentioned, in addition to the standard features of hybrid knowledge-based systems, ART has a specific feature which when combined with some of the aforementioned features is well suited to implementing constraints. This feature is the ability to create what are called "Constraint Rules".

Constraint rules can best be understood in relation to the strategy of viewpoints which were previously described. These "viewpoints" represent possible states that would

result from a certain series of events. Constraint rules act as a check on these viewpoints so that if, and when, a viewpoint (ie. state) is reached in which some constraint is violated, then the entire hypothetical path leading up to that situation is discarded and the next alternative path (if any remain) can be considered. In the terminology of ART, the viewpoint is said to be "poisoned".

Creating Constraint Rules:

For the sake of an example, we consider a constraint which says (in english) that the total cost of the system must not exceed $20,000.00 .

Syntactically the rule would look like the following

If (cost of system > $20,000.00)
then (POISON the plan))

This is shown below as a somewhat more formally defined ART rule involving the **poison** command.

```
( Defrule Cost-limit "Keep cost under $20,000"
          (system-cost > $20,000)

          ---\
          ---/

          (Poison "cost > $20,000) )
```

In this rule, the part in double quotes is merely a descriptive comment. Cost-limit is the name of the rule and Defrule is a standard art function to define a rule. However, ART makes available a special purpose function called **DefContradiction** for defining constraint rules. It is used below to more easily create our example constraint rule.

```
(DefContradiction Cost-limit "Keep cost under $20,000"
                  (system-cost > $20,000))
```

Now, assuming that the cost of the system conjectured is stored in some schema named "system" in a slot named "cost" the rule would then become:

```
(DefContradiction Cost-limit "Keep cost of system < $20,000")
        (Schema System (Cost $20,000)).
```

An example of a more complex and possibly more relevant rule is a case in which the loads on a beam must not be allowed to exceed a certain limit. Such a rule might look something like this

```
(DefContradiction Total-Beam-Load "Keep load Below legal
                                   limit")
                  (Schema Load A  < 1000)
                  (Schema Load B < 70))
```

Belief Rule:

Finally, ART allows rules that are associated with certain hypothetical viewpoints. They specify a condition which, when fulfilled, confirm that the viewpoint should be accepted. This is a cruicial feature of ART because *when a viewpoint is believed the entire structure of viewpoints becomes "restructured" to reflect the new reality and all incompatible viewpoints are discarded.*

Additional Features:

In addition to all the features described above, ART also offers interface package to facilitate the user's work. The package is called STUDIO and, to briefly summarize, it aids in the creation, maintenance and deletion of schemata, rules, facts, goals and viewpoints as well as debugging and monitoring of test systems. In addition a package called ARTIST allows the user to create his own customized interfacing capabilities if required.

2.3 Methodology of Expert System Development :

The development of an Expert System, even a prototype, is normally a complex task which involves a team, with one member being the so-called ***domain expert***, a specialist in the particular field of interest.

The second member of the team is the ***knowledge engineer***, a specialist in the use of knowledge based systems and the tools used to build them [14].

Conventional Approach :

The increasing number of commercially available shells have, in some cases, tended to give the impression that the development of an expert system is now a relatively routine and straightforward process. However, the domain experts usually find the procedure much more difficult and time-consuming than expected.

The conventional approach to the development of a knowledge-based system centers around the ***transfer*** *of knowledge and expertise* from the domain expert to the knowledge engineer. It is this transfer which represents the ***major bottleneck*** in the building of expert systems [6].

" *The most expensive task in the development of an expert system is generally that of formalizing the expert knowledge. Typically, an expert system may take several man-*

years to develop. The simplest method of teaching an expert system what the expert knows is to "literally spell it out", in the form of production rules ... " [27].

At the initialization of a project the two team members will engage in a set of preliminary meetings in order for the knowledge engineer to obtain a grasp of the basic concepts involved. These are followed by an extended period of time (usually several months or years) in which detailed information (ie. knowledge) is explained to the knowledge engineer by the domain expert. Once the knowledge engineer has obtained enough know how, he may begin to transfer the knowledge into the form of knowledge representation structures. Eventually enough knowledge will be transferred to form a basic framework for the expert system.

Such a framework is generally only a first step towards the development of a truly realistic and useful knowledge base. In the case of large systems, several more months or even years of such transfer of expertise may be required followed by continual updating of the knowledge base.

Alternate Approach :

An alternate approach to the standard described above involves an emphasis on training the *domain expert* in the use of the required tools and, once a thorough understanding has been obtained, to allow this same expert to perform the knowledge engineering himself.

Typical training programs consist of two training sessions, each a week long. The first one is a more general introductory course and the second much more focused and specific. These training sessions are presented as a crash course in the basics of knowledge engineering principles sufficient to make preliminary design decisions towards the development of expert systems.

Following this, trainees are sent back to their companies with the reassurance of hotline technical support and service so as to insure the proper design and development of the expert system.

It is in the implementation of this policy that the high-degree of complexity and sophistication of the latest shells becomes an achilles heel.

Although this approach seems workable, in reality, the time required for a domain expert to obtain a true understanding of the concepts and methods of implementation involved in knowledge engineering, is much greater than two

weeks. In other words, the training of a novice presents a ***bottleneck just as formidable*** as does the transfer of expertise, and it is these bottlenecks which have **presented a widespread industrial acceptance** of expert system strategy up to this point.

An additional problem involves the recent observations, by many users, that due to the dynamic nature of *real-life knowledge*, an expert system can be useful only if it allows for easy and convenient ongoing modification by the domain expert[32,35].

Since, over an extended period of time, it is likely that more than one domain expert will be involved with the maintenance and upgrading of a knowledge base, the constraint described above translates into the requirement that a novice be able to make modifications to a database with little or no training.

Attempts to address this problem have led to various attempts to design "automated knowledge acquisition" systems such as; TEIRESIAS, MORE[33] and MOCE[34]. At the time of this working, however, all such attempts have lacked any substantial degree of real knowledge engineering capability (see section 3.2).

NEW APPROACH :

One of the key apects of the Escher project is the use of a novel strategy of implementation of the transfer of

expertise. The solution to the bottleneck of knowledge acquisition was found in the form of a hybrid approach. On the one hand, the transfer of expertise involved was extremely minimal because most of the domain knowledge was entered directly by the domain expert. On the other hand, the training effort was also kept to a minimum by the use of a front-end system which guided the domain expert every step of the way, performing on his behalf all of the required implementational checks and operations. This is of extreme significance when dealing with an initial prototype that is merely representative of a much larger amount of knowledge to follow. With this approach, more knowledge can be added and existing knowledge modified by the domain expert himself as he sees fit.

This front-end system is called ESCHER (Expertise, Structure, Class and Hierarchy Engineering Resource).

CHAPTER III - INTERFACES

3.1 Interfaces:

It is extremely important, at the start of a review of interfaces, to outline some distinctions between the two concepts of interfaces related to shells.

Firstly, there is the distinction between the *developer-interface* and *user-interface*. Not only are these two to be used by different individuals, but they are meant to be used in a different manner and for a completely different purpose.

Development interfaces act as an aid in the building up of and the maintaining of a knowledge-base. That is, to say, the adding of new knowledge and deletion of erroneous or obsolete knowledge. In addition, it may also support debugging facilities.

An end-user interface, on the other hand, is designed as a mechanism for maintenance of the data base of facts related to the individual case being studied and serves to allow no direct access to the knowledge base. It is simply a means of displaying pertinent questions, receiving the user-supplied responses and perhaps displaying explanations of the reasoning behind the questions that it has posed.

As, the functionality of Escher is focused on the manipulation of the knowledge base, it is the development interface that shall be the focus of this section.

Secondly, one must distinguish between the two goals, either one or both of which may be associated with the developer-interface.

The first goal, the more feasible and implementable of the two, is that of making available, on the level of primitive structures, any information desired and allowing manipulation of these same primitive structures.

The second, more ambitious goal, which may be associated with the interface, involves the higher level *knowledge acquisition* process. The key difference, associated with this process, is that it involves some key decisions being made by the system and ***transparent to the user*** in regards to the translation of actual knowledge into a collection of knowledge representation structures.

It is important to clarify which goals are addressed in the case of each interface since the approaches to the design of it will be greatly determined by this.

In the case of Escher, NO automated knowledge acquisition was intended, and it was the goal of primitive structure management , pure and simple, which was addressed.

Interfaces, at least those in relation to knowledge engineering, tend to fall into two general classes; Graphical and explanatory.

The concept of interfaces to frame-based shells stems initially from the desire and ,later on, the industrial

necessity to enhance these complex tools with a greater degree of user-friendliness.

The need for such an interface was recognized from the start. Following the development of MYCIN it was discovered that a user-friendly interface to the knowledge-base was required to allow the constant maintenance and upgrading of it. It was on this basis that one of the first interfaces, a system called TEIRESIAS was built [1], [6] (which is discussed below). Its function revolved around providing an english type of translation of the rules it was actually using. Such a translation was referred to as an *explanation*.

The more recent high-powered knowledge-base development tools all **boast** of the user-friendliness of their knowledge base interfaces, in some cases, going so far as to claim full natural language capability. Of course, as might be expected, the products deliver far less and problems ensue from that point on. The user-friendliness, it often turns out, usually relates more to the sophisticated screen graphics and text editing facilities rather than any real knowledge engineering automation. In other words, the interface improvements have been in the manner of inputting the text rather than in the knowledge engineering itself.

Other systems, albeit smaller ones, boast a facility for inputting expertise which is independent of any syntax [7]. Such systems (such as TIMM , RULEMASTER , etc ..) have inevitably involved the replacement of such syntax with some

variation of table building facilities which, in the end, again, amount to text input assistance but *leave the user to his own means* insofar as the ***knowledge engineering*** itself is concerned.

Other, more recent, attempts revolve around the building of rule-based systems that will be capable of self-modification (ie. a system that will add rules to its own knowledge base) [15].

A substantial amount of work has also been done in the area of interface but with more limited degree of success.

Early attempts at user interfaces, for the most part, revolved around some sort of english-type of version of the rules present in the knowledge base, TEIRESIAS being a typical example [1], [6].

Other, smaller systems boast a facility for inputting expertise which is independent of any syntax. Such systems have inevitably involved the replacement of such syntax with some variation of table building facilities which, in the end, again amount to text input aid but leave the user to his own means insofar as the knowledge engineering itself is concerned.

3.2 TEIRESIAS :

Since Teiresias, the functionality of which, is shown below in Fig. 3.1, was developed as an interface to EMYCIN,

it dealt with only one kind of knowledge structure, namely rules an example of which is shown below in Fig.3.2.

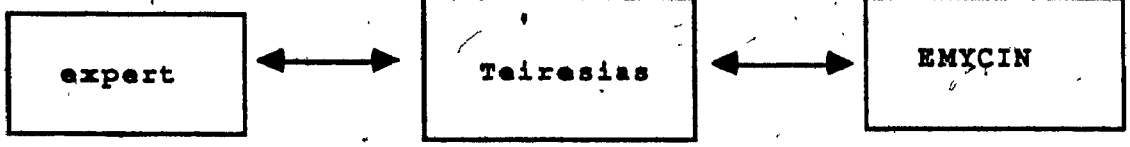


Fig.3.1 Schematic of Teiresias Functionality

RULE 050

If

1) the infection is primary-bacteremia, and
2) the site of the culture is one of the sterile sites, and
3) the suspected portal of entry of the organism is the gastro-intestinal tract,

Then there is suggestive evidence (.7) that the identity of the organism is bacteroides.

```
PREMISE    ($AND (SAME CNTXT INFECT PRIMARY-
                  BACTERMIA)
              (MEMBF CNTXT SITE STERILESITES)
              (CONCLUDE CNTXT IDENT BACTEROIDES TALL.7)
```

Fig.3.2 A MYCIN rule.

The goal of Teiresias, as previously mentioned, was to facilitate the interaction between the domain expert and the Emycin. Such an interaction involved its use as both an end-user and development interface, and was intended as a means with which to automate the knowledge acquisition and knowledge engineering process, that is to say, to automate an "interactive transfer of expertise". It is important to note that TEIRESIAS was designed to accept new knowledge from a developer but not to derive new knowledge on its own.

What Teiresias ended up as was ,at least, in so far as the development interface was concerned, a sophisticated rule editor along with an associated debugger(at least by today's standards).

It allowed the domain expert to state rules in a pseudo-english form. Such rules were then checked by the system to screen out certain conflicts on inconsistencies.

New knowledge was added in the form of rules. Test runs, with traces, allowed a developer to search out at what points the expert system was making erroneous decisions, and allowed for addition, deletion or modifications of those rules, as required.

Certainly, there was a user-transparent back-end part to Teiresias which was responsible for "modeling" the expertise and updating it as the need arose and, no doubt, this was a great aid in helping to structure the rules. However, the

influence of this to the overall user-friendliness of the system could be argued both for and against.

In the final analysis, it was still left largely up to the developer to break the expertise up into chunks and form them into rules, and it is this process which, to this day, remains largely a manual and human one.

In addition, such a strategy greatly facilitates the development of customized data types.

Some key conclusions that were made part of the Teiresias group are very noteworthy and have a bearing on the functionality of Escher:

1 - In order to facilitate the automation of knowledge engineering acquisition it is of great use to supply the system with some concepts, even primitive ones, in regards to the knowledge representation structures which are being used.

2 - The use of a higher-level conceptual representation of the various structures used, wherein each structure is an extension of the uniform conceptual structure allows for a clearer and more compact and maintainable control mechanism. (Note that such an approach is used, at least to some degree, by most of the large frame-based systems, and in the case of knowledge-craft, is used to a high degree).

In addition, such a strategy greatly facilitates the development of customized data types.

Teiresias was eventually incorporated and updated to become part of the package known as S/I. (marketed by the Teknowledge Corporation), and while the package as a whole is touted as one of the most user friendly of the large frame-based shells. Yet, even after more that 25 years of improvements, developers without an appropriate background and training require careful guiding and training every step of the way, in the development of industrial scale knowledge bases.

3.3 Knowledge Acquisition in ART :

In evaluating one of the more contempary shells, the Automatic Reasoning Tool, we find the basic strategy and structure surprisingly similar to that of the previously discussed Teiresias. The structure is shown below in Fig 3.3 Once again, the same 3 components come up.

1) Facility for editing the knowledge representation structures
2) Debugging facilities
3) End-user interface development tool

As before, the editing and debugging facilities constitute the mechanisms for knowledge transfer. In this case the structures involved are more varied (rules, frames, etc...) and the graphics mechanisms much more sophisticated, incorporating menus and mouse capabilities. However, it is still the responsibility of the developer to deal with the knowledge directly in a premature structure level and deal with the management of the knowledge base at this level.

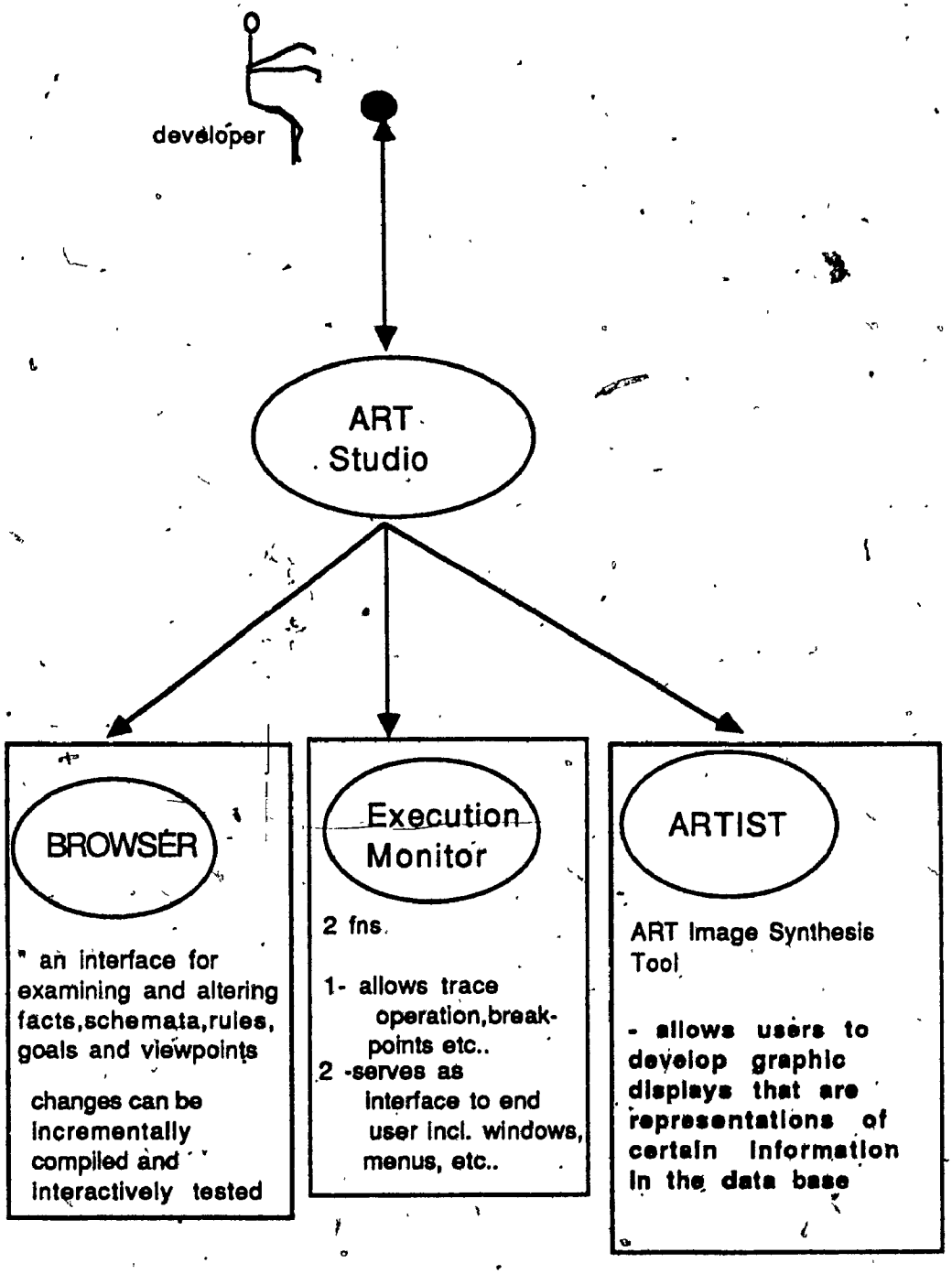


Fig. 3.3 Schematic of Art Studio Interface

3.4 ESCHER :

As a result of the analysis of past and present knowledge acquisition aids, the focus of Escher as an interface was strictly to facilitate the management of the knowledge base by the developer, on the level of primitive structures, essentially, frames, slots and values. How this was achieved is outlined in detail, in the following section.

CHAPTER IV, - Escher

4.1 ESCHER:

The functionality behind ESCHER (**E**xpertise **S**tructure **C**lass and **H**ierarchy **E**ngineering **R**esource) is that it has three responsabilities:

1 - to display available actions in a menu driven form, thereby, avoiding the syntactical restraints which normally accompany the initial use of a large system (such as Knowledge-craft)

2 - to perform various types of procedural and domain constraint checks on all specified **actions**

3 - to perform bookkeeping and house cleaning functions transparent to the user (ie. such things as preparing and setting up files for display or printing, transferring all garbage frames to a junk file,etc..)

The concept of ESCHER (**E**xpertise **S**tructure **C**lass and **H**ierarchy **E**ngineering **R**esource) was to make available to the domain expert some means of directly inputting, to the knowledge base, some of his/her expertise. Furthermore, this facility must operate on the assumption that the user has little or no experience or specialized field of knowledge engineering.

What is, therefore, required is some system in which the mechanism and factors involved in the building of various

types of knowledge structures (including the building of various types of connections between them) is implicitly encoded. What this effectively translates to is that Escher, itself, is a type of expert system in the building of knowledge bases. This removes from the domain expert much of the overhead in the transfer of expertise to the computer.

4.2 Development Environment :

For the purposes of designing, building and testing ESCHER, the schema-based system of Knowledge-Craft running on a Vax-8650 Super mini on V.M.S. with 44 Megs of R.A.M. was used.

Knowledge-craft is a knowledge engineering environment which offers two structure types to represent knowledge. These structures are, the previously discussed, *rules* and *schemata*.

4.3 Architecture of ESCHER :

Fig 4.1 below illustrates that the basic architecture of ESCHER consisting of three parts:

- the users interface
- the model builder

- the shell interface

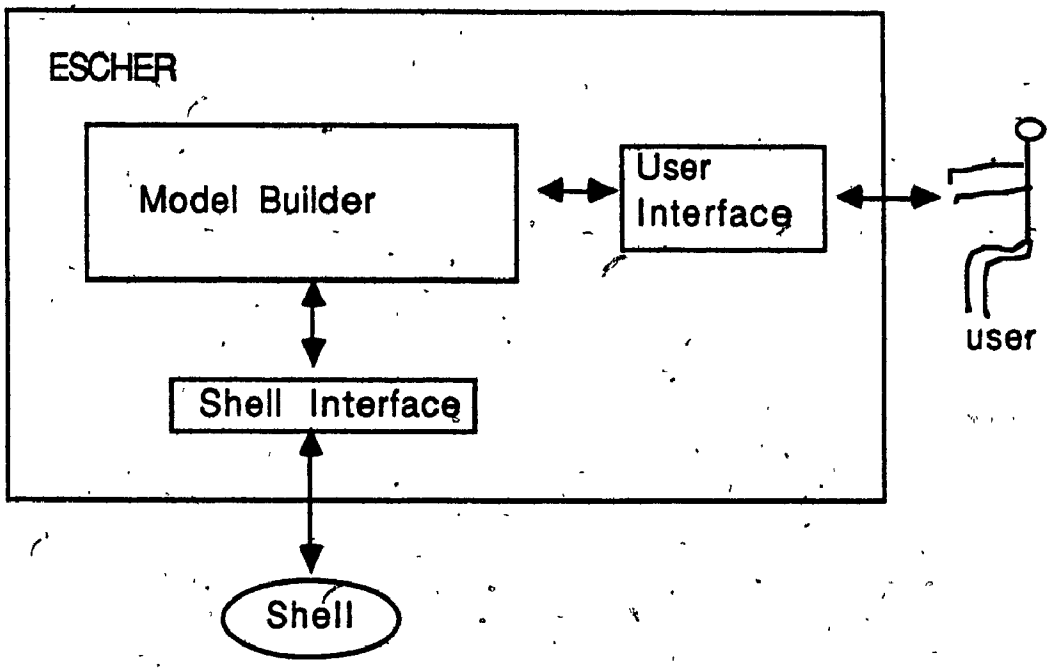


fig. 4.1 Overview of ESCHER

User Interface:

The part consists of modules which are responsable for the controlling of and communication with the user. This means handling of all parts of the screens and windows as well as the mechanisms for allowing the user to input command requests.

Shell Interface:

This part is responsable for all interaction with the shell. This interaction takes place transparent to the user and basically serves as a driver specific to the shell.

Model Builder:

This is the most complex part of Escher and contains all of the modules involved in the actual creation and maintenance of the frames in the knowledge base.

Escher's User Interface:

The user interface, (ie the interface to the user of Escher, not to be confused with the interface to the user of the final expert system discussed on page 49), has two

responsibilities. The first is controlling the means of display of information, which includes the breaking up of the screen into various parts and the control of each part.

The second is relaying the action requests from the user to the model builder. Once a proper menu selection has been made, the proper checks are implemented and if no problems are found the action is then implemented.

The architecture of the interface consists of two distinct modules.

1) Graphics - involves the control of the physical display of information, design and management of several screens, highlighlighting, menu control, etc...

2) User-Escher-Communication - the actual mechanism by which Escher allows the user to specify the action requested

Graphics :

Screen-Display - There are 4 separate screens in the Escher User Interface (ie the screen is divided up into 4 parts). These are, as shown below in fig. 4.2 .

header - this screen sits at the top of the interface and displays a description of

the information which is displayed concurrently in the information screen

info - this screen contains the display of currently relevant information (ie. information related to the latest action requested by the user)

status - this information displays the current status of the system (ie. What command it is performing or what part of the command has already been entered as well as what information the system requires next)

command - this screen displays the various menus available to allow the user to specify the the appropriate command

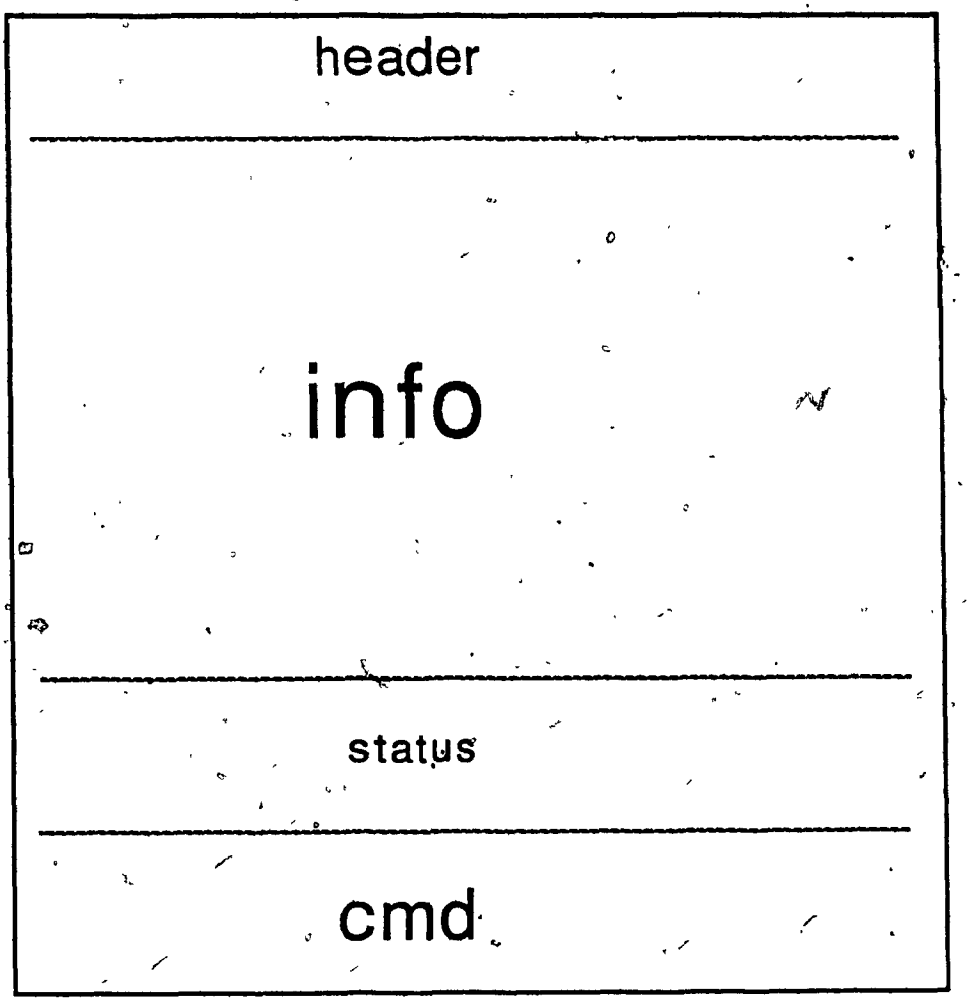


Fig. 4.2 - Structure of Screen Layout

User-Escher Communication :

The communication between the user and Escher involves a series of menus and questions presented to the user to allow the user to specify commands. The menus and questions are all presented in the cmd screen of the interface. The actual contents of the commands are based upon the syntax of commands in Escher (see section 4.3.6.5.2).

Some sample menus are shown below to display the straightforward manner in which user requests can be communicated with a minimum of confusion.

Fig.4.3, below, shows the command screen layout. Each command is associated with a key letter or letters (which must be entered to implement the command) which are highlighted on the screen in front of the appropriate command (ie. "B" for build, "DE" for destroy, etc ..). There is a space available (contained in between the square brackets on the bottom line) for default values, should any exist.

As shown, the status part of the screen will display, as shown, nothing in the command part of the screen since no part of the command has yet been issued, and show the status as "awaiting action selection", indicating it is awaiting the selection of one of the actions shown in the menu.

Once a legitimate selection has been made, the system will process the selection and make available the next screen. Should an inappropriate choice be made, the system will indicate the error with a message as shown below in Fig. 4.4 and redisplay the available choices.

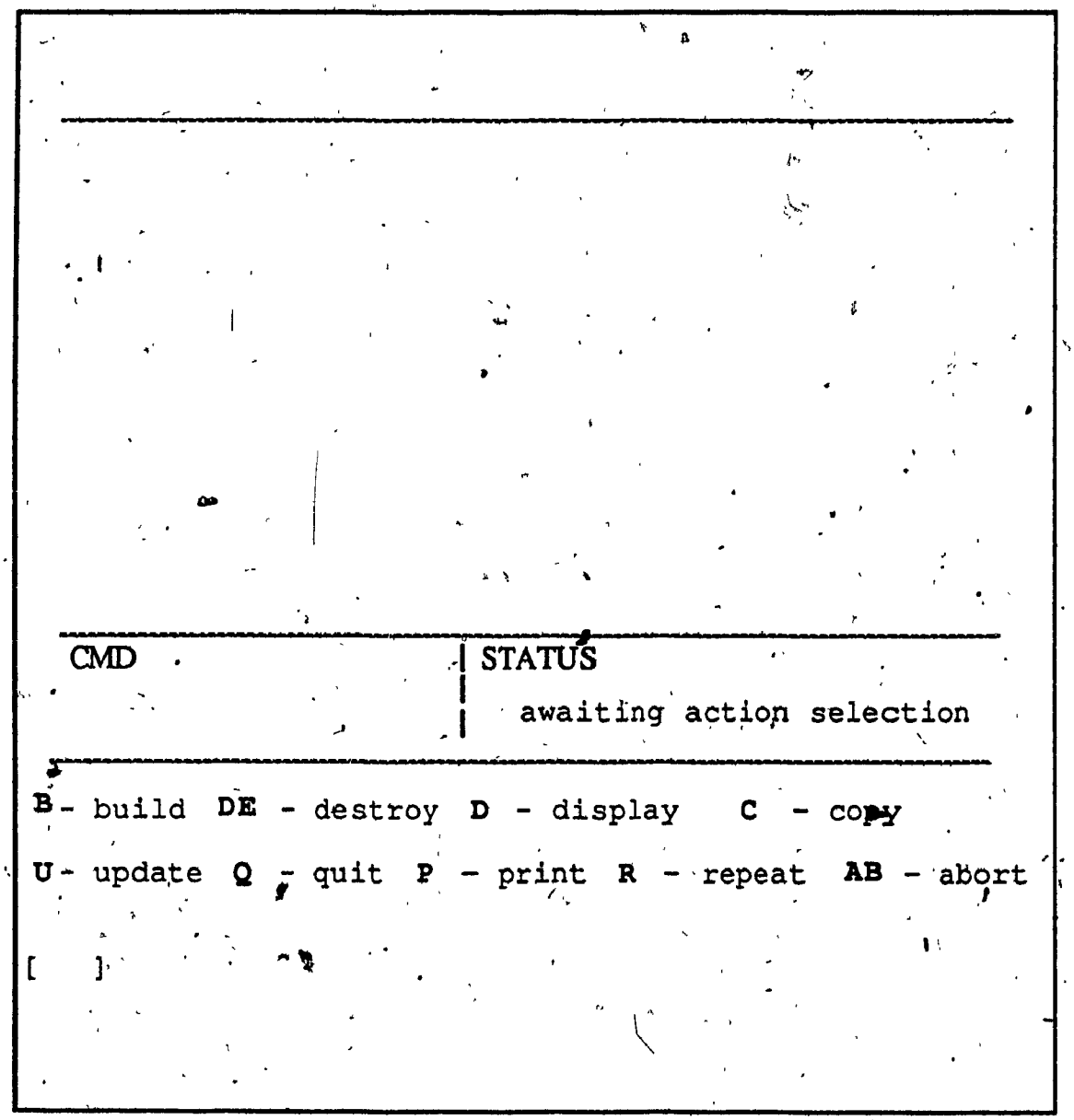


Fig. 4.3 - Command Menu Screen

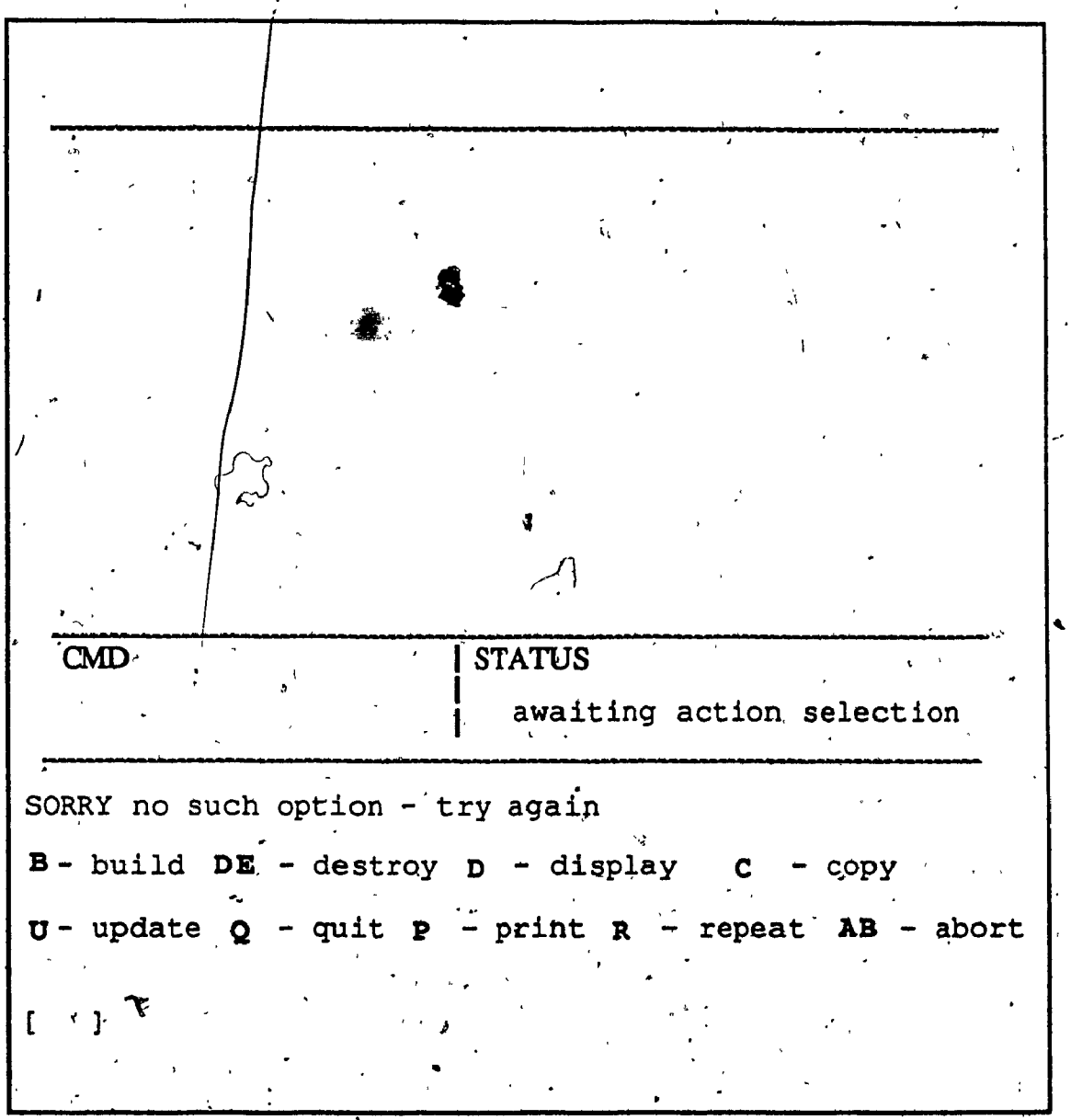


Fig. 4.4 - Command Error Screen Layout

Fig. 4.5 shows the Structure Menu Screen, in which each conceivable structure that a user might require manipulation of, is available in the menu. Again, as with the command menu. each has an associated highlighted letter which indicates the key or keys to be used in indicating the desired structure. Also, as with the command menu, a space is available to display the default.

The status, at this point, is indicated by the CMD screen showing that a " BUILD " command has been entered and shows the status as " awaiting structure selection " (ie. what type of structure is to be built).

Fig. 4.6 shows the screen format for entering a specification. In this case the name of a frame to be accessed, created, etc.. . Should a default specification exist it would appear allowing the user the convenient option of skipping over the input by means of a carriage return.

The CMD screen shows the command "BUILD the FRAME " and shows the status is " awaiting specifications" clearly indicating to the user that the system is awaiting specification of the frame to be built.

Once all of the specifications have been completed the status screen will indicate " command entry complete " and the command implementation will be attempted.

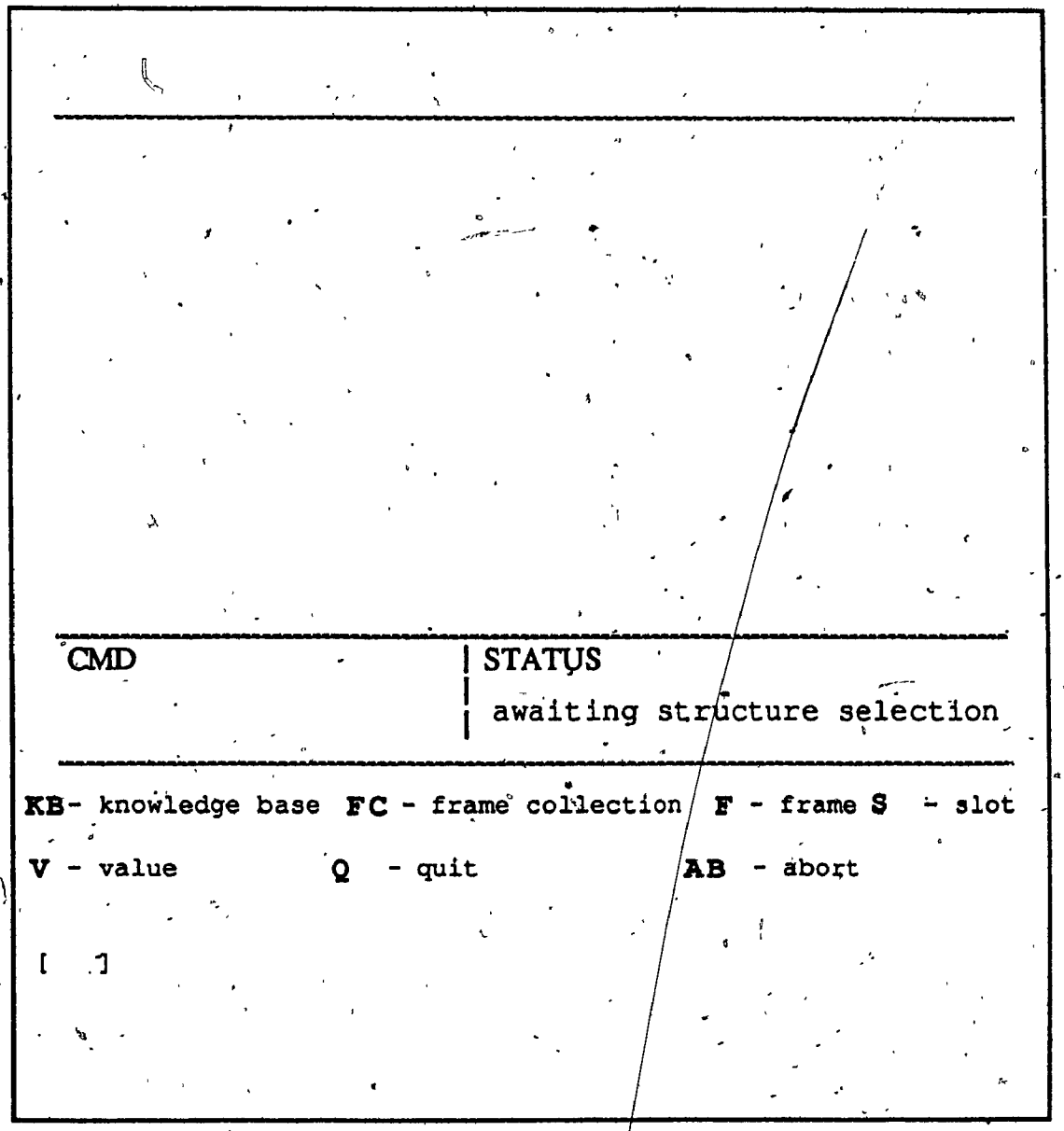


Fig. 4.5 - Structure Menu Screen

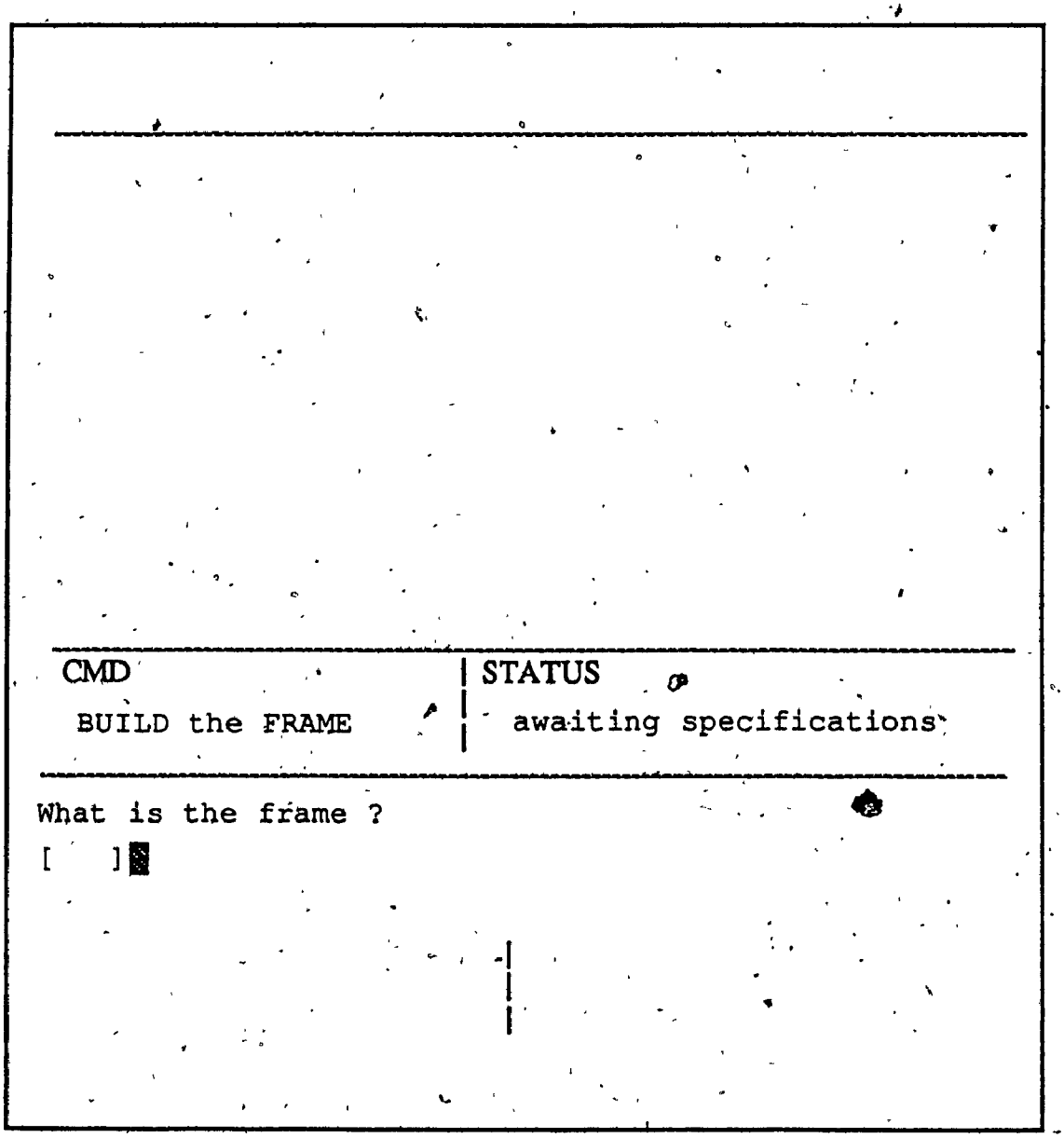


Fig. 4.6 - Request For Specification

For certain commands an option menu will appear. For example, in the case of displaying the contents of a frame collection, the option menu shown below in fig. 4.8 will be displayed, with the previously described highlighting, status and default space.

Displaying Information :

When it is necessary for some information to be displayed on the screen, as a direct or indirect result of a command, both the header and info screen are used. An example is shown in Fig. 4.7 below, wherein the frame CANADA is shown. The header screen displays the contents on the info screen, namely the frame CANADA and the slots, and their associated values are displayed on the info screen. Limits on the space available limit the information on the info screen to six lines at a time. As shown in Fig. 4.7, the user can prompt for the next six lines via a carriage return or abort the display by entering an S.

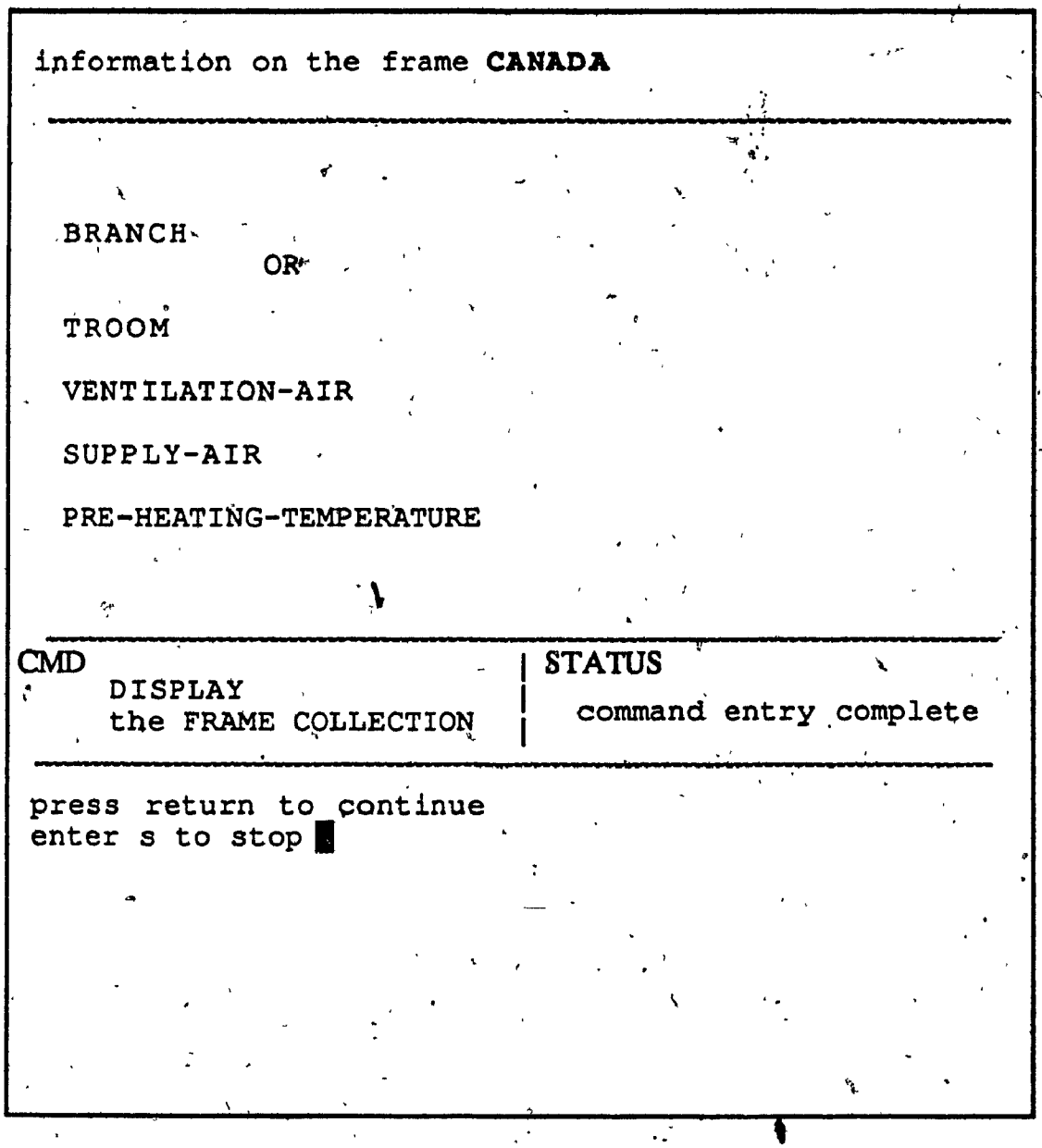


Fig. 4.7 - Sample Display Screen

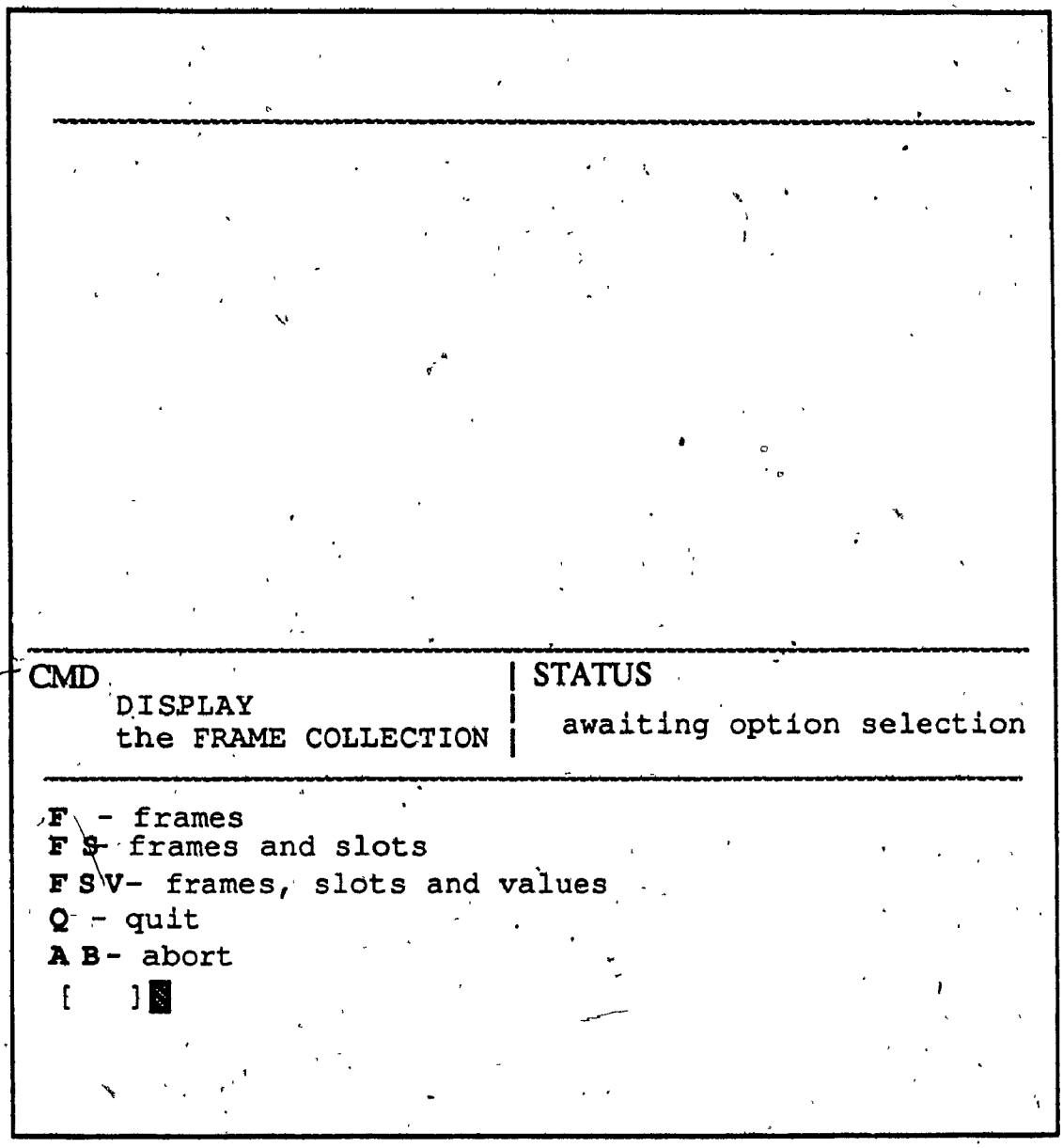


Fig. 4.8 - Sample Option Menu

Shell Interface:

This part of Escher is of course where shell specific calls are made. Generic structure commands are translated into the shell appropriate function calls.

Shell Interface Architecture:

The shell interface consists of 4 basic modules:

- frame collection
- frame
- slot
- value

Each module is responsible for operations performed on the four basic structural units. Fig. 4.9, below, outlines the architecture of shell interface.

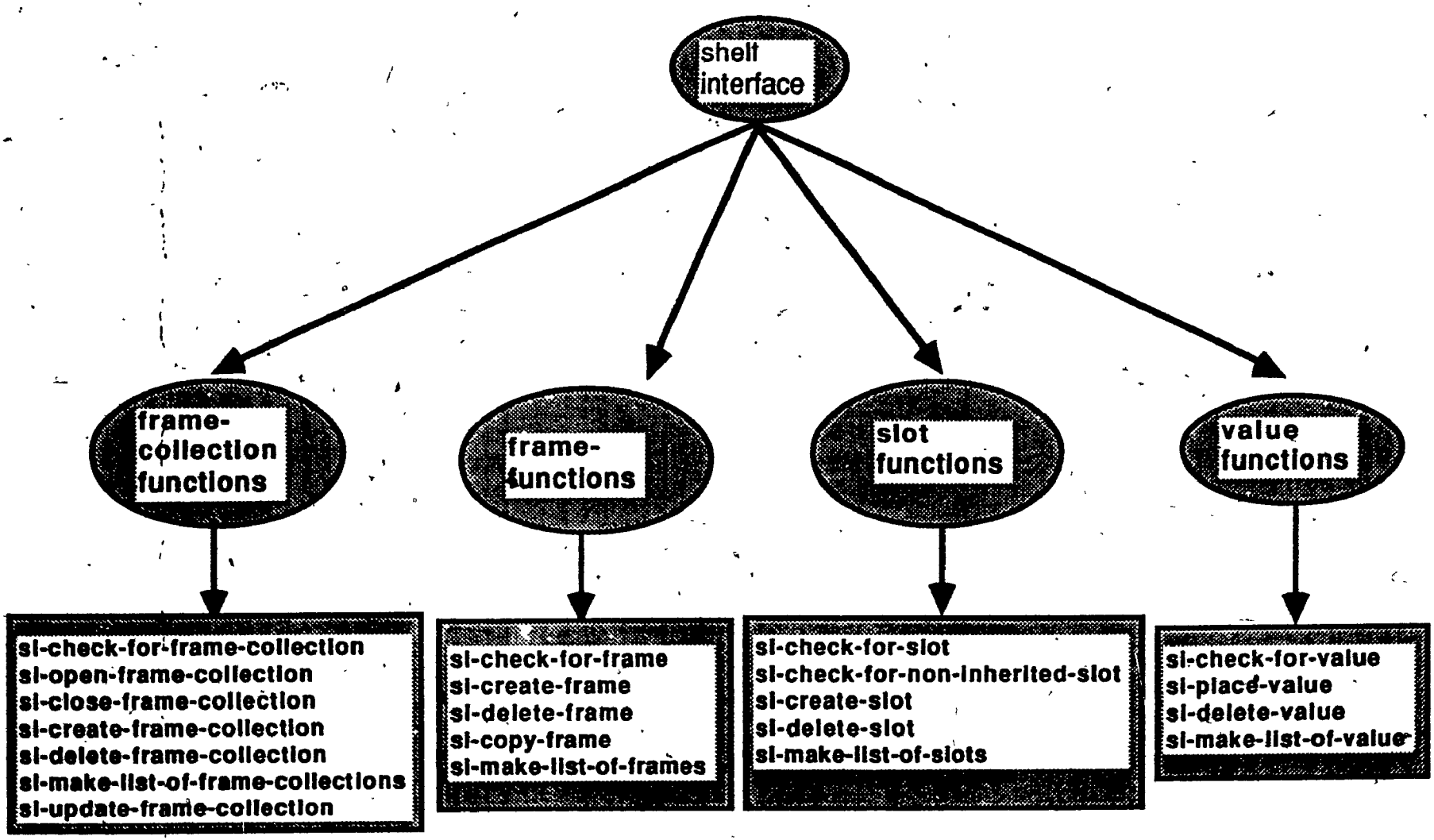


Fig. 4.9 Architecture of Shell Interface

The Model Builder:

Fig. 4.10, below, shows the basic structure of the Model Builder. It consists of 3 basic modules:

- generic interface
- library
- operating system driver

Library :

This module contains a library of routines used frequently throughout the generic interface. Most, but not all, of these routines involve some screen manipulation or more general i/o facility. The main reason for the library is to avoid the unneeded repetition of frequently used, fairly standard lisp code.

The purpose of the library is to support the routine in the generic interface.

O.S. Driver :

This module serves to support the library module, much the same as the library serves to support the generic interface. The routines in this module are used to supply

the calls specific to the operating system in use. The majority of the commands exist to support screen and file handling functions.

Generic Interface:

This module implements the control of knowledge-structure manipulation at all levels (ie. knowledge-base, frame-collection, etc...).

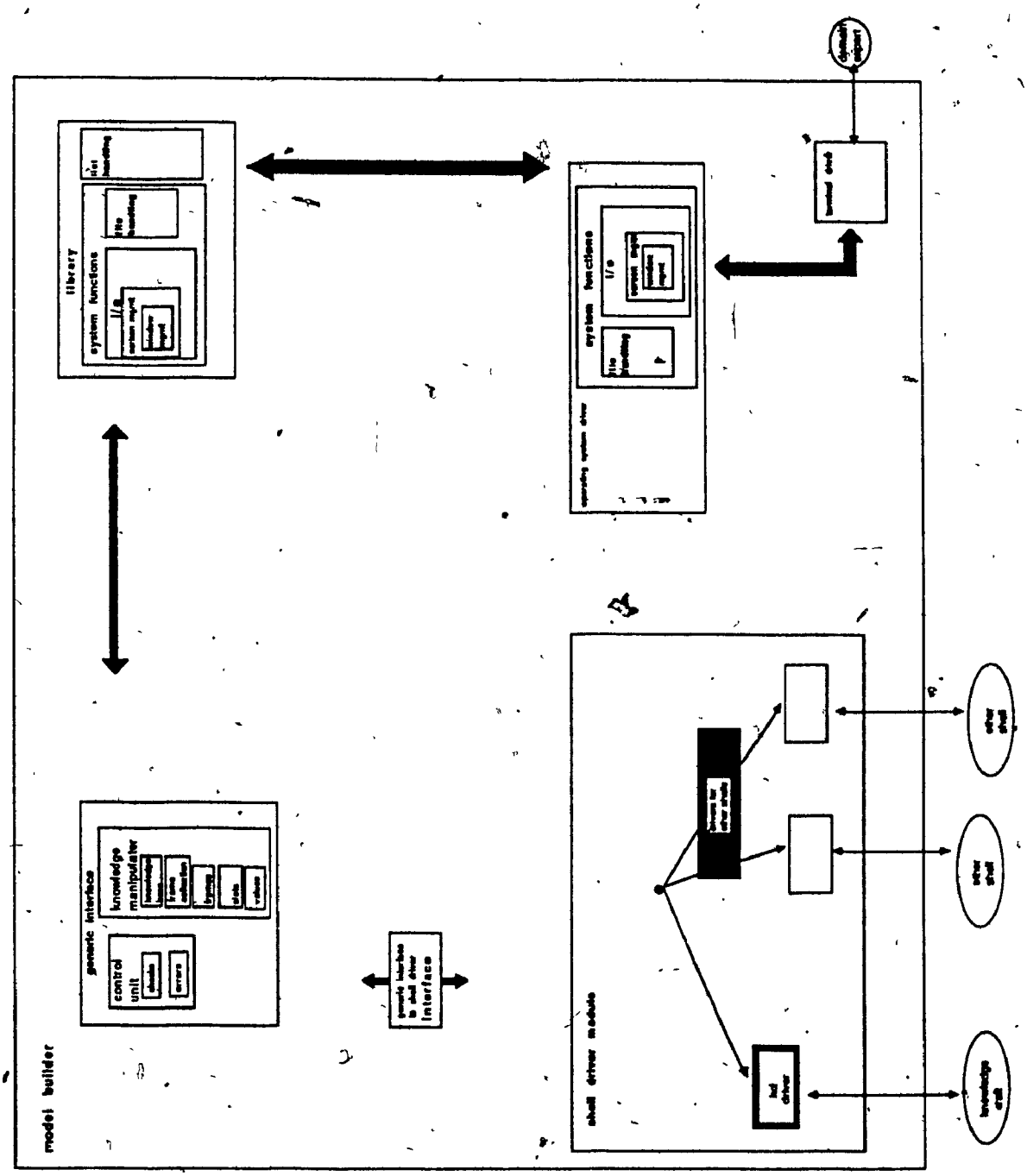


Fig. 4.10 Model Builder Architecture

Fig. 4.11 below shows a detailed schematic of the g.i. (generic interface). It basically consists of two modules. The Central Unit and the K.S.M (Knowledge Structure Manipulator). The two modules work together to implement the user-requested action.

Knowledge Structure Manipulation:

This unit contains 5 modules; one for the management (ie. creation, deletion, testing etc...) of each basic type of knowledge structure.

Control Unit:

The CU is responsible for implementing the proper action by activating the appropriate part of the K.S.M. This activity, however, is conditional within certain constraints, and ensuring that these constraints are met, is the responsibility of the CU as well.

The checks module of the CU is responsible for running all checks required to satisfy the action constraints.

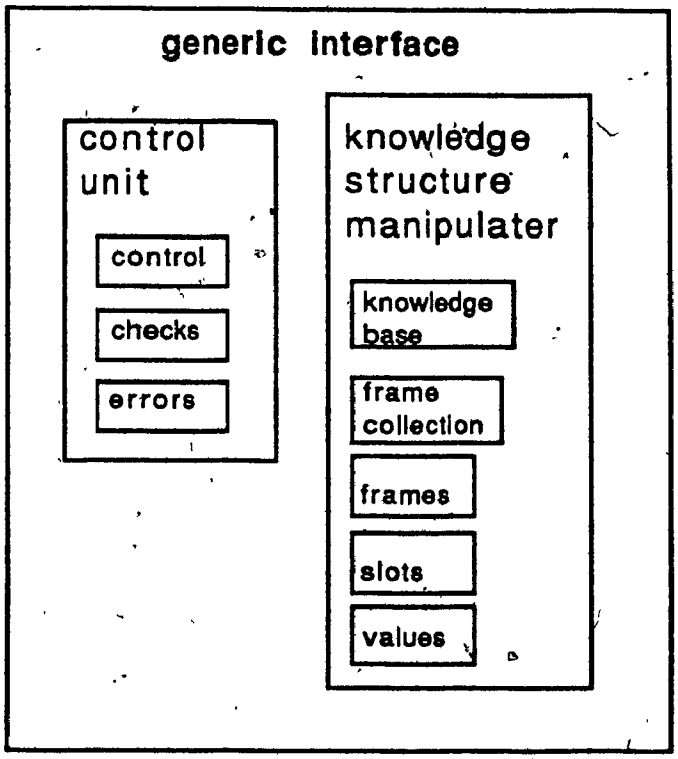


Fig. 4.11 Outline of the Generic Interface

The errors module is responsible for the proper handling of situations in which action constraints are not satisfied.

Action-Requests :

One of the key issues was the format of the *commands* in Escher. The commands were broken down into two categories of commands; *frame manipulation commands* and *procedural commands*.

frame manipulation commands :

these are commands which involve some part of the semantic network (ie. frames, slots , etc ...). In addition, these can be further broken down into three sub-classes

state-change - involving the actual changing of the state of the knowledge base

access - involving a read-only mode that will obtain information on the existing state of the knowledge base without, in any way, changing its state

fms - file management system commands that will handle all maintenance of the knowledge base which may be required

procedural commands :

these are commands whose purpose is merely to allow access to specific modes, menus, etc . . but do not access or affect, in any way, the semantic network.

For this initial prototype ***an Escher command is defined as a specified action and a specified semantic network structure associated with that action.***

Six basic frame manipulation actions were implemented in the first prototype which are:

- **build** (state-change)
- **destroy** (state-change)
- **display** (access)
- **print** (access)
- **copy** (state-change)
- **update** (fms)

In addition there are the procedural commands actions:

- **quit**
- **abort**

In addition five basic knowledge structures were identified. These are:

- knowledge base
- frame collection
- frame
- slot
- value

With these six actions and five structures there are enough commands available to build and maintain a basic but functional semantic network. Of course, not all actions and structures can be combined into logical sequences but there is, within the scope of available commands, a sufficient range for substantial semantic network manipulation.

Fig. 4.12 below shows the permissible combinations of actions and structures (ie. commands) that Escher will recognize and allow.

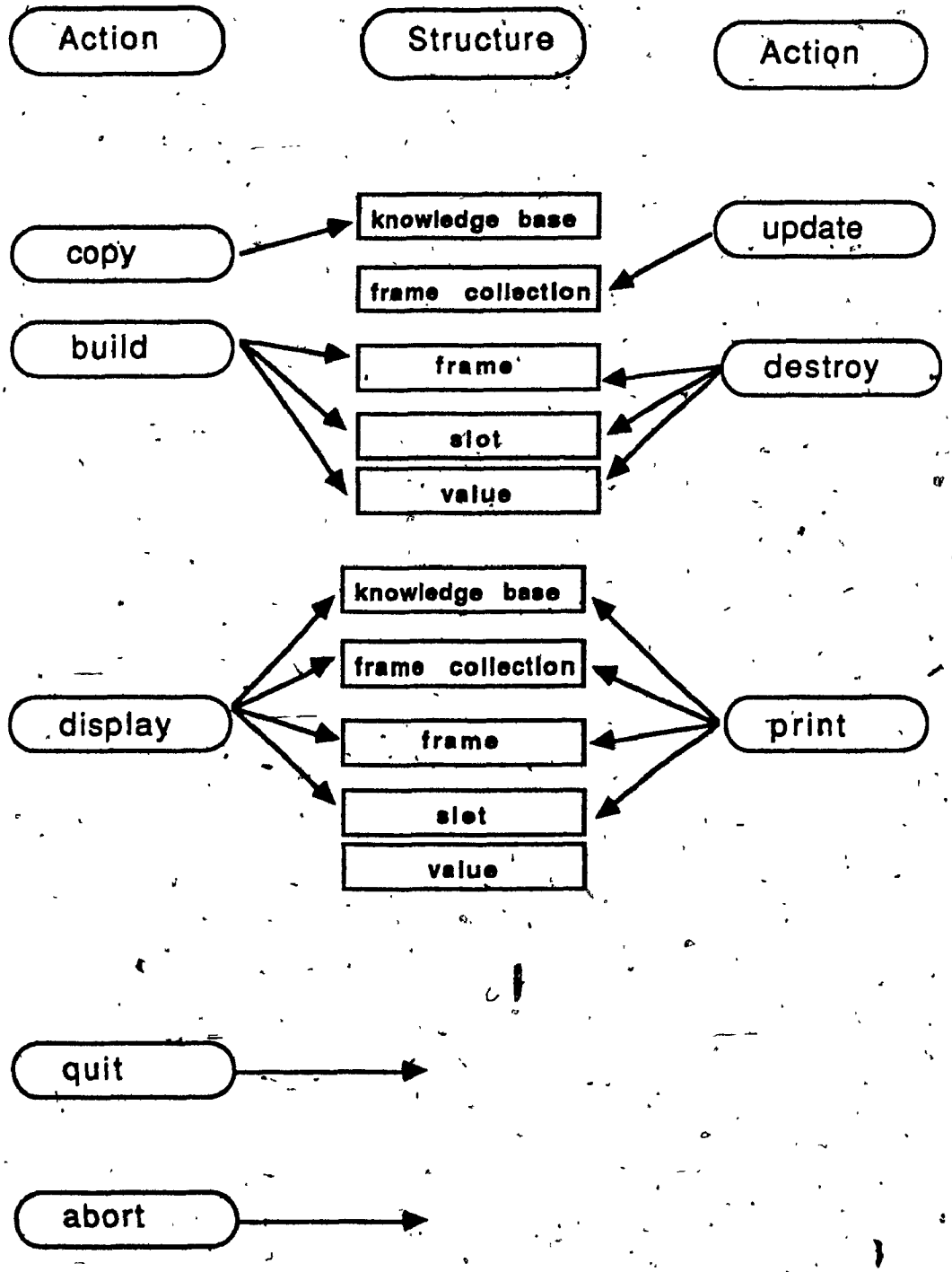


Fig. 4.12 Schematic of ESCHER Action/Structure

Each permissible command is outlined below

State Change Commands :

build frame

build slot

build value

destroy frame

destroy slot

destroy value

copy frame

Access Commands :

display knowledge-base

display frame-collection f-mode

display frame-collection fs mode

display frame-collection fsv mode

display frame

display slot

print knowledge-base

print frame-collection f-mode

print frame-collection fs mode

print frame-collection fsv mode

print frame

print slot

FMS Commands:

update database

Command Syntax:

The syntax of Escher commands are shown below in Fig. 4.13 .

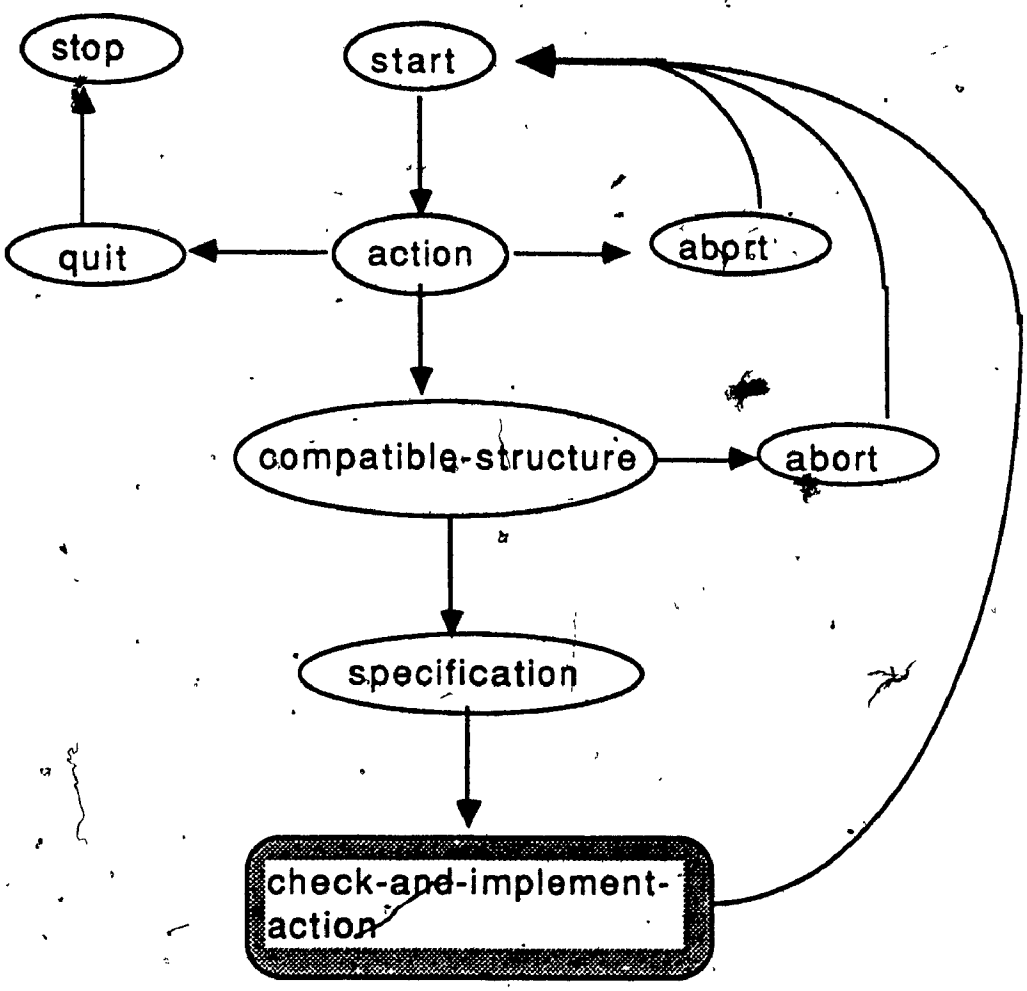


Fig. 4.13 Interface Control Flow Scheme

Action Constraints:

In order for Escher to be able to implement the correct responses, to a user specified command, it must have the ability to determine when the execution of a specified command will produce an error condition of some sort. It must, therefore, have an integrated error-handling facility.

For the sake of clarification, the term error, insofar as Escher processing is concerned, refers to *the request of a command which is not permissible under the situation at the time that the command is submitted*, for some reason. This includes both commands which are permissible, under some conditions but not others, and commands which are never permissible, for either synatctical or semantic reasons.

The architecture of Escher, in regards to the *action constraint management*, was one of the key design issues.

The necessity for an *action constraint manager* was due to two reasons.

Firstly, the handling of errors by various systems is ***incomplete***. It exists only to provide a base-line mechanism with which the user can implement more sophisticated procedures.

Secondly, it is ***inconsistent***, thus, making it, at times, not particularly helpful or even necessarily useful to the user.

Using knowledge-craft as a convenient case in point let us examine some of these short-comings.

Knowledge-craft contains three mechanisms for error-handling. Each particular kind of error is handled by ***one and only one*** mechanism. The three mechanisms are:

1 - ***error is simply ignored***

An example of this would be the command to build a frame which already exists.

Normally, the create-schema function will create-the frame specified and then return the name of the frame just created.

In the case of the error, the create-schema function of knowledge-craft performs no action, leaving the existing frame untouched and still returns the value of the name of the function to be created, just as if no error had occurred.

Therefore, from the user's point of view it is impossible to tell whether or not the command has triggered an error.

2 - ***error is handled by the function itself***

(ie via the return of value indicating a failure).

An example of this would be the deletion of an inherited slot.

In the normal execution of the delete-slot command, the name of the slot deleted is returned if the delete was successful, otherwise a NIL is returned. Thus, there is a means to determine whether or not the slot was actually deleted.

In the case of attempting to delete an inherited slot, however, a NIL will be returned indicating an unsuccessful deletion attempt, but no indication will be available as to the reason for the failure. Furthermore, if a slot is locally associated with a frame, for some reason (local value, initial assignment, etc..) an attempt to delete it will produce a value indicating success (and the slot will have been deleted but instantly inherited) which therefore produces a false impression that the slot is no longer associated with that frame.

3 - ***error is handled by an implicit error-handling mechanism.***

The implicit error-handling mechanism in knowledge-craft employs a strategy of representing certain types of errors by means of schemata representation. An example of this type of error is a *no-slot* error. A *no-slot* error occurs when a slot specific action, such as the adding or placing of a value, is directed towards a slot which does not exist in the specified frame.

All of these errors are controlled through the system-error schema . This schema contains a slot for each type of error it can handle. For the type of error described a " no-slot " error is available.

When a no-slot error occurs, the knowledge craft error-handler (the crl-error function) creates a customized instance of the system-error schema to represent the error. All pertinent slots of the schema are filled in with appropriate values and these values are then used by the general error-handling mechanism to perform appropriate responses.

In knowledge-craft the error-handling is performed (as are many other functions) through the use of schema to represent the various types of errors which may occur. These **system-error** schemata will control the manner in which the error is processed.

Implicit vs Explicit Constraint Mechanism:

Given the incompleteness and inconsistency described above, it was evident that an action constraint handling mechanism was required as part of ESCHER to compensate for these inadequacies. The crucial issue was whether or not to incorporate *implicit* constraint mechanism available in the shells. This translated to a choice between ***preventing*** errors or ***handling*** them after they had occurred.

A typical example, again, would be in the building of a frame which already exists.

The architecture employing strictly the implicit shell error-handling mechanism would be the one outlined below in fig 4.14 .

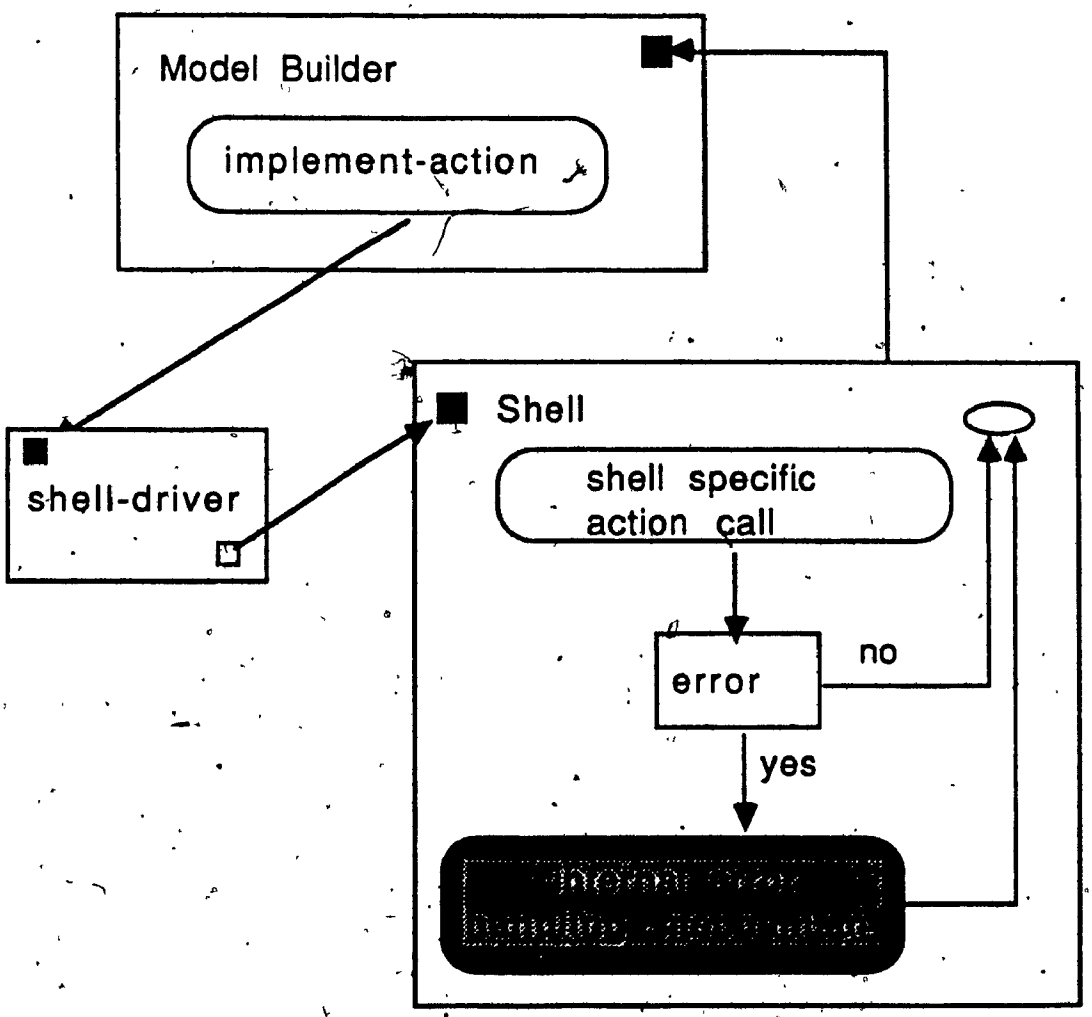


Fig. 4.14 Schematic of Implicit Action Constraint Mechanism

In such an arrangement, the command is submitted through the model builder directly to the shell and, it is the shell which performs the action-constraint processing. If the action is successful then control, along with a value indicating the success, is passed back to Escher. If the action is not successful, the internal mechanism will implement its own error handling conditions, the termination of which will be followed by the passing of control back to Escher.

The advantages of such a system are

- an already existing and tested system is used, thus, avoiding a very substantial amount of effort in designing and testing the mechanism.

The disadvantages associated with such an architecture are:

- if the case where an incompleteness or an inconsistency exists, in the internal mechanism, (both of which are present in Knowledge-Craft) then the incompleteness will be maintained at the level of the Model-Builder.

In order to compensate for that a supplementary architecture, as shown below in fig. 4.15, could be adopted. Such an architecture involves two mutually exclusive error-handling mechanisms; one implicit to the shell and the other a distinct part of the model builder. That is, each error-handler would handle a distinct group of errors. Together they would handle all types of errors and each possible type of error would be handled by one, and only one, of the mechanisms.

The advantages of this system are :

- *as above,* an already existing and tested system is used thus, avoiding a very substantial amount of effort in designing and testing the mechanism.

- the error handling is complete, as all types of errors will be accounted for by one of the two mechanisms.

The disadvantages associated with such an architecture are :

- the case of inconsistency still exists, in that two very distinct and unique mechanisms are being employed. From a software engineering point of view this is extremely undesirable. It, not only, produces inefficiencies in the code but highly increases the possibility of coding problems at a later stage.

- given, that the internal error-handlers of each shell are different in both how they handle the errors and in which errors they handle, there would have to be a customizing of the E.A.C.H, in the form of an E.A.C.H. driver module, as part of each shell driver. Such an architecture is outlined below in Fig. 4.16 . This introduces an inordinate amount of complexity and overhead associated with portability.

In addition, any changes in the error-handler in the shell, by its designers, a virtual certainty in the form of upgrades, would require changes in the model-builder's version to accomodate such changes. Therefore, the aforementioned E.A.C.H. modules in the shell drivers would have to be modified as well.

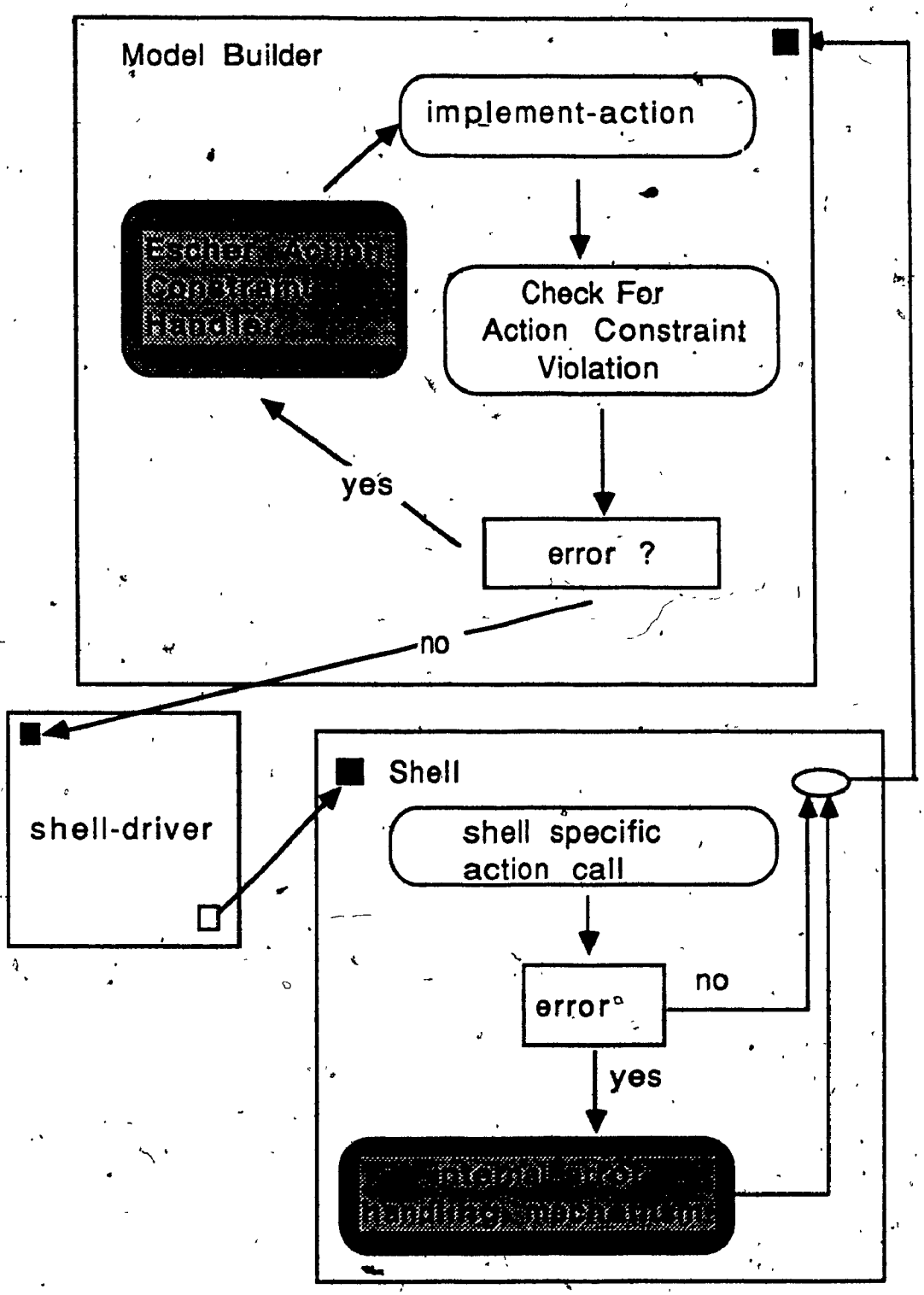


Fig. 4.15 Schematic of Supplementary Action Constraint Mechanism

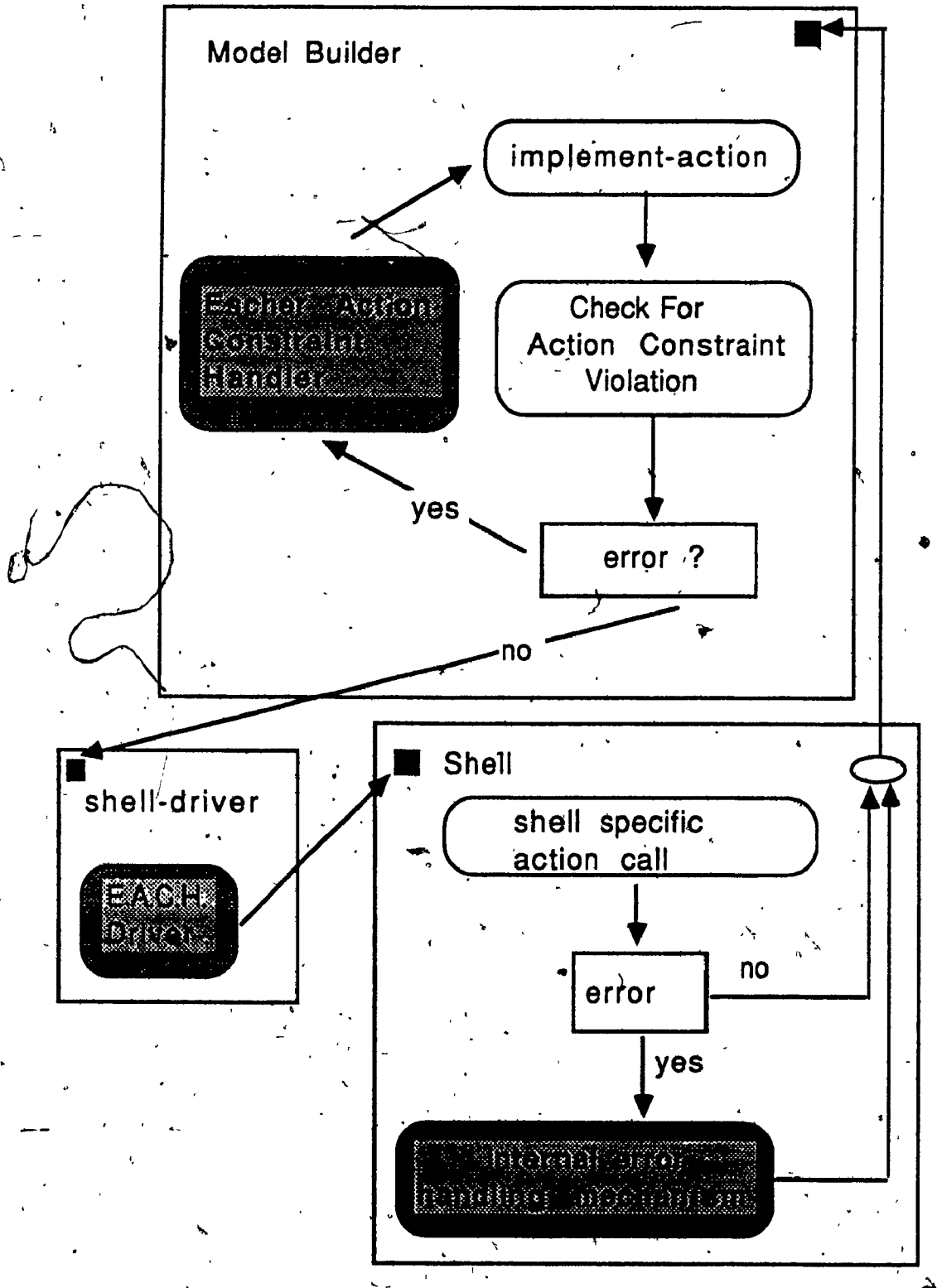


Fig. 4.16 Schematic of Customized Supplementary Action Constraint Mechanism

The final alternative involves the development of a complete and total error-handling mechansim which handles all types of errors consistently. Such an architecture is shown in fig. 4.17. This configuration provides completeness, consistency and portability at the cost of building the entire mechanism from scratch.

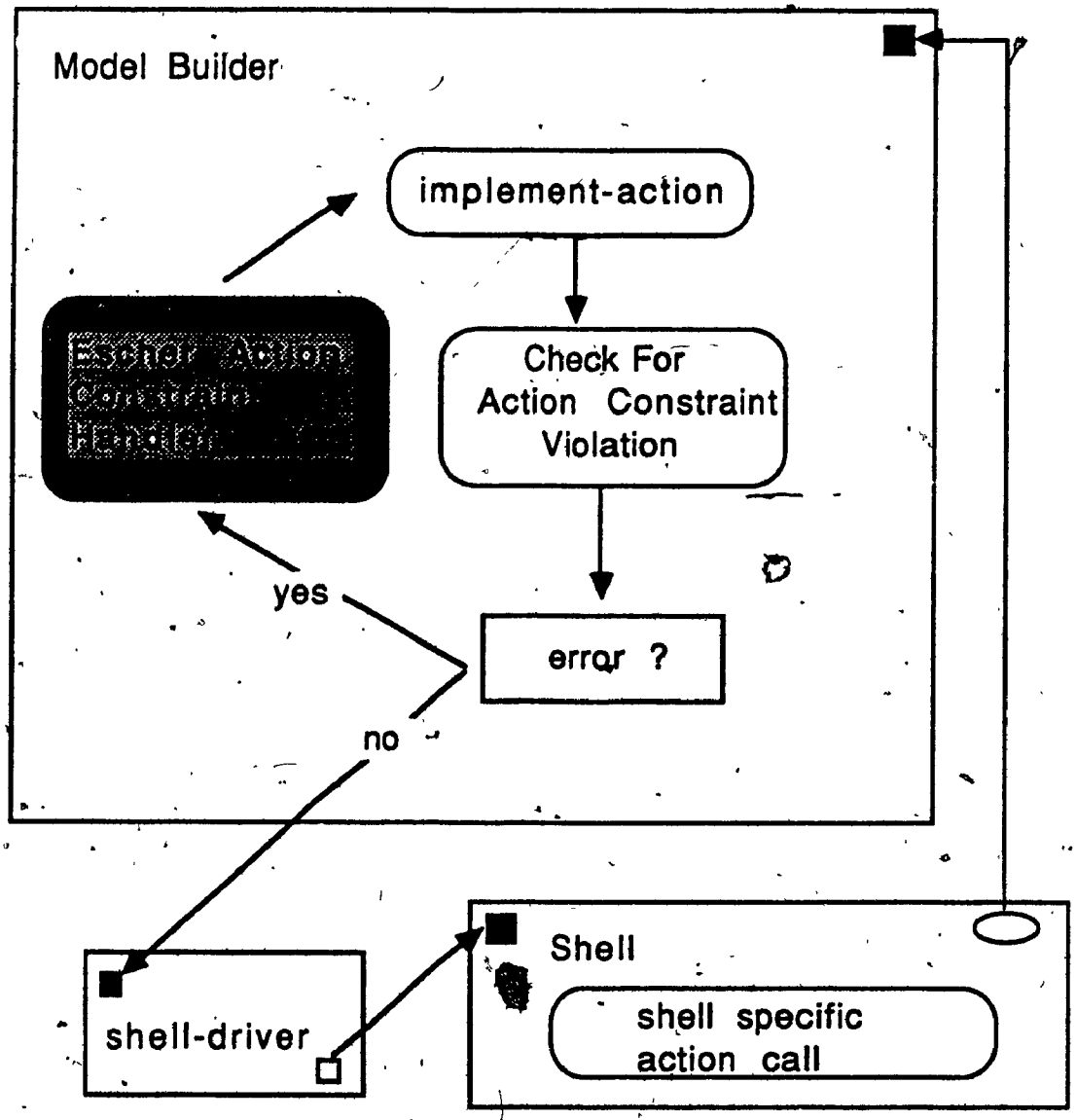


Fig. 4.17 Schematic of Portable Supplementary Action Constraint Mechanism

The selection of this final Explicit Design was made in order to take advantage of the overall modularity, flexibility and portability that the design offered. It was built in lisp as described below.

Error Handler:

Constraint-Checking:

The constraint checking module of ESCHER consists of a collection of two types of error-checking routines. These routines are the primitives and the complex. The primitives are functions which test for one specific constraint returning T if the constraint is satisfied and NIL otherwise. In addition, in the case of an error, an error-code is set up.. The primitives (as shown in fig. 4.18) are

permissible-build-check - this function checks that the structure specified is allowed to be built. Its main purpose is to keep the user from creating superfluous amounts of frame collections.

permissible-destroy-check - this function checks that the structure specified, is allowed to be destroyed. Its

main purpose is to keep the user from erroneously destroying a frame collection or the entire knowledge base.

proper-number-check - this function checks that the position specified for the placement of a value is a number (ie. not a letter, symbol, special character, etc ...).

integer-check - this function checks that the position specified for the placement of a value is an integer (ie. not a rational or irrational number).

positive-number-check - this function checks that the position specified for the placement of a value is an integer (ie. not a negative number or zero).

relation-check - this function checks that the symbol entered (a symbol being used as a slot) is a recognized relation.

existence-check - this function checks for the existence of the specified structure.

non-existence-check - this function checks for the non-existence of the specified structure (ie. that the stucture does not exist).

inheritance-check - this function checks for the locality of a slot (ie. whether or not the slot is inherited).

The complex functions are, of course, those which perform a set of constraint checks related to a specific action or area which are accomplished by means of various permutations and combinations of the primitives. The

breakdown of the complex functions used is shown below in Fig. 4.18. The 3 main groups of checking procedures are :

- legality checks
- conflict checks
- inheritance checks

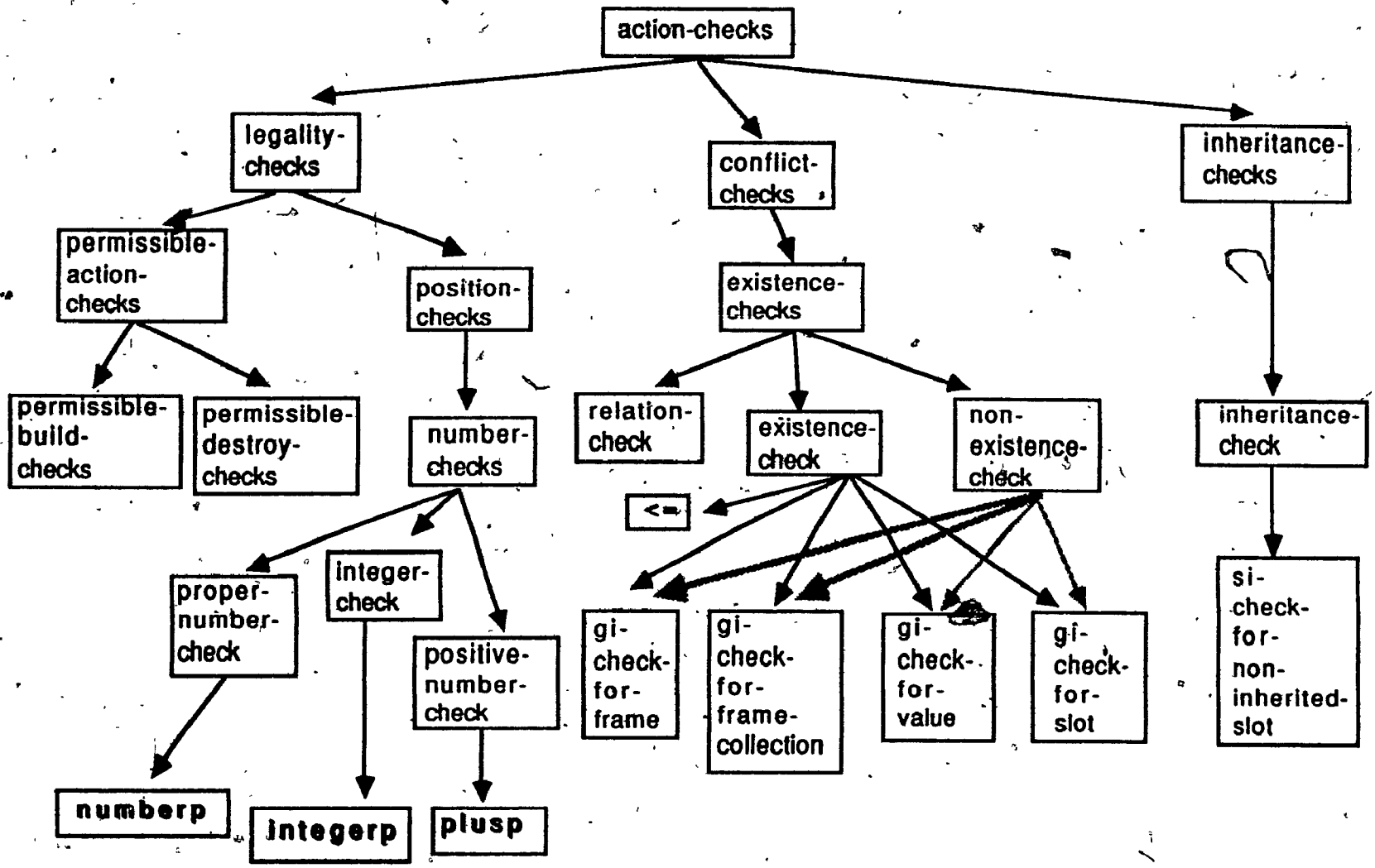


Fig. 4.18 - Hierarchy of Action Constraint Checks

Legality Checks :

These checks concern actions and parameters which are outright illegal under any circumstances. It involves checking each piece of input data to see that the input is both proper and syntactically correct. This includes:

- checking for actions not permitted on certain structures (such as destroying a frame collection

- values which are not permitted (such as minus or non-integer numbers) to specify position)

The present state of the legality checks is such that it involves only position checks. These, in turn, involve only the number checks which involve checking that the number used to specify the position of a value in a slot is a syntactically correct number (ie. it is a positive integer).

Conflict Checks :

These checks concern themselves with actions and parameters which would be permissible under the right situation, but conflict with the state as it exists at that time. In other words, this covers actions which cannot be

performed without some other action or actions being performed first.

This includes such situations as deleting a frame which does not exist, creating a slot in a frame already containing the slot, etc ... .

The present state of these checks are such that, depending upon the action requested it will perform checks to ensure the following conditions are satisfied :

- for building a frame collection :
    - the specified frame collection must **not** exist.

- for building a frame :
    - the specified frame collection must exist.
    - the specified frame must **not** exist.

- for building a slot :
    - the specified frame collection must exist.
    - the specified frame must exist.
    - the specified slot must **not** exist.

- for building a value :
    - the specified frame collection must exist.
    - the specified frame must exist.
    - the specified slot must exist.

- the specified value must **not** exist.

- for building a relation :
  - the specified frame collection must exist.
  - both of the specified frames being related must exist.
  - the specified relation must exist.

- for destroying a frame collection :
  - the specified frame collection must exist.

- for destroying a frame :
  - the specified frame collection must exist.
  - the specified frame must exist.

- for destroying a slot :
  - the specified frame collection must exist.
  - the specified frame must exist.
  - the specified slot must exist.

- for destroying a value :
  - the specified frame collection must exist.
  - the specified frame must exist.

- the specified slot must exist.
- the specified value must exist.

- for updating a frame collection :
  - the specified frame collection must exist.

- for copying a frame :
  - the specified frame-collection-from must exist.
  - the specified frame-collection-to must exist.
  - the specified frame-from must exist.
  - the specified frame-to must **not** exist.

Inheritance Checks :

These checks are concerned with the determination of whether the object of a specified action is inherited or local. For certain actions, this determination can be crucial in differentiating permissable and non-permissable actions. A typical example would be the attempt to delete an inherited slot from a frame.

At the present time, the inheritance checks are limited to checking the inheritance status of slots, that are to be deleted.

Error-Handling:

The error handling facility of Escher is outlined below in Fig. 4.19. Every call to check-and-implement-1-action will produce a set of action-checks. The result of all of the action checks will indicate that all the checks were passed (t) or there was a check failure (NIL). In the case of a failure, the error-handling routines are triggered which, in turn, sets a special frame, called a problem flag, to record the error and then print out a related message to the user.

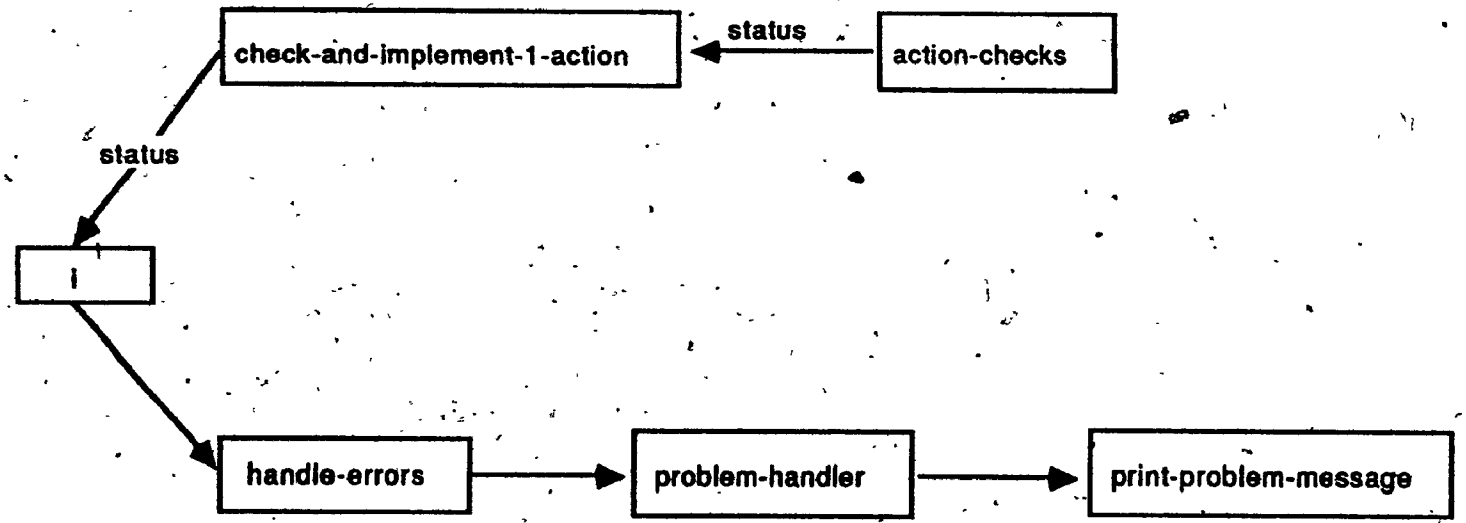


Fig. 4.19 - Schematic of Error-handling Mechanism

The *problem-flag* is related to the *action flag* (described below in this section) and examples of each are shown below in figs 4.20 and 4.21.

The problem flag consists of the following descriptive slots:

switch - this is the switch activated by the error-handler to indicate that an error condition has been triggered

structure - this is the type of structure (ie. frame, slot, etc ...) which was part of the specified command and is obtained directly from the action command

object - this is the name of the object being acted upon (ie. the name of the frame, slot, ...)

suffix - this is the suffix (for use in the error message to be displayed) associated with the particular error type which was triggered, and is set during the error-checking procedure as previously described

code - this is a code used to describe the specific type of error which was flagged during the check, as previously described

The example shown in fig. 4.21 involves the attempt to build a frame called " TEST " at a time when such a frame already exists. At the time that the command " *Build Frame Test in Knowledge* "is entered the action-flag is filled (as described below) as shown in Fig 4.21.

The action-checks are implemented for that command and the non-existence check for the frame " test " returns a NIL to indicate that the frame already exists. This NIL is then passed up through existence-checks, conflict-checks and, finally, action-checks. (see Fig h). NIL is then returned to the check-and-implement-1-action which finally passes it back to the interface controller *i*. From the interface, with the status set to NIL, the handle-errors is called.

The handle-errors will, in turn, call the problem-handler, and this function sets all of the appropriate values for the action flag and calls the print-problem-message, which will display a natural language type of diagnostic message to the user.

Problem-Flag	Action-Flag
Full-structure	Action
Switch	Full-structure
	Structure
Suffix	Knowledge-Base
Type	Frame-collection
Structure	Frame-collection-to
Code	Frame-collection-from
Object	Frame
Level	Frame-to
Slot	Frame-from
	Slot
	Value
	Position

Fig. 4.20 - Structure Of Problem Flag and Action Flag

Problem-Flag

Switch	OFF
Full-structure	NIL
Structure	FRAME
Suffix	ALREADY-EXISTS
Code	CONFLICT
Object	TEST
Level	DUMMY
Slot	NIL

Action-Flag

Action	BUILD
Full-structure	FRAME
Structure	FRAME
Knowledge-Base	SAMPLE
Frame-collection	KNOWLEDGE
Frame-collection-to	NIL
Frame-collection-from	NIL
Frame	TEST
Frame-to	NIL
Frame-from	NIL
Slot	NIL
Value	NIL
Position	NIL

Fig. 4.21 - Structure Of Problem Flag and Action Flag

CHAPTER V - Conclusions

5.1 Test Run :

The development of Escher could not be complete without a realistic testing of the system. Such a testing was set up and implemented so as to determine the level of performance of Escher in the hands of a domain expert with no experience in knowledge base design and development.

A detailed outline of the strategy and application of this project is presented in the paper " SELECT - HVAC: KNOWLEDGE-BASED SYSTEM AS AN ADVISOR TO CONFIGURATE HVAC SYSTEMS " [28] which, at the time of this writing, was submitted to be presented at the ASHRAE Conference, Ottawa 1988. A brief summary follows below, although, a copy of the complete paper can be found in appendix I of this report.

5.2 SELECT-HVAC :

The objective of this project was to build an expert system in the selection and configuration of Heating, Ventilation and Air Conditioning equipment, hence the name SELECT-HVAC. In addition to this objective, the vast majority of the domain knowledge was to be entered directly by the expert on H.V.A.C equipment whose A.I. training and background consisted, solely, of a ten-minute tutorial on the use of Escher.

The total period of time involved, on the part of the domain expert, was about ten weeks. Five weeks were spent designing a system on paper and the remaining five were spent entering the knowledge.

A large portion of the expertise, though not all of it, was entered directly by the domain expert and without assistance from the knowledge engineer.

The use of Escher by the domain expert proceeded very smoothly and little additional training or supervision was required.

5.3 Future Work :

At this point, with Escher now tested and ready for use, the development of several expert systems in the area of building engineering is one of the next logical steps. A central site, offering both a frame based shell and the Escher front-end to be shared among several users, should allow for the relatively fast development of several expert system protoypes.

Such work would accomplish two main goals. Firstly, it would allow for the development of expert systems in a reasonably quick and efficient manner, producing results without the excessively high cost generally associated with expert system development. Secondly, it would make possible an evaluation of Escher by the users, on a general enough

basis, so as to produce a fairly good indication of which features or capabilities would be most useful as an addition to the present version of Escher.

5.4 Conclusion :

In conclusion the following main points may be made regarding Escher :

1) On the basis of the test run results, Escher can be considered a success, in that, it successfully allowed the bypassing of the expertise transfer bottleneck.

2) Though certainly far from complete, Escher provides a framework onto which addition and modifications can be made that should provide an increasingly user-friendly and productive knowledge base development tool. This could include, for certain applications, enhanced graphical capabilities that have proven very helpful in engineering applications.

Finally, it should be noted that the process of developing ESCHER was, in itself, an exercise well suited to developing a greater and, more in depth understanding of the

problems and possible solutions to the problem of the knowledge engineering bottleneck. Certainly, the focus of attention on pre-processing of commands, (ie. action-constraint checks) could be exteneded with useful results.

No doubt, at some point in the future, knowledge acquisition will be a process that is much more highly automated than it is today. In addition, the domain expert will have a much more direct role in the development of the knowledge base than is common practice today. To this end ESCHER has been one small step.

References

[1] Randall Davis, Douglas B. Lenat, Knowledge Based Systems in Artificial Intelligence, 1982, McGraw-Hill International Book Company.

[2] Knowledge Craft, Introduction to Documentation Set, Febuary 27, 1987., vers. 3.1. 1987 C.G.I.

[3] W.J.Clancy, The Epistemology of a Rule-Based Expert System -a Framework for Explanation, 1983.

[4] James R. Miller, Human-computer interaction and intelligent tutoring systems, Microelectronics and Computer Technology Corporation 1987.

[5] Marilyn Golden, Ronald W. Siemans, Jay C. Ferguson, What's Wrong With Rules ?, 5/1986 IEEE., Ford Aerospace & Communications Corporation, Sunnyvale Operation, 1260 Crossman Avenue, Sunnyvale, California.

[6] Randall Davis*, Interactive Transfer of Expertise: Acquisition of New Inference Rules, Computer Science Department, Stanford University, Artificial Intelligence

12 (1979), 121-157, Copyright 1979 North Holland Publishing Company.

[7] Grant Buckler, Artificial Intelligence Today, InfoAge June 1985.

[8] Construction Leads Way in Expert System Software, Building March 1986.

[9] Hon Wai Chun, Alejandro Mimo, Harry Wu, ISCS - An Intelligent System Configuration Shell, 1986 ACM Computer Science Conference, Cinncinnati, Ohio., SCOS Advanced Systems Technology, Honeywell Information Systems, 300 Concord Road, Billerica, MA 01821

[10] Micheal A Rosenman, John S. Gero, Peter J Hutchinson, Rivka Oxman, Expert systems applications in Computer-Aided Design, Computer Aided Design, Vol 18, No. 10, December 1986.

[11] Leon Sterling, Meta-Interpreters for Expert Systems, Department of Computer Engineering and Science and Center for Automation and Intelligent Systems Research, Case Western Reserve University, Cleveland, Ohio, 44106, Proceedings of 1986 ACM Computer Science Conference, Cinncinnati, Ohio.

[12] Alex Bykat, Designing an Intelligent Operating System Consultant and Teacher , Center of Excellence for Computer Applications, University of Tenesee at Chattanooga, Chattanooga, TN 37402, Proceedings of 1986 ACM Computer Science, Conference, Cinncinnati, Ohio.

[13] Chuck Williams, Software tool packages the expertise needed to build expert systems, Inference Corporation, Electronics Design,August 9, 1984.

[14] M.L.Maher, D.Sriram, S.J.Fenves, Tools and Techniques for Knowledge Based Expert Systems for Engineering Design. Department of Civil Engineering, Carnegie-Mellin Univeristy, Pitsburgh, PA 15213, USA, Advanced Engineering Software, 1984, Volume 6, No. 4.

[15] K.J.SiddiquiD.R. HayC.Y.Suen, Evaluation of Materials Using A Rule-Based Non-Destructive Monitoring System. University of Nebraska, Omaha, NE 68182 USA, Tektrend Int. Inc., 8200 Decarie, Montreal, Canada, Concordia University, 1455 De Maisonneuve W., Montreal, Canada, Proceedings of 1986 ACM Computer Science Conference Cinncinnati, Ohio.

[16] D.A. Waterman, A Guide to Expert Systems, Addison-Wesley, 1985.

[17] F.Hayes-Roth et al, Building Expert Systems, Addison-Wesley, 1983.

[18] Daniel C. St. Clair, Virginia Johnson, Albert Wetterstoem, UIL - ESP - AD: An Expert System for Analyses and Diagnosis of a Space Station Power Subsystem, University of Missouri-Rolla, Graduate Engineering Center in St. Louis, Nasa/Johnson Space Center, Houston, Texas, Proceedings of 1986 ACM, Computer Science Conference, Cinncinnati, Ohio.

[19] Eugene Lindon, Intellicorp: The Selling of Artificial Intelligence, High Technology 1985.

[20] William A. Kornfield, The Purpose and Promise of Logic Programming, Quintus Computer Systems, Inc., Proceedings of 1986 ACM Computer Science Conference, Cinncinnati, Ohio.

[21] Kenneth A.Bowen, New Directions in Logic Programming, Logic Programming Research Group, School of Computer and Information Science, Syracuse University, Syracuse, NY,

13210 U.S.A., Proceedings of 1986 ACM Computer Science Conference, Cinncinnati, Ohio.

[22] Micheal A Rosenman , John S Gero, Design Codes as Expert Systems, Computer-Aided Design, Volume 17 number 9 november 1985, Butterworth & Co.(publishers) Ltd.

[23] Hitoshi Furuta, King-Sun Tu, James T P Yao, Structural Engineering Applications of Expert Systems, Computer-Aided Design, Volume 17 number 9 november 1985., Butterworth & Co.(publishers) Ltd.

[24] A Borning, THINGLAB - a constraint-oriented simulation laboratory, Technical Report, STAN-CS-79-746, Stanford University (July 1979).

[25] I E Sutherland, SKETCHPAD: a man-machine graphical communications system , PhD Thesis, MIT, Cambridge, MA, USA 1963.

[26] J. Bennett, L. Cleary, R. Englemore, R. Melosh, SACON : A Knowledge-Based Consultant For Structural Analysis, Technical Report STAN-CS-78-699,Stanford University, September, 1978.

[27] John S. Gero, Richrad Coyne, The Place of Expert Systems in Architecture, Computer Applications, and Research Unit, Department of Architectural Science, University of Sydney, N.S.W., CAD84 : 6th International Conference and Exhibition On Computers In Design Engineering

[28] P. Fazio, R. Zmeureanu, A. Kowalski, SELECT-HVAC: Knowledge-based system as advisor to configurate H.V.A.C., Equipment, Submitted to ASHRAE Conference, Ottawa 1988.

[29] Clive L. Dym, EXPERT SYSTEMS: New Approaches to Computer-aided Engineering, Knowledge Systems Area, Intelligent Systems Laboratory, Xerox PARC and Department of Civil Engineering University of Massachusetts, Engineering with Computers 1, 9-25 (1985).

[30] P. Norman, Y.W.Voon, Expert Systems in the Selection of Process Equipment, Conference Proceedings 4th International Conference on Engineering Software London, England June 1985. Computational Mechanics Centre, Ashurst, Southhampton, R.A.Adey.

[31] K.J.MacCallum, A.Duffy, An Expert System For Preliminary Numerical Design Modelling, Department of Ship and

Marine Technology, Universiy of Strathclcyde, Glasgow, Conference Proceedings 4th International Conference on Engineering Software London, England June 1985., Computational Mechanics Centre, Ashurst, Southhampton., R.A.Adey

[32] Janet Marjorie Gould, Artificial Intelligence: A Tool For System Dynamics, Massachusetts Institute of Technology

[33] Gary Kahn, Steven Nowlan, John McDermott, Strategies For Knowledge Acquisition , IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol PAMI-7 No. 5 September 1985.

[34] Larry Eshelman, John McDermott, MOLE: A Knowledge Acquisition Tool, Department of Computer Science, Carnegie-Mellon University, Proceedings AAAI-86, Fifth National Conference on, Artificial Intelligence, August 11-15, 1986.

[35] Thomas Maples, Jerome J. Connor, A Knowledge Based System For Preliminary Design of Plate Girders, Department of Civil Engineering, Massachusetts Institute of Technology, Cambridge Massachusetts, Microsoftware For Engineers, 1986, Vol. 2 No. 2.

[36] Robert B. Terwilliger, Roy H. Campbell, PLEASE : Predicate Logic based Executable Specifications, Department of Computer Science, University of Illinois at Urbana-Champaign, 252 Digital Computer Laboratory,1304 West Springfield Avenue, Urbana, Illinois, Proceedings of 1986 ACM, Computer Science Conference, Cinncinnati, Ohio.

[37] Richard Forsyth, Software Review - Personal Consultant Plus, Expert Systems, October 1986., Volume 3, No. 4 .

[38] ART Reference Manual Vols I and II, Inference Corporation, vers. 3.0., January 1987.

[39] Bruce D. Clayton, ART Programming Tutorial " Volume One: Elementary ART Programming" Inference Corporation, Vers. 2.0 and 3.0

[40] Bruce D. Clayton, ART Programming Tutorial " Volume Two: A First Look at Viewpoints" Inference Corporation, Vers. 2.0.

[41] Bruce D. Clayton, ART Programming Tutorial " Volume Three: Advanced Topics in ART" Inference Corporation, Vers. 2.0.

[42] KEE Software Development System User's Manual, Intellicorp, Vers 3.0, July 25, 1986

[43] S.1 Development Reference Manual, Teknowledge, December 1986.

[44] M.1 Reference Manual, Teknowledge, June 1985.

[45] Goldworks Expert System User's Guide, Goldhill, Vers 1.0, 1987.

[46] Nexpert Object Fundamentals, Neuron Data Inc.,Vers 1.0, 1987.

[47] Lee Brownston, Robert Farrell, Elaine Kant, Programming Expert Systems in OPS5, Addison-Wesley, 1985.

[48] Forgy,C.L., OPS5 User's Manual, Carnegie-Mellon University, July 1981.

Appendix I

SELECT - HVAC: KNOWLEDGE-BASED SYSTEM AS AN ADVISOR TO CONFIGURE HVAC SYSTEMS

P. Fazio, Ph.D., P.E. ASHRAE Member
R. Zmeureanu, Ph.D., P.E. ASHRAE Associate Member
A. Kowalski, B.Cs.

ABSTRACT

This paper presents a knowledge-based system that has been developed to be used as an advisor in the preliminary design stage of HVAC systems. It enables designers to configure and size HVAC equipment for different climatic conditions, in terms of building type, indoor requirements, outdoor and indoor air pollution. A new approach has been used in building the system, using a modified knowledge engineering methodology and a menu-driven front-end to a full frame based system.

INTRODUCTION

Due to the nature of the construction industry, the use of Knowledge-Based Systems have a large potential of application. In the case of small projects (e.g., residential buildings) or of specific activity (e.g., selection of building materials or HVAC equipment), the use of Knowledge-Based Systems by contractors provides them with an

---

P. Fazio is a Professor and Director, Centre for Building Studies, Concordia University Montréal; R. Zmeureanu is Assistant Professor, Centre for Building Studies, Concordia University, Montréal; A. Kowalski, Specialist in Knowledge Engineering, SIRICON, Montréal.

intelligent advisor, which can help to increase the quality and the productivity of their work, and also can avoid their dependence on consultants in field. In the preliminary design of large buildings the architects/engineers can use Knowledge-Based Systems for detailed and fast estimation of the project (human resources, financial support, equipment, schedules etc.) taking into account all aspects such as architectural, mechanical, electrical or economical. In both cases, the Knowledge-Based Systems incorporate recommendations derived from technical literature and correlated with the standard requirements. Also, the specific knowledge of professionals, which has been gained in time and usually cannot be found in textbooks, is incorporated.

The general idea of using Knowledge-Based Systems has produced many expectations among professionals related to the construction industry (Ruberg and Sander 1986; Wright 1985). These systems seem to have a large potential of application in all design stages (programming, conceptual design, preliminary design and detailed design), in financial planning and project management, in construction activities, in building operation and in diagnostics of building problems.

In the past few years several Knowledge-Based Systems related to the building industry have been developed or are presently under development, and they can be classified as follows:

a. Diagnostic tools for
   - moisture damage related problems in buildings (DAMP) (Sachdeva 1985),

- damage assessment of protective structures (DAPS) (Ross et al. 1986),
- problems in air handling units (Brothers 1987; Leah 1986),
- building air infiltration (Read and Persily 1986),
- preventive maintenance of large air conditioning equipment (Edman 1987),
- window problems (Ruberg and Sander 1986).

b. Design tools for

- preliminary structural design of hi-rise buildings (HI-RISE) (Maher and Fenves 1985),
- conceptual design of building energy systems (Monaghan and Doheny 1986),
- retaining walls (RETWALL) and kitchen layout (Roseman et al. 1986),
- design and selection of brickwork cladding (Cornick and Bowen 1986),
- selection of materials for the building envelope (Delcambre and Halleux 1986),
- earthquake resistant elements (Miyamura et al. 1986),
- small office buildings (HVAC Expert) (Schmitt 1987),
- selection of energy conservation strategies (Krajnovici 1987),

- estimation of building energy consumption using database obtained by simulation with the DEROB program (EDP) (Hand and Higgs 1986).

c. Interpretation of the monitored data for
   - analysis and diagnosis of the problems that cause abnormal energy consumption in buildings (Haberl and Claridge 1987),
   - estimation of the end-uses from monitored energy consumption (Akbari 1987).

d. Interface between engineers and complex computer programs such as SACON which has been developed to be used with the finite element program called MARC (Wager 1984).

In this paper, the work carried out at the Centre for Building Studies, Concordia University, by a team incorporating professionals in building and mechanical engineering and computer science is presented.

The following are the objectives of the work:

- To develop a prototype expert system to be used as an advisor in the preliminary design and configuration of HVAC equipment.
- To test the functionality of a new approach to the build
- The term "full frame based shell or system" refers to a system which has the capability of implementing

frames and the full range of features associated with semantic networks.

These features include value constraints, meta-knowledge, full hierarchial inheritance, multiple inheritance, and customized relations just to name a few. Without these features the development of a large and sophisticated semantic network model is virtually impossible.

The term was intended so as to differentiate the system used from other which do not offer the above mentioned features and can be classified as no-frame (eg. guru) or partial frame (eg. ops5).

CONFIGURATION OF HVAC EQUIPMENT

The configuration and size of the HVAC equipment depends on several factors such as:

- building thermal loads (heating, cooling),
- indoor design conditions (temperature, humidity, air
- quality), type of building, activity and schedule of operation,
- available sources of energy.

The designer has to select among the available components (e.g., filters, heating and/or cooling coils, steam humidifier, spray-type air washer, mixing box, fan) and to arrange them in an appropriate sequence. Usually, the psychrometric charts are used to represent the heat

transfer processes occuring when the air is passed through these components. However, several other parameters, which are not included on these charts, are of importance in the selection of the type and size of the HVAC components. The rules of selection of such parameters are dissipated throughout the technical literature, or are derived from long time experience in the design of the HVAC systems. Also, data about the indoor and outdoor design conditions in terms of the type of space and building location are required. Requirements from standards such as minimum air ventilation rate should, also, be available.

The prototype of the Knowledge Based System presented in this paper has been developed as an interactive advisor to configure HVAC equipment. It can be used in the preliminary design stage, where the type and size of the equipment is of interest to establish the required budget of the project. Later on, during the detailed design stage the equipment is selected from manufacturer's catalogues based on the information provided at the preliminary stage.

The software contains seven major blocks which correspond to the major steps in selecting the HVAC equipment:

i) outdoor design conditions,

ii) indoor design conditions,

iii) pre-heating,

iv) filters,

v) air flow rates,

vi) mixing between the outdoor air and the recirculated air,

vii) psychrometric processes.

The sensible and latent space thermal loads should be provided by the user, who can use any available methods of estimation.

The data about the outdoor design conditions (ASHRAE 1981) for locations around the world are structured in terms of country, province/state, city and season (summer/winter) (Fig. 1).

The indoor design conditions (ASHRAE 1987) are selected in terms of the destination of space and season (summer/winter) (Fig. 2).

The pre-heating coil is located in the outdoor air stream to prevent the freezing of equipment downstream of the coil and, is selected only if the design outdoor air temperature is lower than the desired pre-heating temperature (Fig. 3). The need for a heating coil to be located in the mixed air stream is analyzed within the seventh major block (psychrometric processes).

The selection of the most appropriate location for filters depends on factors such as:

- destination of space (clean room, comfort, industrial),
- mixing between the outdoor and the return air,
- use of pre-heating,
- level of outside pollution,
- risk of freezing fog.

For example, filters can be located before the pre-heating coil if

high level of pollution by dust is expected to occur outside (F1) or immediately after the coil if the risk of freezing fog is high (F2) (Fig. 4). Depending on the nature and level of pollution in the indoor air, filters can be located on the recirculated air duct (F3). In case of clean rooms, several stages of filtration are required, including high efficiency filters as close to the room as possible (F5).

The filters can be of panel or renewable type, and of permanent or throw-out type. Other types of filters such as electronic air cleaners can, also, be implemented.

The air ventilation rate (outdoor air) is defined in terms of the destination of space and of the standard requirements such as outdoor air rate per person or per square foot (Fig. 5), while the supply air to the space is calculated either in terms of sensible and latent heat gains, temperature difference between supply and room air, or using standard requirements such as air changes per hour.

The knowledge-based system will define if the recirculation of the return air is accepted, based on the type and level of indoor pollution (dust, smokes, gases, bacteria, radioactivity), and of the availability of equipments to separate these pollutants from the return air. Based on the decision to allow or not the mixing, the temperature of mixed air is calculated (Fig. 6).

The selection and location of all other components is carried out using the psychrometric representation of the heat transfer processes within the HVAC equipment.

The present prototype contains information for the configuration of single-duct constant volume systems. The type of representation can be classified in three categories in terms of the relative position of points defining the outdoor (O) and indoor (R) conditions, as follows:

1) $T_O \quad T_R$ and $W_O \quad W_R$

2) $W_O = W_R$

3) $T_O \quad T_R$ and $W_O \quad W_R$

As an example, the first group corresponds to the winter design conditions for locations such as Montreal or Paris, the second group to the summer design conditions on the same locations, and the third group corresponds to the summer conditions for cities such as Las Vegas, Nevada (U.S.A.), Baghdad (Iraq) or Katmandu (Nepal).

The representation of processes for the first group depends on factors such as (Fig. 7):

- mixing or no mixing,
- humidity control or no humidity control,
- type of humidity control (steam humidifier or spray-type air washer).

As an example, the succession of components within the HVAC equipment, in the case of mixing and humidity control by steam humidifier is: pre-heating coil, mixing box, heating coil, supply fan, and steam humidifier. (Fig. 7.b).

In the case of mixing and humidification by water spray in evaporative mode, there are two different sequences depending if the enthalpy of mixing point M is greater or smaller than the enthalpy of point U (air leaving the water spray). If h_M h_U, then the succession of HVAC components is: preheating coil, mixing box, heating coil, water spray and re-heating coil. If $h_M = h_U$, then the mixed air is passed directly through the water spray and and there is no need for heating coil. If h_M h_U, then the mixing point M is modified to obtain $h_M = h_U$ for avoiding cooling in winter. This is obtained by increasing the outdoor air rate:

$$\dot{m}_O = \dot{m} \frac{T_R - T_M}{T_R - T_P}$$

where

$\dot{m}_O$ = outdoor air rate

$\dot{m}$ = supply air rate,

T_R = room air temperature

T_M = desired temperature of the mixed air

T_P = air temperature after the preheating coil

Besides these components the filters are located in function of the level of outdoor and indoor air pollution, and the air quality requirements.

For each case, the heating or cooling capacity of coils and make-up water flow rate are calculated, and the type of the heating or cooling media (hot water, steam, electricity, chilled water) is selected in terms of available sources.

The type of processes for the second group is presented in Figure 8, with the following succession of components: mixing box (if mixing is allowed), cooling coil, reheating coil (if is required) and supply fan.

In the case of the third group, if no humidity control is required, a washer spray on evaporative cooling mode is used (Fig. 9). In the case of humidity control, a cooling coil should be located ahead the water spray.

Besides the information about the configuration and capacity of HVAC equipment, the Knowledge Based System provides messages helping the user to draw on psychrometric charts the representation of the thermal processes. This feature is particularly useful for students in training and education, for junior engineers, architects and contractors.

Next developments of this system will incorporate graphical representation on terminal of the psychrometric processes and the explanation of selection on each decision node. Also, other systems such as variable volume, induction or dual duct will be incorporated.

METHODOLOGY OF EXPERT SYSTEM DEVELOPMENT

The development of an expert system, even a prototype, is normally a complex task which involves a team, with one member being the so called domain expert, a specialist in the particular field of interest, in this case, the selection and configuration of HVAC equipment for a building.

The second member of the team is the knowledge engineer, a specialist in the use of knowledge based systems and the tools used to build them.

Conventional Approach

The increasing number of commercially available shell-systems (Horn 1986), have, in some cases, tended to give the impression that the development of an expert system is now a relatively routine and straight forward process (Lange et al. 1986). However the domain expert usually find the procedure much more difficult and time-consuming than expected. (Golden et al. 1986).

The conventional approach to the development of a knowledge based system centers around the transfer of knowledge and expertise from the domain experts to the knowledge engineer (Horn 1986). It is this transfer which represents the major bottleneck in the building of expert systems.

At the initialization of a project the two team members will engage in a set of preliminary meetings in order for the knowledge engineer to obtain a grasp of the basic concepts involved. These are

followed by an extended period of time (usually several months or years), in which detailed information (e.g., knowledge) is explained to the knowledge engineer by the domain expert. Once the knowledge engineer has obtained enough know how he may begin to transfer the knowledge into the form of knowledge representation structures. Eventually enough knowledge will be transferred to form a basic framework for the expert system.

At this point however, the expert system is far from completion. This basic framework is only the beginning. A prolonged series of further meetings will be required to refine and polish this primitive knowledge base to a point at which it is a reasonably functional and accurate simulation of a human expert.

The necessity of this extended series of meetings is due to the complexity of the task of transferring the expertise into a form compatible with expert system development tools.

Alternate Approach

An alternate approach (Williams 1984) to the one described above is to put the emphasis on training the domain expert on the use of the required tools and once he has understanding of it, to allow him to perform the knowledge engineering. The problem with this approach is that the time and effort involved for the domain expert to obtain a true understanding of the concepts and methods of implementation involved is as significant as is required in the approach described

previously. In other words, the training of a novice presents a bottleneck just as formidable as the transfer of technology does.

New Approach

One of the key aspects of this project was the use of a novel strategy of implementation. The solution to the bottlenecks was found in the form of a hybrid approach. The transfer of expertise involved was extremely minimal because most of the domain knowledge was entered directly by the domain expert. On the other hand the training effort was also kept to a minimum by means of the use of a front-end system which guided the domain expert every step of the way, performing on his behalf all the required implementational checks and operations. This is of extreme significance when dealing with an initial prototype that is merely repesentative of a much larger amount of knowledge to follow. With the approach taken for SELECT-HVAC, more knowledge can be added and existing knowledge modified, by the domain expert himself as he sees fit.

This front-end system is called ESCHER.

ESCHER

The strategy behind ESCHER (Expertise Structure Class and Hierarchy Engineering Resource) is that it has 2 responsibilities.

1. to display available actions in a menu driven form thereby avoiding the syntactical restraints which normally accompany the initial use of a large system (such as Knowledge-Craft).

2. to perform various types of procedural and domain constraint checks on all specified actions.

For the purposes of the building, implementing and testing of the SELECT-HVAC preliminary-prototype the schema based system of Knowledge Craft (Knowledge Craft 1987) was used on the VAX-8650 mini-computer.

Knowledge Craft is a knowledge engineering environment. It offers two structure types to represent knowledge. These are rules and schemata. For the building of the SELECT-HVAC prototype, it was decided that only the schema structure would be employed.

A schema is a structured representational object which has associated with it a name and, optionally, a set of one or more slots. In addition, each of the slots associated with it may contain a set of one or more values. Together, each schema with its slots and values represents an object or concept. An example shown below is the schema SUMMER, containing information about outdoor design conditions in Montreal in the summer (Fig. 1):

SUMMER

dbtemp	32
mes	wet-bulb temperature is 24
msg	dr-bulb temperature is 32
is-a	montreal

The slot DBTEMP has associated with it the value of 32 degrees centrigrade, which will be used in a later calculation. The slots MES and MSG have associated messages which will be displayed on the sereen.

The slot IS-A indicates that the schema SUMMER is connected to the schema MONTREAL thus creating an inheriting relation between them.

In addition, each schema may be connected to other schema by means of relations. Relations may be of an inheriting or non-inheriting type. Inheriting relations allow the schema which is connected to another to inherit, or automatically obtain the slots and values of the schema it was connected to.

Architecture of SELECT-HVAC

The architecture of SELECT-HVAC, in terms of its knowledge representation, involves the separation of knowledge into 3 separate parts. The purely domain knowledge part of which was set up and implemented directly by the domain expert.

1. Structural domain expertise (semantic network)
   This consists of a collection of schemata representing the various criteria being considered and the attributes and values associated with them. The schemata were organized into a hierarchical framework which served as a basis upon which the control-knowledge could act.

2. Procedural Knowledge (demons)
   This knowledge involved the manipulation of domain specific information within the system. There were discovered to be two basic types of demons required. On the one hand there were straightforward calculation

demons in which the values of slots were calculated from values of other slots within the same schema. On the other hand there were transfer demons in which information from a slot in one schema was transferred to the same slot in another schema. There were, in addition hybrid demons, which are a combination of the two basic types.

This knowledge represented the only area for which any transfer of expertise from the domain expert to the knowledge engineer was required. It should be noted that this knowledge was of a very limited nature and piecemeal in structure and did not require the understanding, on the part of the knowledge engineer, of the overall strategies or concepts. This is important because it allows the design decisions to be in the hands of the domain expert.

3. Control knowledge (lisp functions)

This knowledge, designed completely and solely, by the knowledge engineer, was totally domain independent and as a result it was transparent to the domain expert and his design work, except, of course, in so far as it allowed him easy access to control mechanism, implicit in the schema structures.

The control exists in a set of top-level functions

which analyze each node (schema) which is accessed in order to determine what questions to ask, what questions to print, what calculations and transfers to performs and what schema to access next.

CONCLUSIONS

Upon analysis, the approach to the developing of a prototype expert system as an advisor to configurate HVAC systems produced very favourable results.

Excepting, of course, the time required for the design of hierarchical structure of the domain expertise (5-6 weeks), a prototype was developed in a relatively short period of time (about three weeks). The prototype structure not only allowed for very substantial expansion, but expansion by the domain expert in a manner he saw fit.

The knowledge-based system SELECT-HVAC enable designers to configure and size HVAC equipment for different locations around the world, in terms of indoor requirements, of outdoor and indoor air pollution.

In addition, the capabilities of the system could be further removed by slight modifications to the control knowledge without redesign of the knowledge base.

Based upon these results, further investigation of this approach should prove extremely worthwhile.

ACKNOWLEDGEMENTS

The authors would like to thank both Dr. Renato DeMori of McGill University for his assistance and Centre de Recherche Informatique de Montreal Inc. for providing access to some of the facilities involved in this project.

REFERENCES

Akbari, H., 1987. "Knowledge-Based Software to Identify Building Type and End-Uses from 15-minute Interval Whole-Building Energy Use". Presented in Seminar: Applications of KBS to the HVAC Industry. ASHRAE Annual Meeting, Nashville, TN, June 27-July 1.

ASHRAE 1985. ASHRAE Handbook - 1981 Fundamentals. Atlanta: American Society of Heating, Refrigerating and Air-Conditioning Engineers, Inc.

ASHRAE 1987. ASHRAE Handbook - 1987 HVAC Systems and Applications. Atlanta: American Society of Heating, Refrigerating and Air-Conditioning Engineers, Inc.

Brothers, P.W., 1987. "A Knowledge System for HVAC Diagnostics". Presented in Seminar: Applications of KBS to the HVAC Industry. ASHRAE Annual Meeting, Nashville, TN, June 27-July 1.

Cornick, T., and Bowen, J. 1986. "A Knowledge-based Expert System for Brickwork Cladding Design and Production". Proceedings of the 10th Triennial Congress of the International Council for Building Research, Studies and Documentation, September 22-26, Washington, D.C., Vol. 2, pp. 659-666.

Delcambre, B., and Halleux, D., 1986. "Expert Systems Prototypes Applied to the Building Field". Proceedings of the 10th Triennial Congress of the International Council for Building Research, Studies and documentation, September 22-26, Washington, D.C., Vol. 2, pp. 667-674.

Dry, C. and Givoni, B. 1986. "An Expert System on Bioclimatic and Energy Efficient Building in Hot/Humid Climates". Proceedings of the 10th Triennial Congress of the International Council for Building Research, Studies and Documentation, September 22-26, Washington, D.C., Vol. 2, pp. 675-682.

Edman, T.R., 1987. "A KBS for Preventive Maintenance of Large Air Conditioning Equipment". Presented in Seminar: Applications of KBS to the HVAC Industry. ASHRAE Annual Meeting, Nashville, TN, June 27 - July 1.

Golden, M., Siemens, R.W., and Ferguson, J.C., 1986. "What's Wrong with Rules?" Proceedings of IEEE Conference on Knowledge Based Engineering and Expert Systems. California.

Gupton, G.W., 1987. "An Application of a PC-Based Expert Shell in HVAC & R System Diagnostics". Presented in seminar: Applications of KBS to the HVAC Industry. ASHRAE Annual Meeting, Nashville, TN, June 27 - July 1.

Haberl, J.S., and Claridge, D.E., 1987. "An expert system for building energy consumption analysis: prototype results". ASHRAE Transactions, Vol. 93, part 1.

Hand, J.W., and Higgs, F.S., 1986. "EDT: An expert advisory for energy design of passive and conventional buildings at the sketch design phase". Proceedings of the 10th Triennial Congress of the International Council for Bulding Research, Studies and Documentation, September 22-26, Washington, D.C., Vol. 1, pp. 110-117.

Horn, M.V., 1986. "Understanding Expert Systems", Bantam Books.

Knowledge Craft 1987, Vol. 1 and 2, Version 3.1, Carnegie Group Inc.

Krajnovich, L., 1987. "Development of an Expert System for Energy Conservation Strategy Selection". Presented in seminar: Applications of KBS to the HVAC Industry. ASHRAE Annual Meeting, Nashville, TN, June 27 - July 1.

Lange, R., Hearn, L., and Kearney, F.W., 1986, "The Use of Knowledge Engineering Teams as a Method for the Development of Expert Systems", in Applications of Artificial Intelligence in Engineering Problems. Proceedings of the 1st International Conference, Southhampton University, U.K., Vol. 1, pp. 45-53.

Leah, R., 1986. Johnson Controls Corporation, Milwankee, Wisconsin. Personal communication.

Maher, M.L., and Fenver, S.J., 1985. "Hi-Rise: A Knowledge-Based Expert System for the Preliminary Structural Design of High Rise Buildings". Report No. R-85-146, Department of Civil Engineering, Carnegie Mellon University.

Miyamura, A., Murata, M., and Kato, S., 1986. "Antiseismic Design Support Expert System". Proceedings of the 10th Triennial Congress of the International Council for Building Research, Studies and Documentation, September 22-26, Washington, D.C., Vol. 2. pp. 715-721.

Monaghan, P.F., and Doheny, J.G., 1986. "Knowledge Representation in the conceptual Design Process for Building Energy Systems", in Applications of Artificial Intelligence in Engineering Problems. Proceedings of the 1st International Conference, Southampton University, U.K., Vol. 1, pp. 1187-1192.

Reed, K., and Persily, A. 1986. "KBS applications to interactions among building systems, and a proposed coordinated R & D program". Seminar on research of knowledge-based systems applied to building system design. ASHRAE Annual Meeting. Portland, Oregon.

Roseman, M.A., Gero, J.S., Hutchinson, P.J., and Oxman, R., 1986. "Expert systems applications in computer-aided design". Computer Aided Design, Vol. 18, No. 10, December, pp. 546-551.

Ross, T.J., Wong, F.S., Savage, S.J., and Sorensen, H.C., 1986. "DAPS: An expert system for damage assessment of protective structures". Expert Systems in Civil Engineering. Proceedings of a symposium sponsored by the Technical Council on Computer Practices of the American Society of Civil Engineers. Seattle, Washington, April 8-9, pp. 109-120.

Ruberg, K., and Sander, D.M., 1986. "Information and Knowledge for Building: A Role for Emerging Technology". Proceedings of the 10th Triennial Congress of the International Council for Building Research, Studies and Documentation. September 22-26, Washington, D.C. Vol. 2, pp. 498-505.

Sachdeva, P., 1985. "DAMP - a diagnostic system for architectural moisture damage problems". The Australian Computer Journal, Vol. 17, No. 1, pp. 27-32.

Schmitt, G.N., 1987. "HVAC Expert: A Knowledge Based System for the Design of Small Office Buildings". Presented in seminar: Applications of KBS to the HVAC Industry. ASHRAE Annual Meeting, Nashville, TN, June 27 - July 1.

Wright, R.N., 1985. "AI: Does it have a place in building simulation?". Proceedings of the Building Energy Simulation Conference, Seattle, Washington. August 21-22, pp. 169-173.

Wager, D.M. 1984. Expert Systems and the Construction Association. Guildhall Place, Cambridge, England, pp. 8.

Williams, C., 1984. "Software Tool Packages the Expertise Needed to Build Expert Systems". Electronic Design. August 9.

LIST OF FIGURES

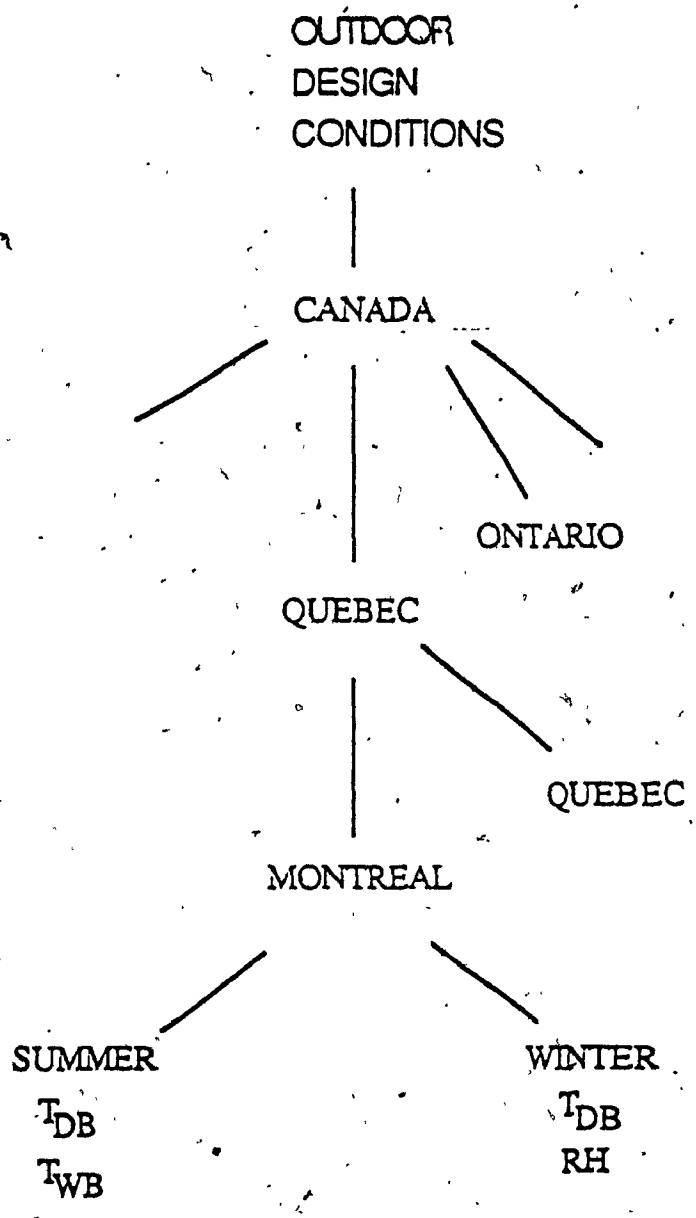

OUTDOOR
DESIGN
CONDITIONS
CANADA
ONTARIO
QUEBEC
QUEBEC
MONTREAL
SUMMER
T_{DB}
T_{WB}
WINTER
T_{DB}
RH

Figure 1

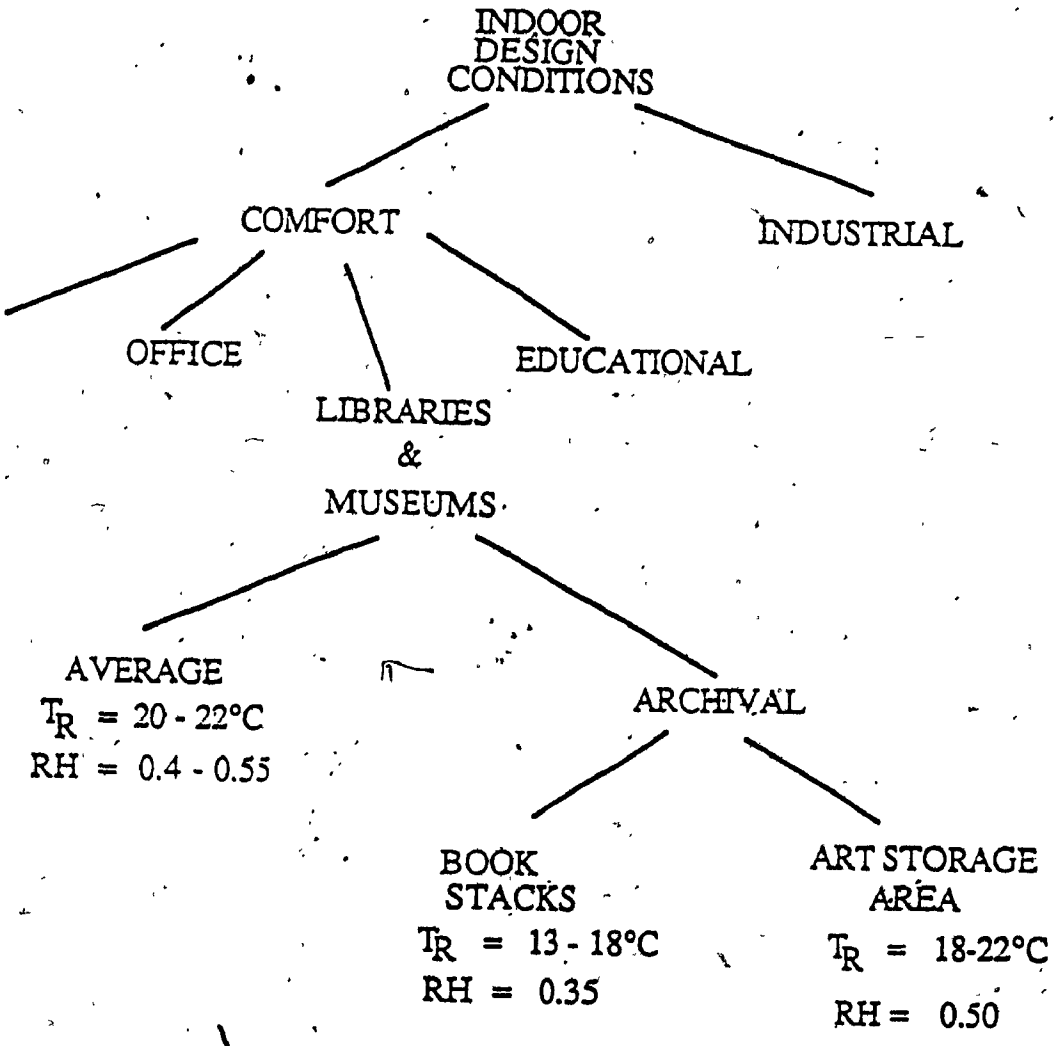

INDOOR DESIGN CONDITIONS
COMFORT
INDUSTRIAL
OFFICE
EDUCATIONAL
LIBRARIES & MUSEUMS
AVERAGE
T_R = 20 - 22°C
RH = 0.4 - 0.55
ARCHIVAL
BOOK STACKS
T_R = 13 - 18°C
RH = 0.35
ART STORAGE AREA
T_R = 18-22°C
RH = 0.50

Figure 2

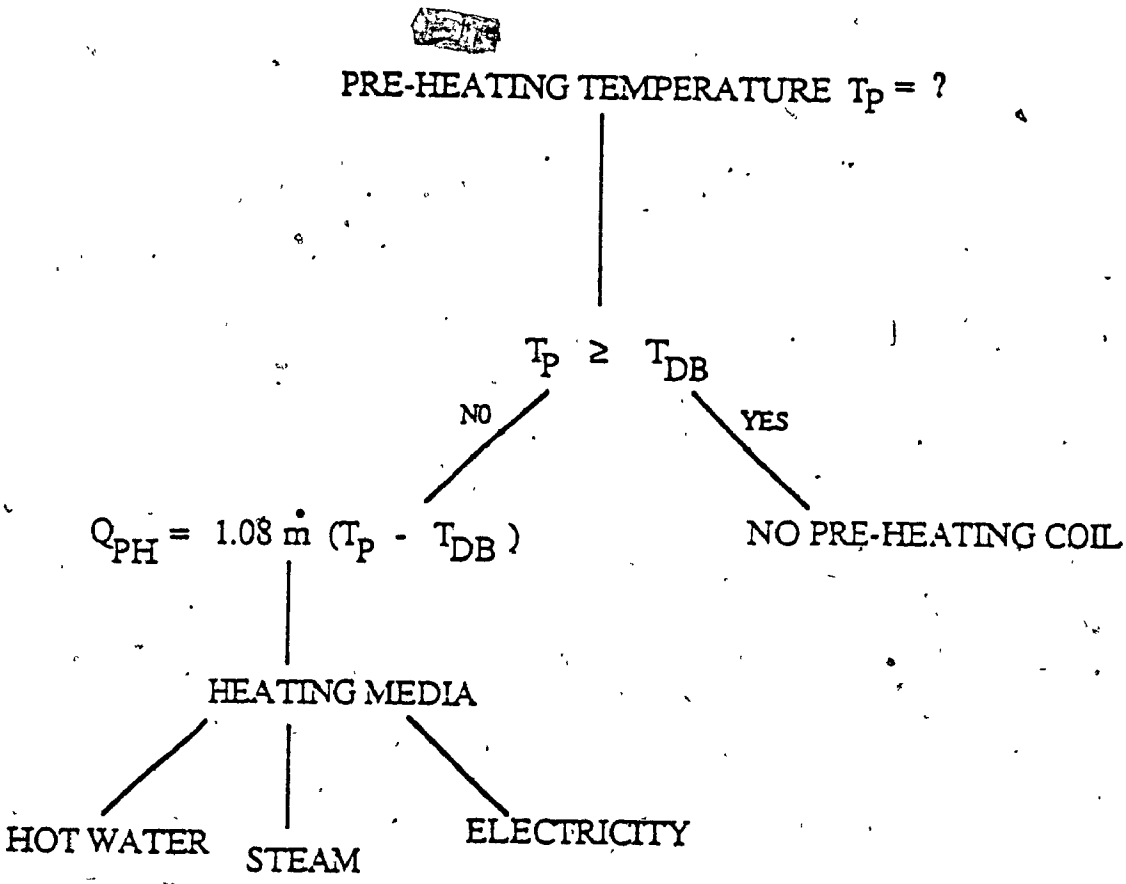

PRE-HEATING TEMPERATURE T_P = ?
$T_P \geq T_{DB}$
NO
YES
$Q_{PH} = 1.08\,\dot{m}\,(T_P - T_{DB})$
NO PRE-HEATING COIL
HEATING MEDIA
HOT WATER
STEAM
ELECTRICITY

Figure 3

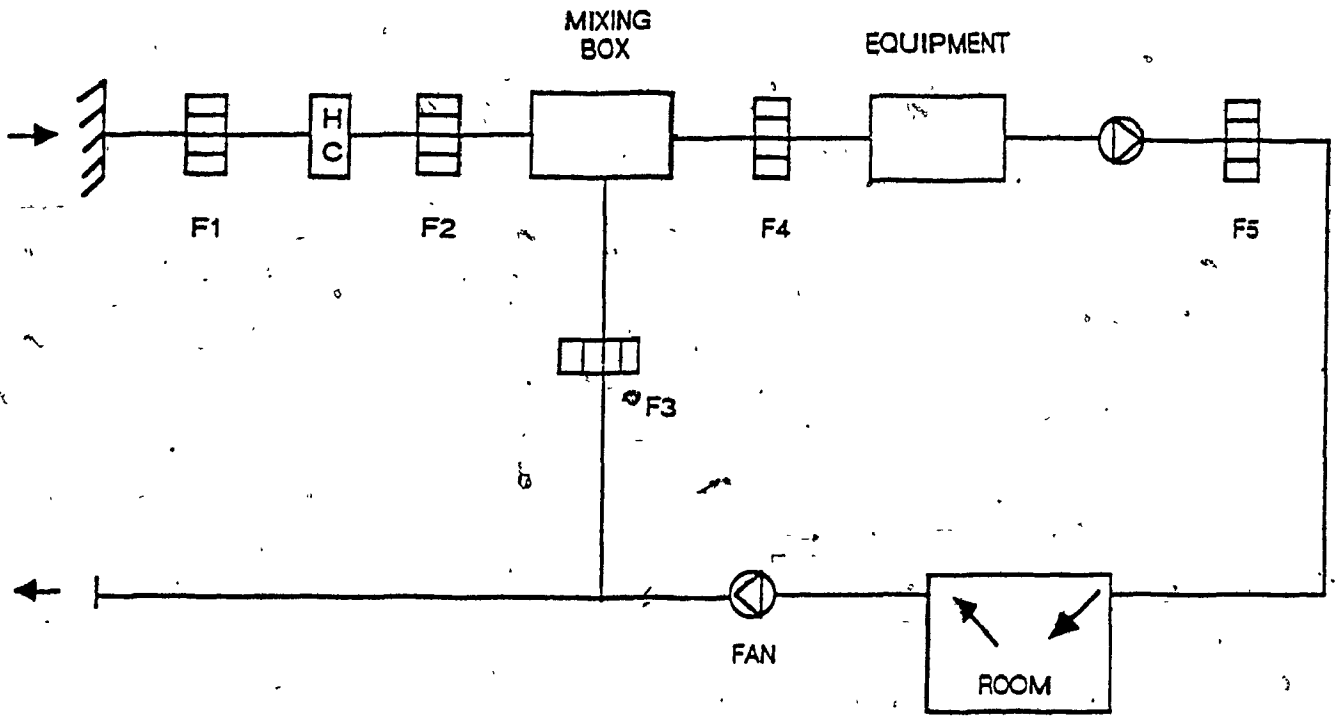


Figure 4

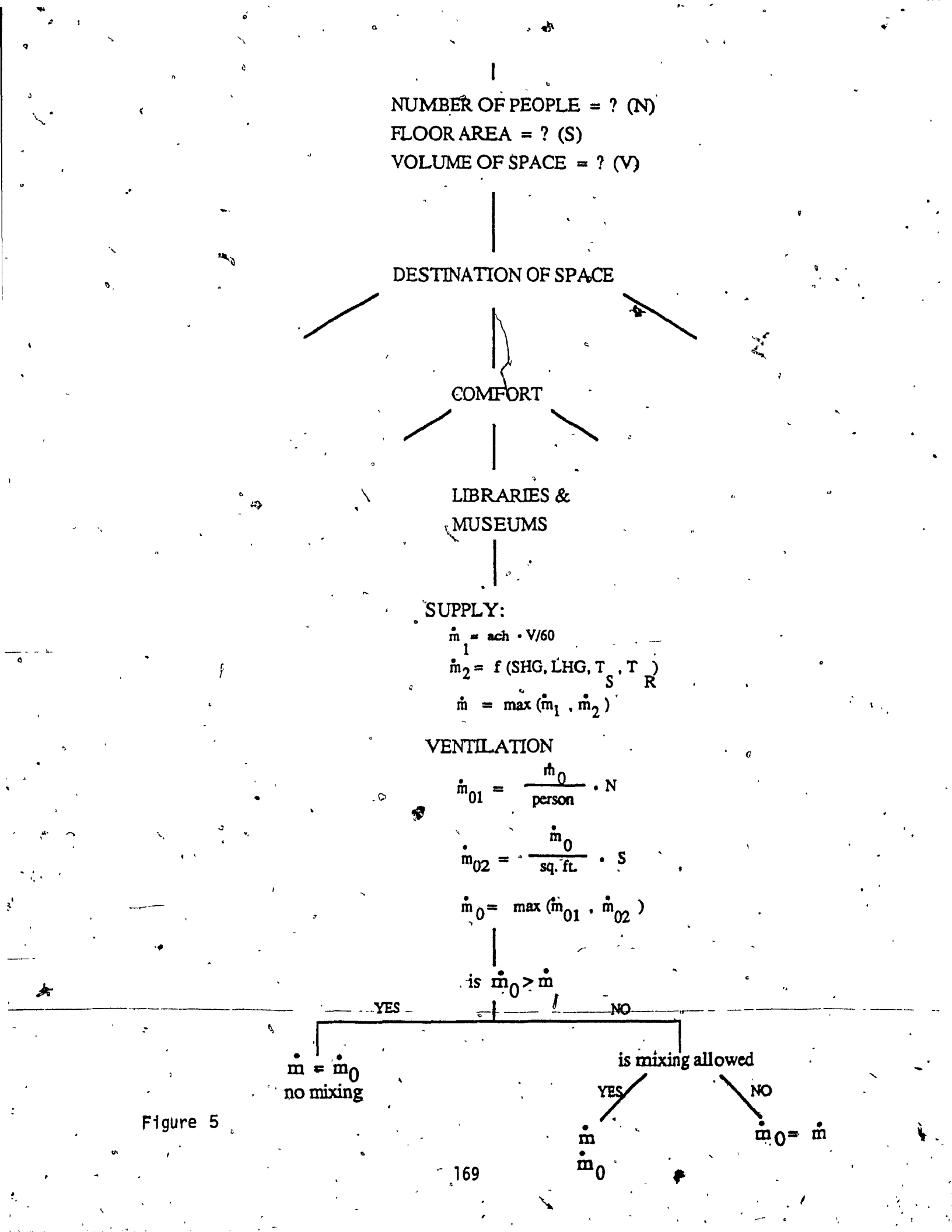

NUMBER OF PEOPLE = ? (N)
FLOOR AREA = ? (S)
VOLUME OF SPACE = ? (V)
DESTINATION OF SPACE
COMFORT
LIBRARIES & MUSEUMS
SUPPLY:
$\dot{m}_1 = ach \cdot V/60$
$\dot{m}_2 = f(SHG, LHG, T_S, T_R)$
$\dot{m} = \max(\dot{m}_1, \dot{m}_2)$
VENTILATION
$\dot{m}_{01} = \frac{\dot{m}_0}{person} \cdot N$
$\dot{m}_{02} = \frac{\dot{m}_0}{sq.\ ft.} \cdot S$
$\dot{m}_0 = \max(\dot{m}_{01}, \dot{m}_{02})$
is $\dot{m}_0 > \dot{m}$
YES
NO
$\dot{m} = \dot{m}_0$
no mixing
is mixing allowed
YES
NO
$\dot{m}$
$\dot{m}_0$
$\dot{m}_0 = \dot{m}$

Figure 5

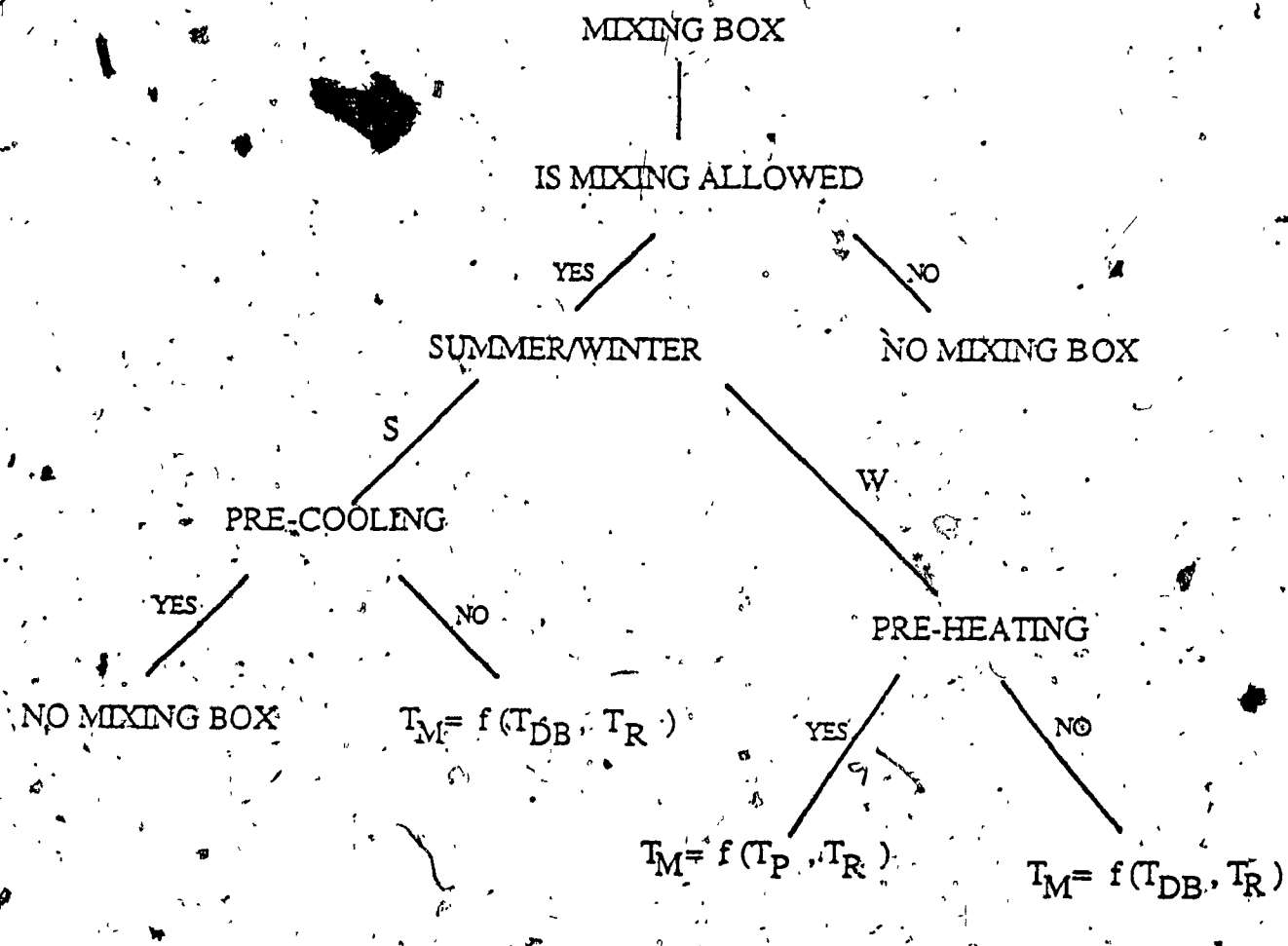
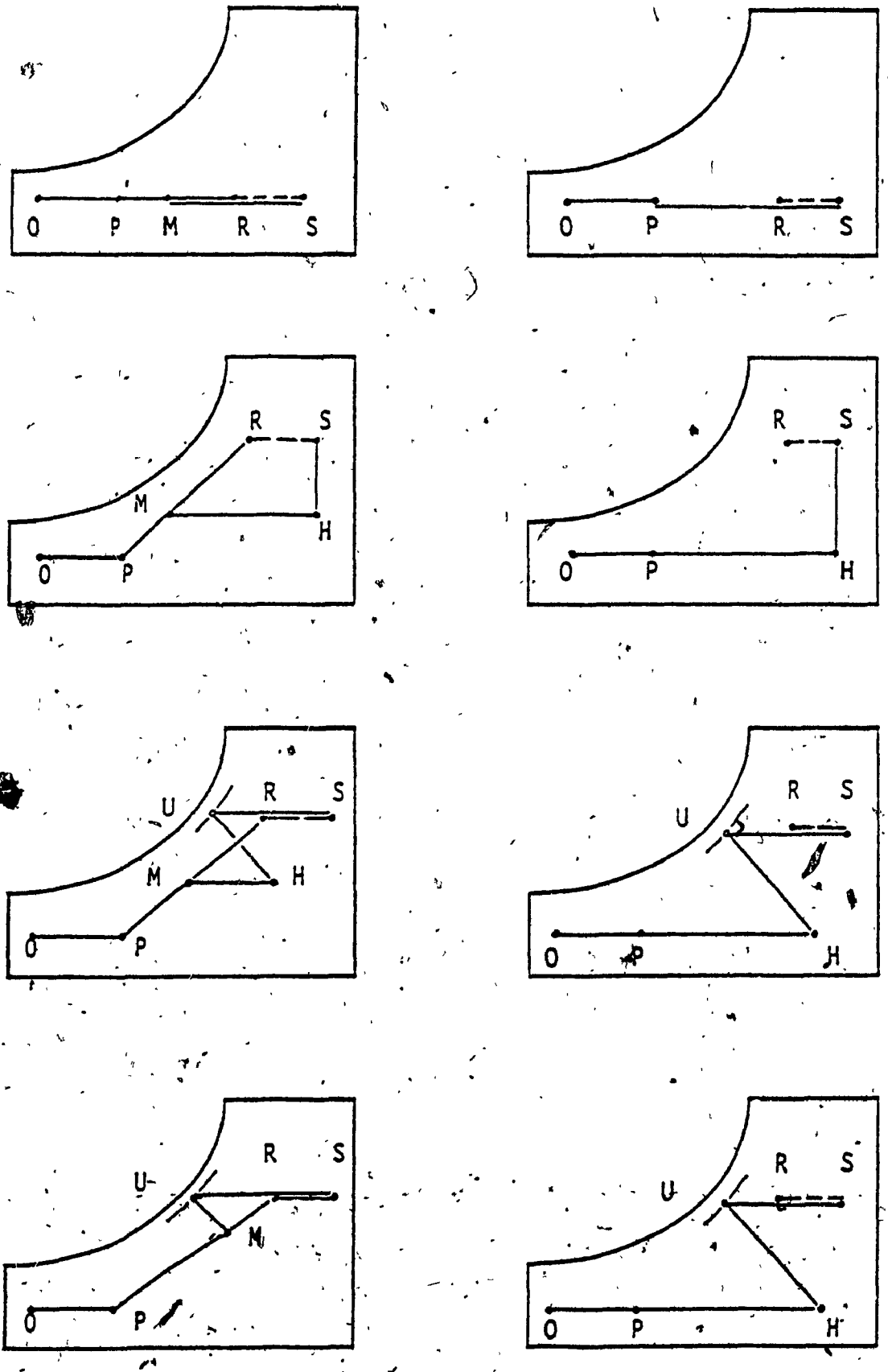
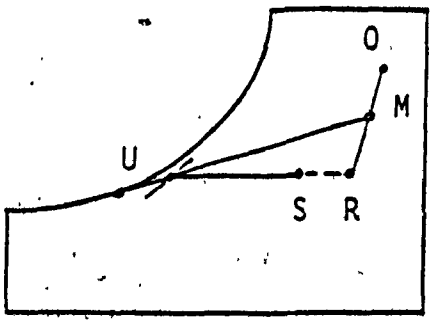

MIXING BOX
IS MIXING ALLOWED
YES
NO
SUMMER/WINTER
NO MIXING BOX
S
W
PRE-COOLING
PRE-HEATING
YES
NO
YES
NO
NO MIXING BOX
$T_M = f(T_{DB}, T_R)$
$T_M = f(T_P, T_R)$
$T_M = f(T_{DB}, T_R)$

Figure 6

Mixing Figure 7 No mixing



Mixing



No mixing

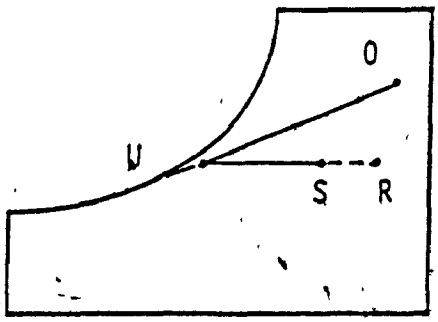
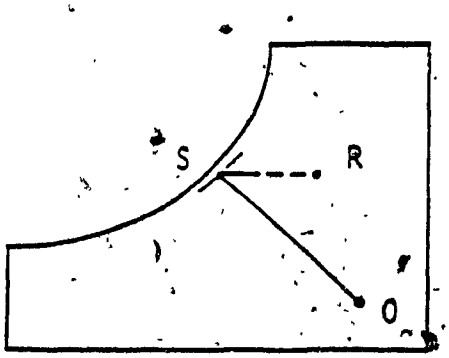
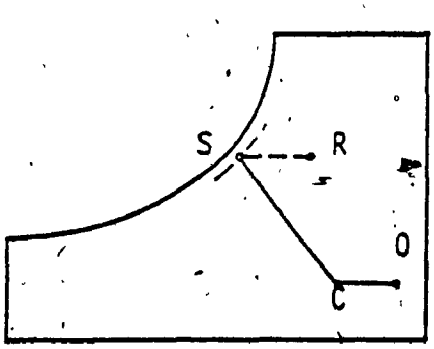


Figure 8

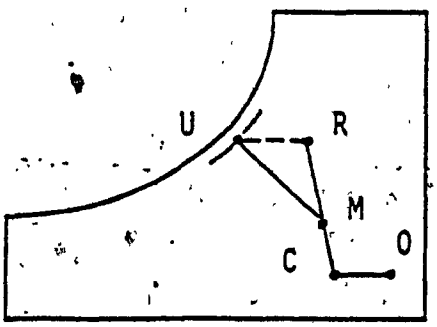
No humidity control



Humidity control



a.



b.

Figure 9