DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA 93943-5002

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

EXPERT DATABASE SUPPORT
OF THE
INTEGRATED MANUFACTURING PROCESS

by

Thomas G. Wilbur Jr.

December 1986

Thesis Advisor: C. Thomas Wu

Approved for public release; distribution is unlimited.

T233779

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION unclassified	1b. RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b DECLASSIFICATION / DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School 6b OFFICE SYMBOL (*If applicable*) 52	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c. ADDRESS (*City, State, and ZIP Code*) Monterey, California 93943-5000	7b. ADDRESS (*City, State, and ZIP Code*) Monterey, California 93943-5000
8a NAME OF FUNDING / SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (*If applicable*)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c ADDRESS (*City, State, and ZIP Code*)	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO / PROJECT NO / TASK NO / WORK UNIT ACCESSION NO

11 TITLE (*Include Security Classification*) EXPERT DATABASE SUPPORT OF THE INTEGRATED MANUFACTURING SYSTEM

12 PERSONAL AUTHOR(S) Wilbur, Thomas G. Jr.

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (*Year, Month, Day*)	15 PAGE COUNT
Master's Thesis	FROM _____ TO _____	1986 December	132

16 SUPPLEMENTARY NOTATION

17 COSATI CODES			18 SUBJECT TERMS (*Continue on reverse if necessary and identify by block number*)
FIELD	GROUP	SUB-GROUP	computer-aided-design; computer-aided-manufacturing; computer-integrated-manufacturing; expert system

19 ABSTRACT (*Continue on reverse if necessary and identify by block number*)

This effort explores the design requirements for an expert translator to be used as an interface between present Computer Aided Design (CAD) and Computer Aided Manufacturing (CAM) systems. The translator's purpose is to perform certain standards checks on the design data and pass assembly information as well as material requirements from CAD to CAM. An example translator was implemented for a simple one room house construction problem using the artificial intelligence language Prolog. This research is part of an effort to design a generic Computer Integrated Manufacturing System in which the design through manufacturing process is totally automated.

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION unclassified
22a NAME OF RESPONSIBLE INDIVIDUAL Prof. C. Thomas Wu	22b TELEPHONE (*Include Area Code*) (408) 646-3391 22c OFFICE SYMBOL Code 52Wq

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE
unclassified

Approved for public release; distribution is unlimited.

Expert Database Support of the Integrated Manufacturing Process

by

Thomas George Wilbur Jr.
Lieutenant, United States Navy
B.S., Jacksonville University, 1974

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 1986

ABSTRACT

This effort explores the design requirements for an expert translator to be used as an interface between present Computer Aided Design (CAD) and Computer Aided Manufacturing (CAM) systems. The translator's purpose is to perform certain standards checks on the design data and pass assembly information as well as material requirements from CAD to CAM. An example translator was implemented for a simple one room house construction problem using the artificial intelligence language Prolog. This research is part of an effort to design a generic Computer Integrated Manufacturing System in which the design through manufacturing process is totally automated.

TABLE OF CONTENTS

I. INTRODUCTION ---------- 6

II. BACKGROUND ---------- 9

III. DATA OUTPUT SPECIFICATION FOR COMPUTER AIDED DESIGN - 12

A. CONCEPTUAL SCHEMA ---------- 12

B. PROTOTYPE ---------- 15

C. SLOT INHERITANCE ---------- 20

D. COORDINATE SYSTEM ---------- 21

IV. DATA REQUIREMENTS FOR COMPUTER AIDED MANUFACTURING - 25

A. ASSEMBLY ---------- 25

B. MATERIALS ---------- 27

V. EXPERT SYSTEM SHELL TRANSLATOR ---------- 30

A. META-KNOWLEDGE ---------- 31

B. ASSEMBLY DATA ---------- 31

C. STANDARDS DATA ---------- 34

D. LANGUAGE OF CHOICE ---------- 35

VI. EXAMPLE EXPERT SYSTEM SHELL TRANSLATOR ---------- 38

A. CAD DESIGN DATA ---------- 38

B. STANDARDS CHECKS ---------- 39

C. PRODUCT ASSEMBLY ---------- 43

D. RAW MATERIALS LISTING ---------- 52

VII. CONCLUSIONS AND RECOMMENDATIONS ---------- 57

A. CONCLUSIONS ---------- 57

B. RECOMMENDATIONS ---------- 58

APPENDIX A ONE ROOM HOUSE DESIGN ---------- 59

APPENDIX B CONCEPTUAL SCHEMA PROTOTYPES ---------- 65

APPENDIX C PROLOG CODE FOR EXAMPLE EXPERT SYSTEM SHELL TRANSLATOR ---------- 74

APPENDIX D EXAMPLE EXPERT SYSTEM SHELL TRANSLATOR OUTPUT DATA ---------- 116

LIST OF REFERENCES ---------- 129

INITIAL DISTRIBUTION LIST ---------- 131

I. INTRODUCTION

In the not too distant future, most factories will be using an integrated, computer aided, design and manufacturing system. There has already been much work performed in Computer Aided Design (CAD) and Computer Aided Manufacturing (CAM) techniques. Although definitions may vary slightly, Ness [Ref. 1] defines CAD and CAM as follows:

> Computer Aided Design is the application of computer technology to the design of a product. This includes layouts, detail design, analysis, drafting and formal release of design data.
>
> Computer Aided Manufacturing is the application of computer technology to the fabrication, assembly and verification of a product that does not depend on and cannot productively use specific CAD data.

Expanding on the above concepts, the CAD process consists of product specification and design. CAD may allow for product simulation and performance analysis using computer models. Some pre-manufacturing testing may also be performed. Product layouts may be generated. Increased interest recently has been focused on the use of interactive graphics with CAD; Beeby [Ref. 2] discusses this issue as applied to the construction of the Boeing 767 airplane. Interactive graphics systems allowed the designers of the Boeing 767 to interact with the design system and data in real time. In addition, some design changes were automatically propagated through the entire design relieving the designer of the update responsibility. Beeby [Ref. 2] cites as an example that changes made to the thickness of the spar chord in the wing of the Boeing 767 produced automatic changes in related items such as ribs and clips without designer intervention.

To expand on the CAM process, it is only necessary to look at some of the methods presently used in the manufacture of the Boeing 767. Computer aided tools locate, drill and fasten wing spars. Robots handle sanding and painting tasks. A self-propelled drilling unit travels through the interior of the plane drilling the more than eight thousand holes that are required for seat installation.

However, it is the lack of integration of the design data produced by CAD into the manufacturing process handled by CAM that is our primary concern. The future goal is to build factories in which a network of computer systems is used to support product design, planning and manufacture and thus facilitate faster and more economical production [Ref. 3]. Current CAD and CAM systems are primarily independent and thus unable to effectively communicate with each other. One solution to this problem is to define a standard data format which will be accessible to both CAD and CAM. With a standardized data format, future CAD and CAM systems, which are built to use the format, will be able to communicate with each other freely.

For example, Boeing Commercial Airplane Company (BCAC) has introduced a standard geometric data format as part of its CAD/CAM Integrated Information Network that has now become the framework for the Initial Graphics Exchange Specification (IGES) [Ref. 4]. More recently, the International Organization for Standardization (ISO) has been developing the Programmer's Hierarchical Interactive Graphics System (Phigs) standard which is based partially on the Graphical Kernal System

(GKS). Both of these systems demonstrate the intense interest in forcing standardization of data.

Another solution to the communication problem between CAD and CAM is to design a communications interface that could be used with current CAD and CAM systems with little change in their structure. The advantage to this method is there is no massive redesign of CAD and CAM systems required to meet the requirements of a new standard data format.

The purpose of this research is to define the software and data requirements that are necessary to integrate CAD and CAM into a fully compatible system by the use of such an interface. We shall call this a Computer Integrated Manufacturing System. The interface will consist of an expert system translator used to link the CAD output data and CAM input data. An example translator will be presented and discussed. This translator will be an enhanced version of the translator first presented in [Ref. 5].

II. Background

The Expert System Shell Translator design to be proposed in this paper will be based on the concepts of Computer Integrated Manufacturing (CIM) as described in [Ref. 5]. CIM can be defined as the use of integrated computer systems to automate the manufacturing of an item starting with design and continuing through final production [Ref. 6]. Two portions of any future CIM computer system that are widely used today are Computer Aided Design (CAD) and Computer Aided Manufacturing (CAM).

With the introduction of the minicomputer and the increasing power of the microcomputer, significant gains have been made in designer productivity and accurracy [Ref. 7]. In addition, newer CAD systems even allow simulation and perfomance analysis. Similar gains are being made in CAM with the advances in sensor technology, computer controlled robot devices and numerically controlled machines. Examples of numerically controlled machines are the tube bending digitizers that automatically measure and record hydraulic tube shapes and then perform multiple tube bends during the construction of the Boeing 767 [Ref. 2]. What is currently missing is a link between CAD and CAM to allow data transfer from one to the other thus allowing a completely automated factory. In an ideal factory, a product would be designed using CAD and then the design would automatically be transformed into a final product with very little human intervention. To even approach such an ideal, a communications path must be established between CAD and CAM. Figure 1 shows how an Expert System Shell Translator would be used to fill this gap.

An example expert translator system was proposed and partially implemented in [Ref. 5] for the construction of a house. In this paper, we will enhance the translator to demonstrate its feasability using a more realistic product design. In addition, proposed data models for CAD output and CAM input as suggested in [Ref. 5] will be refined as necessary to facilitate the enhancement.

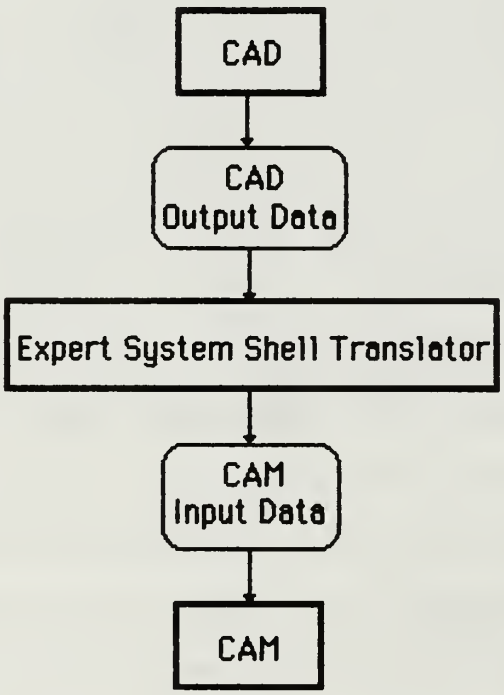


Figure 1. Relationship of Expert System Shell Translator interface to CAD and CAM.

There are at least two reasons for developing the technology to integrate current CAD and CAM systems rather than starting over with a fully integrated CIM system from scratch [Ref. 8]. First, low level manufacturing equipment will most likely be produced by a varied

assortment of manufacturers and will require different hardware and software for control. Second, factories that are interested in increasing automation will be more likely to add to existing equipment to reduce costs. They cannot afford to keep up with the latest in technology and abandon their previous investments. Each of these component systems can be expected to have a wide variety of data management systems. Therefore, an easily adaptive CIM system structure is required to handle these differences. The translator provides that adaptability by providing the interface necessary to link the various CAD and CAM systems already in use.

III. Data Output Specification for Computer Aided Design

Figure 2 shows the necessary processing and generation of data performed by CAD.

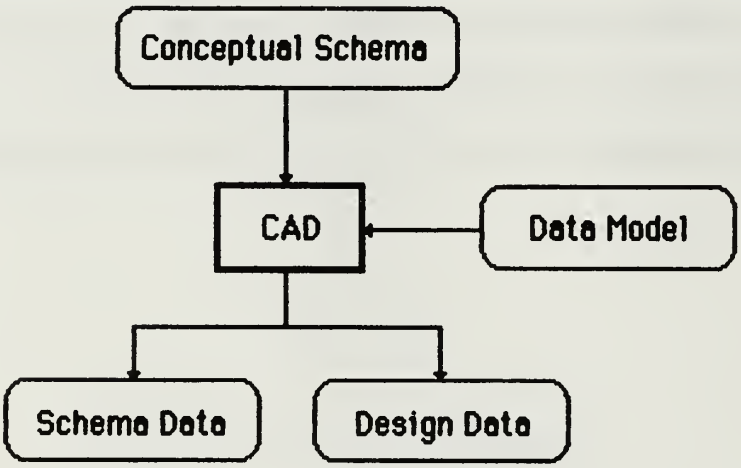


Figure 2. Processing of data during Computer Aided Design.

Using both the conceptual schema and the data model as input, CAD generates a design schema and the corresponding design data for the product to be manufactured. The conceptual schema provides the hierarchical part_of relationships while the data model acts as a guide for the CAD process.

A. CONCEPTUAL SCHEMA

Most of our discussion in this section will center on the proposed format for the design data but first lets examine a sample conceptual schema. Figure 3 is the conceptual schema for a generic house

construction application. The conceptual schema provides CAD with the hierarchical part_of relationships between primitive types from which composite objects can be built. Each block in the conceptual schema represents a different primitive.

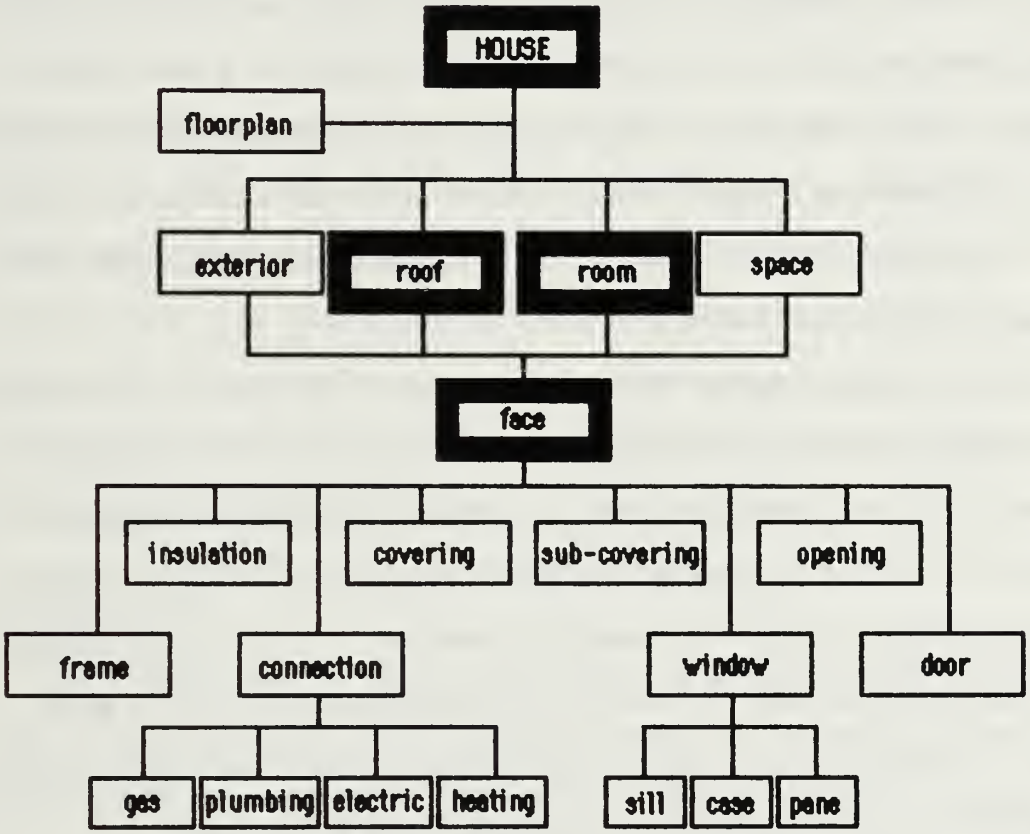


Figure 3. Conceptual Schema for a generic house.

Each part, real or abstract, of a final product house would correspond to an instantiation of some primitive. Instantiation is said to occur when a primitive is copied and that copy refers to an actual part of a design in progress. For example, the living room and bedroom of a house are

different instantiations of the same primitive **room** while their walls, floors and ceilings would correspond to the primitive **face**. There is actually a two-way relationship present. That is, if a particular face, **face_A**, is part of a room, **room_1**, then **room_1** contains **face_A**. This dual relationship will be used later in the development of the design data. It is also important to note that while a sub_cover must be part of some given face, it is not true that any instantiation of a face must contain a sub_cover. For example, consider a face consisting of unpainted brick attached to a wooden frame. Then the face would consist of a cover (the brick) and a frame. If painted, the brick becomes a sub_cover with the paint acting as a cover.

Primitive types may be defined to any level of abstraction and become the building blocks for the final product design; thus, each conceptual schema is product dependent [Ref. 5]. Those primitive types with dark borders in Figure 3 have named subtypes associated with them. A house may be subtype **colonial** or **ranch** for example. A room may have subtype **bedroom** or **bathroom**. The use of subtypes allows information about specific configurations of types to be stored for later use. They become a framework on which to build. For example, there are certain characteristics about a bathroom that makes it a bathroom. The subtype **bathroom** should capture that information which is true for all bathrooms for use in future designs.

The CAD process, guided by the data model, records actual instantiations of the primitive types of the conceptual schema to form the design schema for the product of interest. A particular house design

schema for which our example translator will be based is shown in Figure 4.

The design schema for a product uses inheritance properties to infer some information about the primitive types using known information about related types. Inheritance refers to those cases where there is no need to specify different values for two different types that are related in a specific manner. One type simply inherits the information from the other type. Consider a car being manufactured. If the car has its color specified, then parts of the car such as fenders, doors and hood would inherit that color information. In addition, using both **part_of** and **contains** relationships, this information is easily passed both up and down the hierarchical structure. The conceptual schema provides the CAD/CAM translator information on how the different building blocks of the final product will fit together.

B. PROTOTYPE

A prototype can be thought of as a block of memory allocated to store data for any given type (primitives and subtypes). For each different type, a new prototype must be defined since the amount of memory storage required is type dependent. In this way, we can partition our data such that all the facts known about any particular type instantiation can be aggregated in much the same way modern programming techniques allow the partitioning of programs into modules [Ref. 9]. There exists a one-to-one correspondence between the set of all types and the set of all classes or prototypes [Ref. 5].

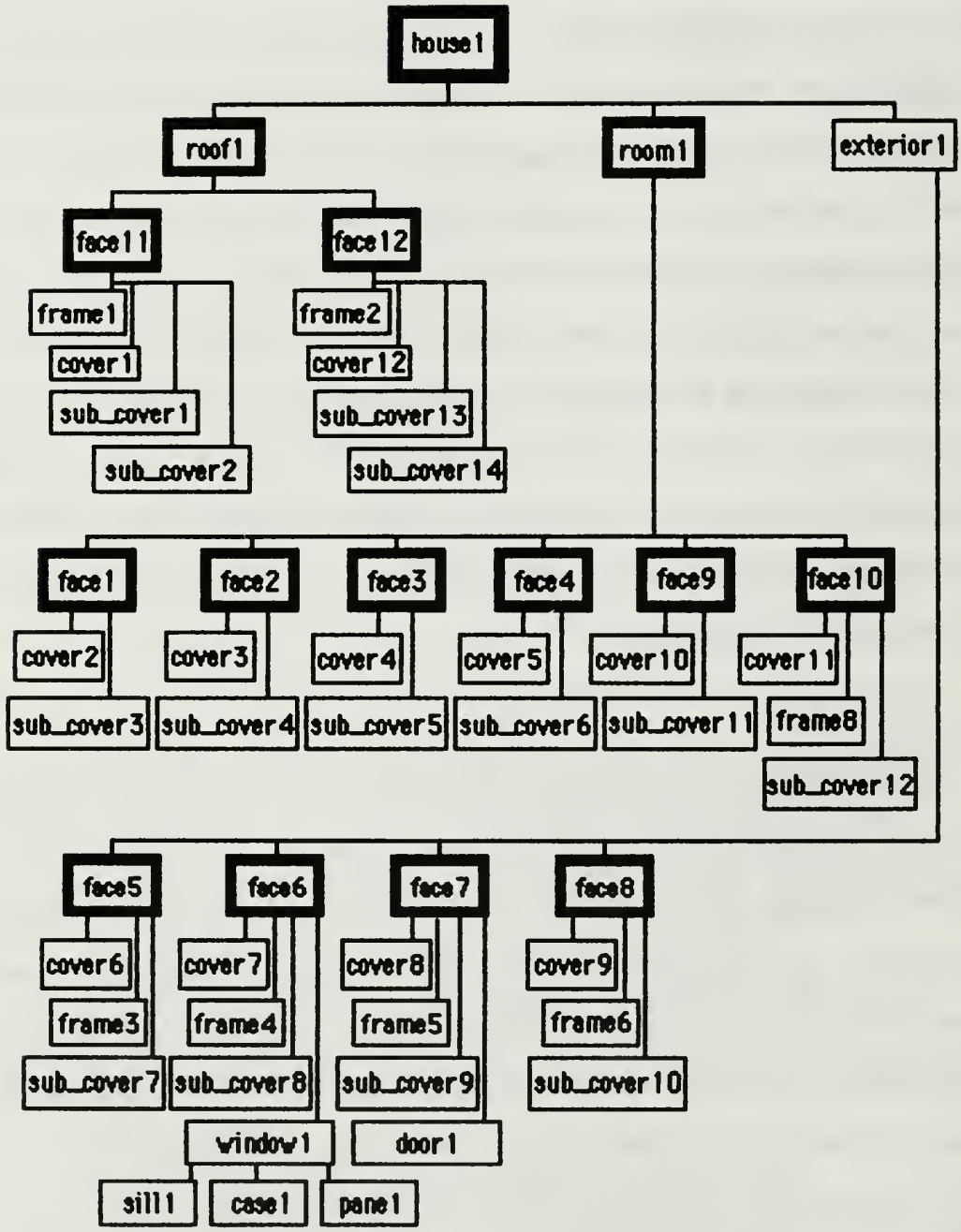


Figure 4. Design Schema for House1.

The set of all prototypes for any given conceptual schema will be designed to provide the interface between CAD and CAM with the necessary data to determine all required input for the CAM routines. Now consider the design of a prototype that will allow us to accomplish this task. Figure 5 is an example prototype for the primitive type **cover** listed in our conceptual schema presented earlier.

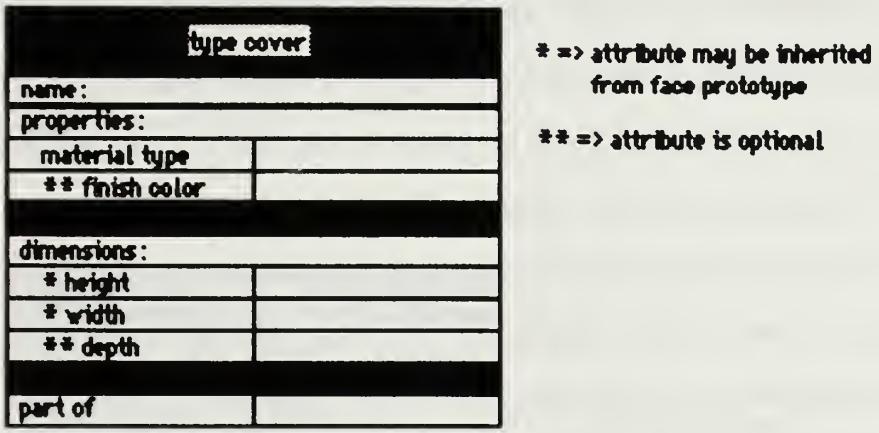


Figure 5. Prototype for primitive Cover.

Each prototype has named slots which can be filled in for a particular instantiation of that prototype. These slots contain either relation or attribute data. The slot **part of** in Figure 5 is for relation data. It relates the cover to a particular face. The slot **material type** is for attribute data and is used to store material information about the cover.
A blank prototype is also known as an intension or abstract specification. An intension is a meaning of a concept; that is, the prototype for a cover defines what it means to be a **cover** [Ref. 9]. With the appropriate slots filled in, it becomes an extension of the original prototype [Ref. 5]. An

extension is a concept which corresponds to an actual item which has existed in the past, currently exists or will exist in the future [Ref. 9]. In the above illustrated prototype, the slots **material type**, **height** and **depth** fall under different rules for being filled in as Figure 5 notes. The attribute **material type** is required to be filled in while **depth** is optional. Those slots made optional were those that could have a nonsensical value under some circumstances. Consider the depth of a cover of paint. While some number could be given, it would not normally be relevant to the design and construction of a house. Therefore, its value is optional. Careful thought must be given to the use of optional entries during the design of the prototypes. The design and efficiency of the CAD/CAM interface may be dependent on the number of vacant slots allowed. For those cases where the slots may or may not be filled in, the interface will have to consider both cases. The number of vacant slots allowed can have an adverse effect on the number of rules used, the complexity of rules used, or both for the interface system.

The attribute **height** shown in Figure 5 may be filled in or left blank with the value to be determined for input to the CAD/CAM interface using inheritance rules. Inheritance will be discussed in more depth in the next section.

The format for storing data in a prototype's slots should be kept simple to minimize the effect on the interface design. That is, value information for each slot should consist of only two parts at the most: actual value and units of measurement when required. The attribute/relation that specifies the slot may consist of several parts

itself such as **dimensions(height)** but should be standard. Using this method, each prototype slot is bound to a specific value based on some standard attribute or relation. For any given product, there will exist a set of attributes and relations for which each prototype will require a subset to define its required slots. Some attributes and relations will be universal to all products while others will be product specific. Consider, for example, **property(material type)**. This would be required knowledge for all products. Now consider the attribute **dimensions(height)** in relation to spherical product. It has no meaning while the attribute **dimensions(radius)** does.

type face	
name:	face 11
properties:	
* finish color	brown
dimensions:	
height	151.5 inches
width	384 inches
depth	6.5 inches
contains	[frame1 ,sub_cover2, sub_cover1 ,cover1]
normal_X	(0)
normal_Y	(0.34)
normal_Z	(0.94)
part of	roof1

* => attribute may be inherited from cover prototype

Figure 6. Prototype for primitive Face filled in for instantiation face11.

Figure 6 is an example of a filled in prototype showing extension name, qualifying data and values. This prototype is for a particular face shown

in the house design schema of Figure 4. It represents CAD's knowledge about **face11**. Note that one of the slots violates the rules presented above for simplicity of slot specification. This is the slot for the relationship **contains**. This type slot allows for a one-to-many relationship between a composite object and its components. This was done for two reasons. First, **part_of** already provides a simple relation between any two related parts that meets the requirements listed above. Second, the use of multi-values allows CAD to move quickly up and down the hierarchical design schema while conserving storage memory required for each prototype.

C. SLOT INHERITANCE

Now we consider in more detail how inheritance can be used to determine a slot's value. Inheritance refers to the property exhibited by two prototypes, which due to their relationship, possess slots which must take on the same value as the other's slot.

Face11, whose prototype is shown in Figure 6, contains **cover1**. Figure 7 is a filled in prototype for **cover1** based on known CAD design data. The primitives **height** and **width** for **cover_1** are not filled in and therefore their values must be determined using inheritance rules when determining the design data for input to the CAD/CAM interface. For this reason, the inheritance rules must be part of the prototype definition. Looking at Figure 7, the '*' indicates that the slot, if left blank, will inherit its value from the face which the cover is part of. Using this method, it is possible to specify an entire chain of primitives from which inheritance can take place. For example, the **cover1** prototype could have

specified that inheritance from the sub_cover beneath **cover1** would take priority over inheritance from **face11**. In addition, if inheritance from sub_covers was allowed, then ordering of the sub_covers becomes a factor. Using this method for a cover with two sub_covers beneath it, we should first look for the slot value in the sub_cover directly beneath the cover, next check the second sub_cover, and lastly get the value from the appropriate face if not yet successful in finding a value.

type cover	
name:	cover1
properties:	
material type	shingle12
** finish color	brown
dimensions:	
* height	
* width	
** depth	.25 inches
part of	face11

* => attribute may be inherited from face prototype

** => attribute is optional

Figure 7. Prototype for primitive Cover filled in for cover1.

Part of the CAD process is verifying that the correct values for those slots left blank will be properly inherited for input to the translator.

D. COORDINATE SYSTEM

In order to specify location information in a prototype, it is imperative that the frame of reference be known by any process using that information. For most products, three frames of reference should suffice.

These are global or world, product and local coordinates . Figure 8 shows the relationship between each of these systems.

Global or world coordinates relate to the real world. For example, the lines of the compass could be used with the Z axis perpendicular to the ground. The product coordinate system expresses location information relative to the product itself and is useful when locating parts on the product regardless of the absolute location of the parts relative to the earth. For large parts that are made up of many smaller parts, the local coordinate system may be used to express the location of the smaller parts relative to the large part.

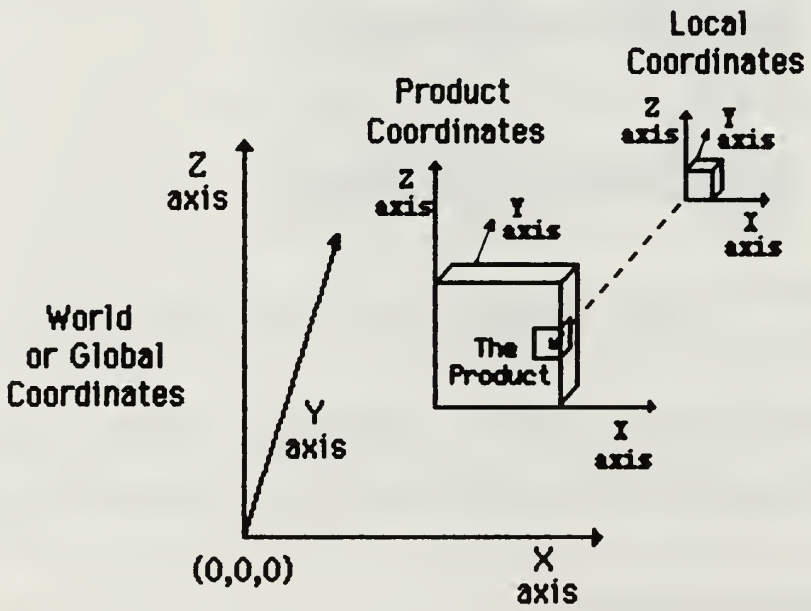


Figure 8. World, Product and Local coordinate system relationships.

The use of product and local coordinate systems not only eliminates the need for absolute or global location coordinate information under most

circumstances but also provides automatic update of location information during design changes. For example, consider a face, **face1**, that contains a window, **window1**. A design change is made that causes **face1** to be moved ten feet. If **window1**'s location has been expressed relative to **face1**, then updating **face1**'s location will automatically handle **window1** since the relative location for **window1** has not changed.

In addition to specifying locations of various parts, other information about the part's position may be desired. If many flat parts are being used in a product, then the unit vector perpendicular to their surface can be used to gather additional information on how the parts will align in the final product. This vector is called a **normal**. Figure 9 shows an example of a normal.

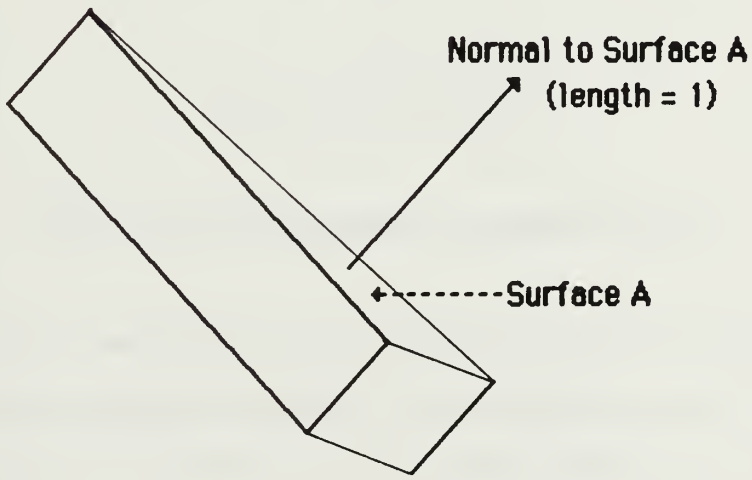


Figure 9. Example of a normal vector to a surface.

By definition, a normal is of length 1 unit, regardless of the units used. In this manner, each flat surface will have one unique normal associated

with it and each normal will specify one and only one surface. It is important to remember that the coordinate system used to express a normal will affect its value and therefore, for a normal to be useful, the coordinate system associated with that normal must be specified.

IV. Data Requirements for Computer Aided Manufacturing

Figure 9 shows the data flow for a typical CAM process. The Material Requirements Planning Data consists of two parts, assembly instructions and a raw material requirements listing [Ref. 5]. Each of these will be discussed in this section.

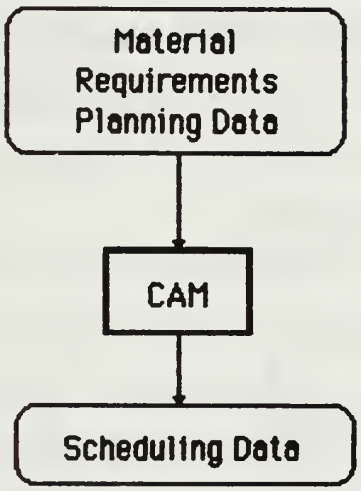


Figure 9. Computer Aided Manufacturing data flow [Ref. 5].

The purpose of the expert system translator will be to provide the assembly instructions and raw material requirements in such a format that will allow it to interface to an automated manufacturing system.

A. ASSEMBLY

What information is necessarily contained in the generated assembly instructions? That is product dependent. For example, when putting solder on an electrical connection, the type of solder used as well as the

temperature of application can be important. When gluing two components together, the type glue used, the pressure applied and the drying time become important. These are examples of information that may be generated by the CAM system normally but may also be overridden during CAD. The interface must be capable of passing this information through to the CAM process.

In addition, the assembly instructions will provide sequencing information, the determination of which may require not only the conceptual schema for relation data, but also prototype data. This would be due to information contained in the prototype that affects priority such as space requirements for installation. Figure 10 is an example where this type of information is necessary.

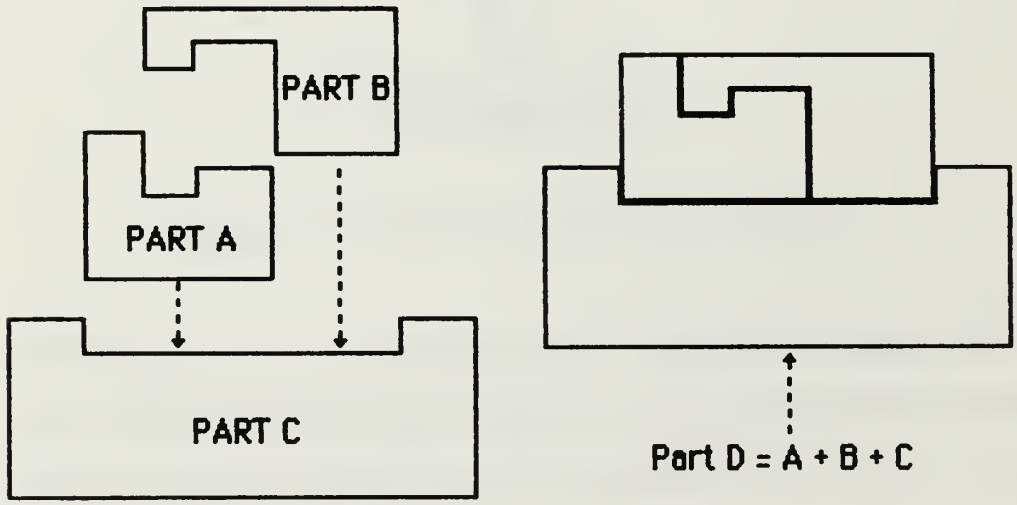


Figure 10. Assembly consisting of parts A, B, and C.

Part A, part B and part C assemble together to make part D. From the diagram it can be seen that an assembly sequence that attaches part B to

part C and then part A to the pair to form part D will not work since part B will interfere with part A unless sufficient slack exists in the fit to allow part A to slide in from the side. This type of information would not be contained in the conceptual schema.

For some products, component location information during manufacturing is important. Consider an electrical circuit again. Locating components on a circuit board is a very complex problem which is not currently well supported by CAM and ignored during CAD [Ref. 10]. Well placed components makes efficient routing of interconnections a simple task and therefore desirable.

B. MATERIALS

The other portion of the Material Requirements Planning Data is the raw material requirements listing. Figure 11 illustrates some example output for raw materials, again from the house construction example. The names of each item corresponds to well known building materials. For example, **hardboard32** and **hardboard34** are both hardboard material but have different dimensions and costs. To calculate requirements for **shingle12** and **brick88**, effective areas were used. Shingles are overlapped during construction decreasing their effective area covered while bricks have concrete placed between them increasing their effective area.

Based on dimensions of the product, cost per unit of raw materials, and design information about the raw materials of interest, the translator can generate the total amount of raw materials required and the total cost.

Raw Materials Report

Item	Cost	Units Required
door1	$16	1
window1	$30	1
concrete1	$1737	347.514
wood8	$3582	434.194
tar_paper2	$841	6.73333
hardboard32	$211	1.54776
hardboard34	$147	1.54722
hardboard78	$200	0.694444
hard_wood9	$900	75
sheath_paper24	$64	0.850277
shingle12	$2020	1616
brick88	$4224	3673.2
paint9	$8	1.0317
paint17	$4	0.551818
paint21	$12	0.923095

Total material cost is $13996

Figure 11. Raw materials requirements listing.

In addition, information on allowed substitute materials could be kept in the database to take advantage of not only fluctuating cost, but also fluctuating inventory in order to get the most optimal cost product [Ref. 5]. For such a system to work successfully though, it must allow for CAD to specify a no-substitute condition when required.

Using the Material Requirements Planning Data as input, CAM will generate the Scheduling Data necessary for final production. The Scheduling Data may then be used as input to an automated manufacturing system.

V. Expert System Shell Translator

The Expert System Shell Translator will act as the interface between CAD and CAM. Figure 12 demonstrates how the translator relates CAD data output to CAM required input.

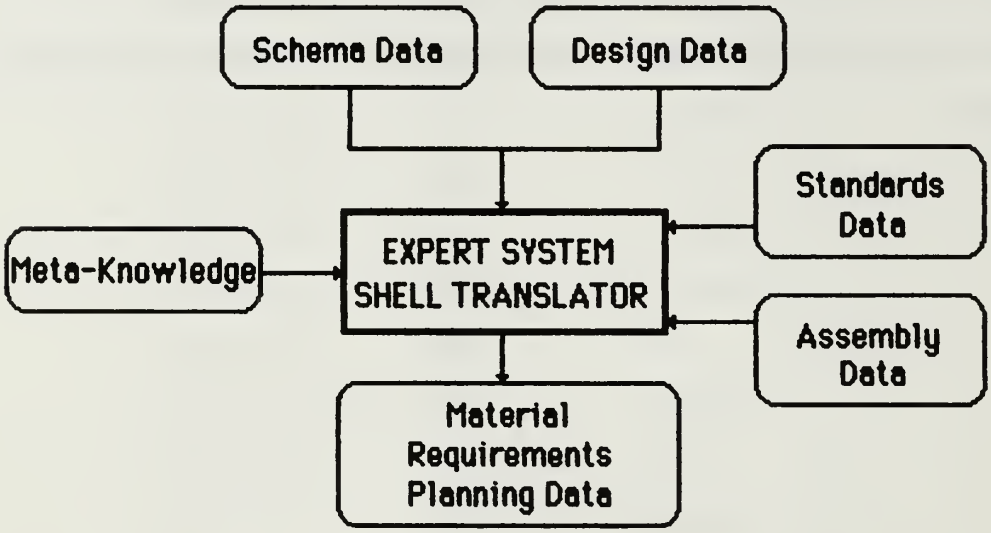


Figure 12. Expert System Shell Translator [Ref. 5].

Using the schema data and design data from CAD as input, the translator determines the necessary material requirements planning data to pass to CAM by making use of meta-knowledge and assembly data. A deduction-oriented rule-based system that starts with known facts and deduces new facts is said to exhibit forwards chaining [Ref. 11]. Forwards chaining has the ability to reach many conclusions based on the data which is what we desire for our translator. These new conclusions are then used to search for more facts. When no more conclusions are possible, forwards

chaining is complete. This is just the type structure that would be needed. In addition, the translator performs various standards checks on the CAD data to ensure its correctness. Correctness is used here to imply the data meets all known requirements. These requirements may be based on laws of physics, laws of government, or anything else deemed appropriate.

A. META-KNOWLEDGE

Meta-knowledge can be defined as knowledge about knowledge [Ref. 12]. It is used to guide the program in the selection of rules to apply. This knowledge may be either implicit or explicit. Explicit meta-knowledge is sometimes employed using meta-rules. Meta-rules are used to guide in the application of other rules. An example would be the case where two possible rules could be applied, rule one and rule two. A meta-rule might state that rule two should be tried prior to rule one for some specific conditions. Implicit meta-knowledge is more common but more difficult to handle when changes are made to the program. Figure 13 shows an example of implicit meta-knowledge that could be used in house design and manufacture.

Explicit is the fact that rule number one passes control to rule number three and rule number three uses recursion to find and handle all faces except those facing upwards. Implicit is the fact that when rule three eventually fails, we backtrack to rule number two.

B. ASSEMBLY DATA

The assembly data must provide sequencing information which makes use of not only the conceptual schema for relational data, but also

prototype data. The conceptual schema gives information on how the components fit together, but the prototype data may give new insight to priorities for those cases where assembly may take several paths. Consider soldering two electrical components, a transistor and a resistor, to a circuit board. The transistor is much more heat sensitive than a resistor normally. Therefore, even though the schema would show no priority for ordering, data contained in the prototype for the transistor would show its heat limitations during soldering and would be used to ensure the resistor was placed on the circuit board first.

```
assemble(L,face) :-
  assemble1(L,[],face).

assemble(L,face) :-
  member(Face,L),
  normal_Z(Face,1),
  assertz(operation(comment,'build floor as last step',_,_)),
  contains(Face,L1),
  assemble2([L1],[L1],face).

assemble1(L,L1,face) :-
  member(Face,L),
  not(normal_Z(Face,1)),
  delete(Face,L,L2),
  contains(Face,L3),
  assemble1(L2,[L3|L1],face),!.
```

Figure 13. Use of Meta-Knowledge.

It is important to remember that the term components is being used here in an abstract way. Items such as solder, grease, oil and glue may all be considered components of a product.

Figure 14 is a sample of two assembly rules written in Prolog and used for the example house construction project. Note that both rules make use

of schema data as well as design data to provide sequencing. The schema data allows access to the **frame** attached to the **face** currently under construction while design data, in this case **normal** information, is used to prioritize the frames.

```
/* do foundation frame */
assemble(H,house) :-
  is_a(Yface,face),
  trans_partof(Yface,H),
  normal_Z(Yface,1),
  contains(Yface,L),
  member(Frame,L),
  is_a(Frame,frame),
  property(Frame,material_type,Mtype),
  assertz(operation(Frame,assemble,'material type: ',Mtype)).

/* do frame perpendicular to ground */
assemble(H,house) :-
  is_a(Yface,face),
  trans_partof(Yface,H),
  normal_Y(Yface,0),
  normal_Z(Yface,0),
  contains(Yface,L),
  member(Frame,L),
  is_a(Frame,frame),
  property(Frame,material_type,Mtype),
  assertz(operation(Frame,assemble,'material type: ',Mtype)).
```

Figure 14. Assembly rules for house construction.

In addition to sequencing instructions, assembly data can be used to determine component location. This is a very difficult problem in VLSI design and manufacture and is not well supported. The number of possible placements in many circuits makes an exhaustive search for the ideal placement impossible. In addition, there are many factors controlling placement including temperature limitations, position of edge connectors, position of busses, and speed restrictions which in turn restricts length

of interconnections [Ref. 10]. To find a recommended solution to such a problem, an expert system translator could make use of advanced search techniques currently employed in artificial intelligence programming such as depth-first, best-first or breadth-first. Because this is such a difficult problem, a system might allow a user to interact with it to aid in the search.

C. STANDARDS DATA

Standards data is used here in a broad sense and represents not only governmental standards required by Federal, State, local and Occupational Safety and Health (OSHA) regulations but also those standards, which if violated, will result in a product that may not function properly. Governmental requirements have always been with us and include federal, state and local regulations. But with the advent of CAD systems, new product designs have become increasingly complex. For instance, VLSI and multichip systems may be comprised of more than 250,000 gates [Ref. 13]. Due to this increased complexity, it has become increasingly harder to detect design errors prior to manufacture and shipment to customers. Figure 15 shows data put together at IBM which relates circuit errors remaining to rate of errors detected.

It is important to note that there exists a point in time when there is a marked decrease in design error detection even though the number of errors left remains relatively high. Design requirements can be included in the standards data to reduce the number of errors.

For example, in VLSI design, the translator could check for loading of components, power supplies to components and all leads properly

connected. An example of this would be verifying that the fanout of each integrated circuit had not been violated. The fanout of an integrated circuit is the number of gates that may be connected up to one pin of that integrated circuit without overloading the circuit and causing failure. This type of error could produce a product that functioned properly but had a shortened lifespan. It might only be found after numerous customer complaints and by that time the supply stock of the device could be extremely costly to replace.

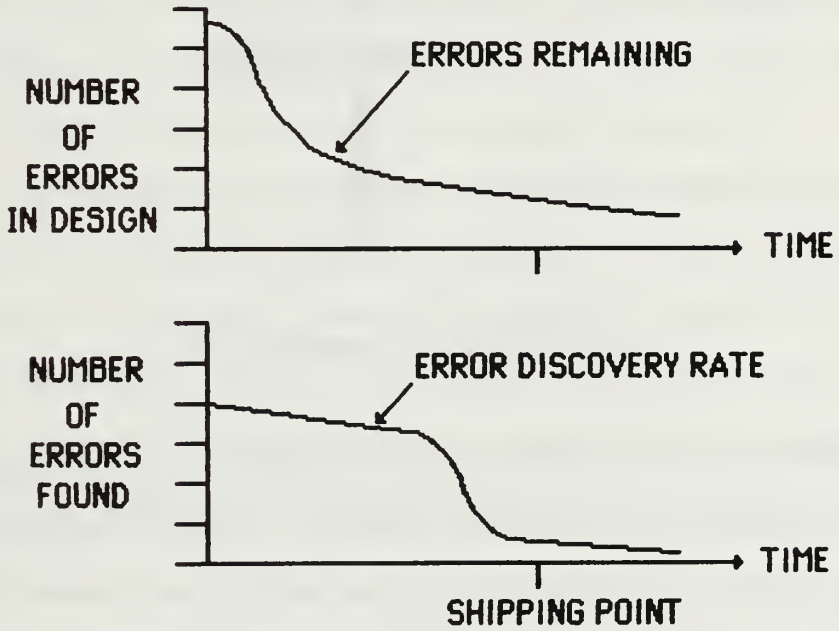


Figure 15. Error detection in VLSI circuit design [Ref. 7].

D. LANGUAGE OF CHOICE

The terminology **expert system** has been used in our discussion on the CAD to CAM translator. True expert systems are written using artificial intelligence languages such as Prolog and belong to the class of artificial

intelligence applications known as **knowledge-based systems** [Ref. 5]. Figure 16 is an example of the structure of an expert system.

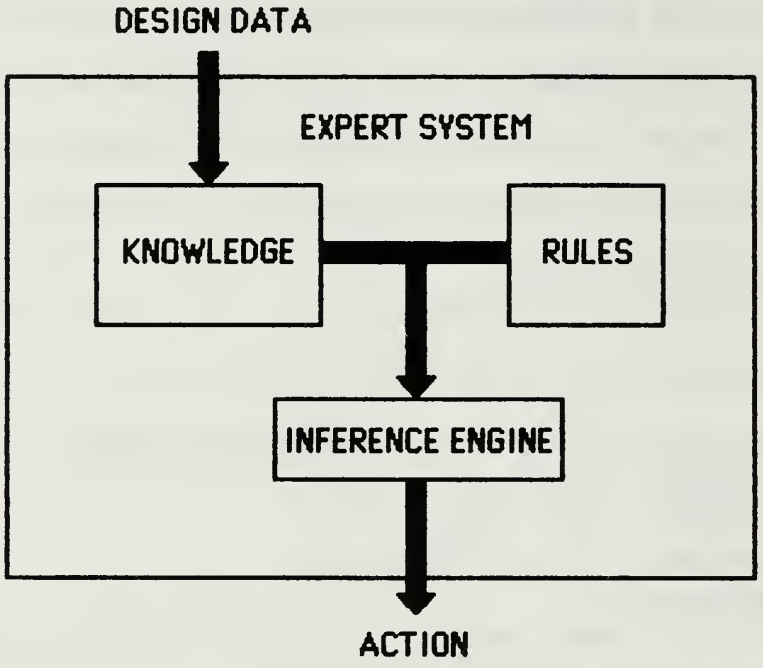


Figure 16. An Expert System [Ref. 13]

Prolog programs are actually rule-based where each rule represents an expert's knowledge of the problem. "Knowing" is reduced to being able to represent symbolically facts about the surrounding environment [Ref. 12]. Data supplied to the expert system is then treated as facts and used to deduce more facts. An interesting feature of some of these rules is their apparent link to rules of thumb; these rules are known as heuristics [Ref. 12].

What are the necessary qualities of an expert system? It must of course properly perform its assigned tasks. The problem is that what is

proper to one expert may not be proper to another. For example, consider two builders each constructing a house of similar design. While most of the assembly priorities for each one would be similar, some differences could exist. If both builders are experts in their field, who is more correct? The advantage of using artificial intelligence languages is the flexibility to allow manufacturers to set their own priorities by modifying the expert system rule base without any change to the CAD or CAM systems in use.

Expert systems also have the ability to explain their path of reasoning, although in today's systems the explanation is usually nothing more than a trace of rules sucessfully being fired. A rule is said to fire if enough facts are known so that the rule is now proven to be true.

When actually constructing an expert system, it is also important that the code be partitioned according to the areas of concern. This is due to the fact that many experts, who would be expected to supply their knowledge to build the system, are limited in the breadth of their knowledge. Therefore, the code should be divided in such a way that each expert has responsibility for only that part pertaining to his area of expertise.

VI. Example Expert Shell Translator

Appendix A is a diagram of a simple one room house which was used as a basis for this example expert shell translator. The prototypes for the house are based on the conceptual schema of Figure 3 and are shown in Appendix B. The actual prolog computer code for the translator is contained in Appendix C. Appendix D gives the translator output for the house construction example whose design is depicted in Appendix A.

```
part_of(house,room).

part_of(room,face).

part_of(face,door).
part_of(face,window).
part_of(face,opening).
part_of(face,covering).
part_of(face,sub_covering).
part_of(face,frame).
part_of(face,insulation).

trans_partof(X,Y) :- part_of(X,Y),!.
trans_partof(X,Y) :- part_of(X,Z),
      trans_partof(Z,Y),!.
```

Figure 17. Implementation of Conceptual Schema.

A. CAD DESIGN DATA

Two of the computer code files in Appendix C are used to store data that would be expected as input to the translator from CAD. The **schema** file contains the schema data for the product of interest, in this case a one room house. Figure 17 shows some of the code from this file.

The last two rules in Figure 17 are used to specify transitivity in prolog. Thus, if piece_A is part of piece_B, and piece_B is part of piece_C, these rules would imply that piece_A must also be part of piece_C. The use of such rules precludes the necessity to explicitly declare these relationships for all possible cases.

The design schema and other design data, which are derived from the filled in applicable prototype slots, are contained in the **house1** file. Figure 18 demonstrates the relationship between one sample prototype and its corresponding data in the **house1** file.

It can be seen that the use of a language like prolog makes it easy to convert the prototype data to usable data for the translator.

B. STANDARDS CHECKS

The first series of operations performed on the input data by the translator are those necessary to verify all applicable standards requirements are met. Figure 19 contains some of the standards from the **standards** file in Appendix C that were used for our one room house.

Note that while the width and height standards for doors apply only to a door of type door1, the depth standard for doors and the window pane quality standard apply to all doors and windows respectively. This demonstrates the flexibility of the language and our system.

In addition to actual physical checks, two other types of standards data are also contained in the **standards** file. These are shown in Figure 20.

The first is the **comment_for** data. For example, consider the rule **comment_for(frame,wood,framing)**; this rule relates any frame made out of a wood product to the comment **framing**. This allows data that can

HOUSE1 ➡

```
/* face1 */

is_a(face1,face).

dimension(face1,height,115,inches).
dimension(face1,width,362,inches).
dimension(face1,depth,1,inches).

contains(face1,[sub_cover3,cover21).
normal_X(face1,0).
normal_Y(face1,-1).
normal_Z(face1,0).
part_of(face1,room1).
```

PROTOTYPE ➡

type face	
name:	face1
properties:	
* finish color	
dimensions:	
height	115 inches
width	362 inches
depth	1 inch
contains	[sub_cover3,cover2]
normal_X	(0)
normal_Y	(-1)
normal_Z	(0)
part of	room1

* => attribute may be inherited from the cover prototype

Figure 18. Computer code derived from Prototype data.

```
minimum(door,door1,width,32,inches).
minimum(door,door1,height,6,feet).
maximum(door,door1,width,4,feet).
maximum(door,door1,height,7,feet).
minimum(door,_,depth,2,inches).
maximum(door,_,depth,3,inches).

minimum(pane,_,quality,3).
```

Figure 19. Standards checks.

```
comment(masonry,'approved methods must be used for building
  masonry walls when outside air temperature drops below 40
  degrees farenheit').
comment_for(cover,brick,masonry).
comment_for(cover,concrete_block,masonry).
comment_for(sub_cover,brick,masonry).
comment_for(sub_cover,concrete_block,masonry).

comment(framing,'grade marks must be clearly visible on all
  framing members for inspection').
comment_for(frame,wood,framing).

check_for(sub_cover,tar_paper,[tar_paper1,tar_paper2,tar_paper3]).
```

Figure 20. Comments on standards requirements.

only be verified during manufacturing to be output by the translator for the information of the CAM system. Sample output from Appendix D is shown in Figure 21.

```
check for frame frame3

    grade marks must be clearly visible on all framing
    members for inspection

check for sub_cover sub_cover7

check for cover cover6

    approved methods must be used for building masonry walls
    when outside air temperature drops below 40 degrees farenheit
```

Figure 21. Translator output following application of standards comments.

The last data type contained in the standards file is the **check_for**. In Figure 20, the rule **check_for(sub_cover, tar_paper, [tar_paper1, tar_paper2, tar_paper3])** is used to verify that all sub_covers made out of tar_paper use tar_paper1, tar_paper2 or tar_paper3. In addition, those types of tar_paper not used are listed as possible material substitutions. These lists of possible substitutions will become important again when determining raw material requirements later in this chapter. Figure 22 is example output data showing the use of this type standards check.

Note that the design data currently has sub_cover14 made from tar_paper2. Therefore, the other two types are listed as possible substitutes. For more realistic situations, substitutions of one material may affect other parts of the product. For example, consider a case where

several types of plastic have been listed as acceptable for the product piece in question. However, if a glue is being used on the plastic during the manufacturing process, different plastics may require different glues. Therefore, caution must be used in making substitutions.

```
check for sub_cover sub_cover14
    sub_cover sub_cover14 meets requirements; allowed substitutes are:
    - tar_paper1
    - tar_paper3
```

Figure 22. Listing of substitutions by translator during standards checks.

C. PRODUCT ASSEMBLY

Once the standards checks have been completed, the translator must determine the product assembly sequence. To build our one room house, we would expect the frame to be erected first. Figure 23 is a listing of prolog rules used to generate the assembly steps for the frame foundation and walls.

The first frame selected for assembly is the foundation. This frame is located by finding a face which is part of the house being built and which also faces away from the ground. The **trans_partof(Yface,H)** will locate any face that is part of the house represented by the variable **H**. Then **normal_Z(Yface,1)** checks if the Z component of the normal to the face of interest is equal to one. If so, then this face is a floor. Figure 24 shows example orientations of normals for our one room house.

```
/* do foundation frame */
assemble(H,house) :-
  is_a(Yface,face),
  trans_partof(Yface,H),
  normal_Z(Yface,1),
  contains(Yface,L),
  member(Frame,L),
  is_a(Frame,frame),
  property(Frame,material_type,Mtype),
  assertz(operation(Frame,assemble,'material type: ',Mtype)).

/* do frame perpendicular to ground */
assemble(H,house) :-
  is_a(Yface,face),
  trans_partof(Yface,H),
  normal_Y(Yface,0),
  normal_Z(Yface,0),
  contains(Yface,L),
  member(Frame,L),
  is_a(Frame,frame),
  property(Frame,material_type,Mtype),
  assertz(operation(Frame,assemble,'material type: ',Mtype)).

assemble(H,house) :-
  is_a(Yface,face),
  trans_partof(Yface,H),
  normal_X(Yface,0),
  normal_Z(Yface,0),
  contains(Yface,L),
  member(Frame,L),
  is_a(Frame,frame),
  property(Frame,material_type,Mtype),
  assertz(operation(Frame,assemble,'material type: ',Mtype)).
```

Figure 23. House sample assembly rules for translator.

Any normal parallel to a coordinate axis will have that axis' component equal to one in value if it points in the positive direction along the axis and equal to minus one if it points in the negative direction. For the example house, only the normals to the faces contained in the roof do not meet these requirements. It is not necessary that any face meet this requirement; it has been done only to simplify the example.

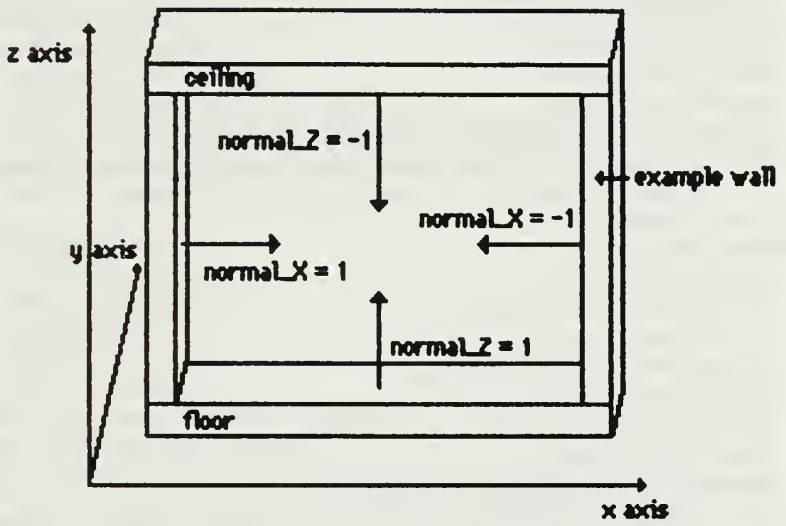


Figure 24. Use of normal vectors for house construction.

Once the floor frame is in place, the second and third rules in Figure 23 locate the wall frames and add them to the assembly list. The second rule looks for faces with normals parallel to the X axis by specifying that the Y and Z components of the normal are equal to zero. Similarly, the third rule locates those faces parallel to the Y axis. In prolog, backtracking will force these rules to be tried until no more valid solutions are found. In this way, we locate all faces meeting the

specifications of each rule. Therefore, we only need be sure that each rule does indeed fully state all specifications of concern.

In Figure 25, the rules which generate assembly data for the ceiling and roof are shown.

```
/* ceiling frame */
assemble(H,house) :-
  is_a(Yface,face),
  trans_partof(Yface,H),
  normal_Z(Yface,-1),
  contains(Yface,L),
  member(Frame,L),
  is_a(Frame,frame),
  property(Frame,material_type,Mtype),
  assertz(operation(Frame,assemble,'material type: ',Mtype)).

/* roof frame */
assemble(H,house) :-
  is_a(Roof,roof),
  trans_partof(Roof,H),
  is_a(Yface,face),
  trans_partof(Yface,Roof),
  contains(Yface,L),
  member(Frame,L),
  is_a(Frame,frame),
  property(Frame,material_type,Mtype),
  assertz(operation(Frame,assemble,'material type: ',Mtype)).
```

Figure 25. Assembly rules for roof and ceiling.

The only notable difference from our previous rules in Figure 23 is that faces associated with the roof are located by using the contains relation associated with the roof. This is a better method than using normals since the normal vector for a roof face can vary so much depending on the design of the house. The only framing now left to be performed is for the windows and doors. Figure 26 lists the rules that handle these two cases.

Both rules again check only parts of the house of concern. For the door, we determine its material and the two faces to which it is attached and save this information in assembly data. The same is done for the window except now the sill is treated as its frame. Again, both rules allow backtracking to get all occurrences of windows and doors as will all the assembly rules.

```
assemble(H,house) :-
  is_a(Door,door),
  trans_partof(Door,H),
  property(Door,material_type,Mtype),
  assertz(operation(Door,assemble,'material type: ',Mtype)),
  get_faces(Door,Face1,Face2),
  assertz(operation('','- attach to: ',Face1,Face2)).

assemble(H,house) :-
  is_a(W,window),
  trans_partof(W,H),
  contains(W,L),
  member(Sill,L),
  is_a(Sill,sill),
  assertz(operation(Sill,assemble,'window sill for: ',W)),
  get_faces(W,Face1,Face2),
  assertz(operation('','- attach to: ',Face1,Face2)).
```

Figure 26. Assembly rules for windows and doors.

With all the framing in place, the faces must now be constructed. Figure 27 gives the code to handle this. Note that first the exterior and roof are performed, and then the interior room itself. For each area, the **contains** relation is used to get a list of all parts, including faces, of each and the information is passed to an **assemble(L,face)** routine to erect only the faces. This is actually a series of routines that use both backtracking and recursion to determine the assembly data. Figure 28 gives the routines that start the process.

```
assemble(H,house) :-
   is_a(E,exterior),
   trans_partof(E,H),
   contains(E,L),
   assemble(L,face).

assemble(H,house) :-
   is_a(R,roof),
   trans_partof(R,H),
   contains(R,L),
   assemble(L,face).
assemble(H,house) :-
   is_a(R,room),
   trans_partof(R,H),
   contains(R,L),
   assemble(L,face).
```

Figure 27. Assembly rules for house faces.

```
assemble(L,face) :-
  assemble1(L,[],face).

assemble(L,face) :-
  member(Face,L),
  normal_Z(Face,1),
  assertz(operation(comment,'build floor as last step',_,_)),
  contains(Face,L1),
  assemble2([L1],[L1],face).

assemble1(L,L1,face) :-
  member(Face,L),
  not(normal_Z(Face,1)),
  delete(Face,L,L2),
  contains(Face,L3),
  assemble1(L2,[L3|L1],face),!.

assemble1(L,L1,face) :-
  assemble2(L1,L1,face),!.
```

Figure 28. Assembly rules to prioritize and obtain face data.

The first and second rules in the list handle two different cases, non-floors and floors respectively. Any face pointing directly upward is considered a floor, of which, for our simple example, there is only one. The first rule takes precedence over the second rule and calls the third rule in Figure 28. The third rule simply finds all faces which are part of the area of concern but are not facing upward. Looking at the left side of the third rule, **assemble1(L,L1,face)**, **L** is the set of parts determined using the **contains** relationship earlier and **L1** is a set which we will construct. **L1** is initialized to empty when the first rule calls the third rule. When the third rule finds a face meeting its requirements, the **contains** relation is again used to determine the parts of the face. This set of parts is added to **L1** and **assemble1** recursively calls itself looking for more faces. When none are found, we fall through to the fourth rule which calls **assemble2**. The **!** symbol at the end of the **assemble1** rules is there to prevent backtracking into them. We may only proceed forward into these rules. Backtracking is not necessary since we exit these rules only when all faces meeting our specifications are found.

Looking again at the second rule in Figure 28, we put only one face in the list at a time and backtracking is necessary in the case where there is more than one possible floor face. This may or may not be desirable depending on the house design. For the other faces though, a list of all faces in the area of concern is created using recursion to allow a search for common building materials to better organize the assembly data.

Figure 29 shows the rest of the routines necessary to complete the face assemblies. Note that **assemble2** will recursively call itself until

```
assemble2(Full_⊥,L,face) :-
  member(Face,L),
  delete(Face,L,L1),
  member(Item,Face),
  is_a(Item,sub_cover),
  property(Item,material_type,Mtype),
  operation(Y,_,_,Mtype),
  member(Face1,Full_⊥),
  member(Y,Face1),
  assertz(operation(Item,assemble,'material type: ',Mtype)),
  delete(Item,Face,Face2),
  assemble2(Full_⊥,[Face2|L1],face),!.

assemble2(Full_⊥,L,face) :-
  member(Face,L),
  delete(Face,L,L1),
  member(Item,Face),
  is_a(Item,sub_cover),
  property(Item,material_type,Mtype),
  assertz(operation(Item,assemble,'material type: ',Mtype)),
  delete(Item,Face,Face1),
  assemble2(Full_⊥.[Face1|L1].face).!.

assemble2(Full_⊥,L,face) :-
  member(Face,L),
  delete(Face,L,L1),
  member(Item,Face),
  is_a(Item,cover),
  property(Item,material_type,Mtype),
  not(liquid(Mtype,paint,_,_,_,_,_)),
  operation(Y,_,_,Mtype),
  member(Face1,Full_⊥),
  member(Y,Face1),
  assertz(operation(Item,assemble,'material type: ',Mtype)),
  delete(Item,Face,Face2),
  assemble2(Full_⊥,[Face2|L1],face),!.

assemble2(Full_⊥,L,face) :-
  member(Face,L),
  delete(Face,L,L1),
  member(Item,Face),
  is_a(Item,cover),
  property(Item,material_type,Mtype),
  not(liquid(Mtype,paint,_,_,_,_,_)),
  assertz(operation(Item,assemble,'material type: ',Mtype)),
  delete(Item,Face,Face1),
  assemble2(Full_⊥,[Face1|L1],face),!.

assemble2(Full_⊥,L,face).
```

Figure 29. Assembly rules to get list of covers and sub_covers.

there are no face parts left. It then falls through to the last rule which succeeds and thus exits. Again, no backtracking is allowed or necessary.

The first two rules in Figure 29 search for all sub_covers letting those sub_covers made up of material already used in the area of concern take priority over material not yet used. This is accomplished by searching through all the current **operation** predicates looking at all sub_covers already asserted that used the same material. If a sub_cover is found, then a search is performed over the list of all sub_covers in the area of concern to attempt a match. If a match is found, then that material has already been used and it will take priority. If no match is found, then the next sub_cover in the next face is listed in the assembly data.

The third and fourth rules provide a similar function for the covers except that covers made from paint are not yet allowed to be listed. The painting will be performed at the end of the house construction to prevent damage to the finish.

The house is now close to completion. The window panes are inserted into place, the windows and doors are painted, and the doors are installed using the appropriate doorknobs and hinges. Now is the time to complete the painting of the faces that was previously skipped over. Figure 30 shows the rules that handle this.

Note that first the roof is painted, then the exterior and then last we paint any rooms. The **paint_face** routines are similar to what we have already seen. First, the ceiling, if one exists in the area currently being taken care of, is painted. Then the walls are painted and next the floor is

painted. Any face left over is painted at the end. This handles slanted surfaces such as the faces of the roof. The one room house is now fully constructed.

```
assemble(H,house) :-
   is_a(R,roof),
   trans_partof(R,H),
   contains(R,L),
   paint_face(L).

assemble(H,house) :-
   is_a(E,exterior),
   trans_partof(E,H),
   contains(E,L),
   paint_face(L).

assemble(H,house) :-
   is_a(R,room),
   trans_partof(R,H),
   contains(R,L),
   paint_face(L).
```

Figure 30. Assembly rules to generate paint data.

D. RAW MATERIALS LISTING

With the assembly data finished, the translator must now determine the raw material requirements to build the house. It does this by calling on the **raw_materials_needed** rules. All the rules work in much the same manner. They first determine what extension is being considered, then the material of concern associated with this extension, and last the dimensions of this extension or area. All dimensions and areas are converted to a common unit of measurement prior to calculations.

Those parts of the house associated with a **face** extension such as **cover** and **sub_cover** call a routine **get_area** to determine the surface

area involved. This special routine is necessary since faces may have areas such as doors, windows and openings which subtract from the total area of the **face** to be covered. This is handled by calculating a negative area for each **face** to be subtracted out prior to material requirements calculations. This negative area is then asserted as a fact for each **face** prior to actual entry into the **raw_materials_needed** routines. An example calculation routine is shown in Figure 31.

```
raw_materials_needed :-
  is_a(Extens,sub_cover),
  dimension(Extens,depth,Th,Thunits),
  property(Extens,material_type,Material),
  material(Material,_,Ht,Htunits,Wd,Wdunits,Dp,Dpunits,
    _,_,_,_,Cost),
  match(Ht,Htunits,Wd,Wdunits,Dp,Dpunits,Th,Thunits,Act_Ht,
    Units1,Act_Wd,Units2),
  get_area(Extens,Area,Units),
  convert(Act_Ht,Units1,Act_Ht2,Units),
  convert(Act_Wd,Units2,Act_Wd2,Units),
  Num_Units is (Area / (Act_Ht2 * Act_Wd2)),
  Tot_Cost is (Num_Units * Cost),
  add_material(Material,Num_Units,Tot_Cost),fail.
```

Figure 31. Example material calculations for a sub_cover.

One aspect of how the above example works not yet mentioned is the call to **match**. This rule attempts to find a match between the dimensions of the material to be used and the thickness of the **sub_cover** within a tolerance band. This is then used to determine the orientation of the material within the **sub_cover**. For example, if a board, measuring two inches by four inches by four feet, is used to build a **sub_cover** which is four inches thick, then the two inch dimension would be used for area calculations. This type of check is necessary since the dimensions height,

width and depth are based on the view of the person determining the values.

Once the units of material required and cost are determined, these values are added to the total by calling **add_material**. This rule first checks for any previous data on this material. If some is found, then a new total is calculated and saved. Otherwise, a new fact on the material of concern is created and saved.

The only other unusual calculation performed during the material calculations is the one to determine the frame requirements along the center of the roof, between the roof and the ceiling. We need the height of the roof above the ceiling to make this calculation. This is easy to do though since the normal vectors for the roof faces are known. It turns out that each component of the normal is equal to the cosine of the angle created by the intersection of a line parallel to that component's axis and the plane containing the other two axis [Ref. 14]. Figure 32 demonstrates this concept. In Figure 32, the Z component of the normal vector is equal to the cosine of the angle created by the intersection of the normal and the plane containing the X and Y axis. With this fact, we can calculate the angle of intersection, **Beta**, of the roof and the house. Using the dimensions of the roof faces, it is now possible to determine the height of the roof above the ceiling since **sin(Beta)** is equal to the height of the roof above the ceiling divided by the length of the roof face.

Once the amount of materials required and their costs have been determined, a Raw Materials Report is output. The report lists units

required for each material item and that item's cost. Following the list of individual items is a total cost. Figure 11 gave a listing of this output.

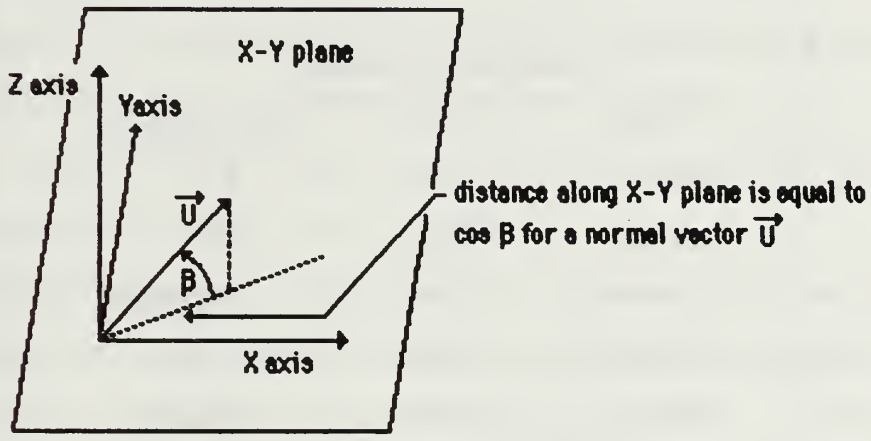


Figure 32. Distance of a normal vector along a plane.

After the initial Raw Materials Report, the example translator examines possible material substitutions reported during the standards checks and makes each substitution, one at a time, to generate a new report. Figure 33 is an example of a modified Raw Materials Report output by the translator. It shows the cost for parts when **sub_cover14** is made out of **tar_paper1** in place of **tar_paper2**.

```
**********************************************************
*                                                        *
 sub_cover14: substitute tar_paper1 for tar_paper2
*                                                        *
**********************************************************
```

Raw Materials Report

Item	Cost	Units Required
door1	$16	1
window1	$30	1
concrete1	$1737	347.514
tar_paper2	$420	3.36666
wood8	$3582	434.194
tar_paper1	$504	3.36666
hardboard32	$211	1.54776
hardboard34	$147	1.54722
hardboard78	$200	0.694444
hard_wood9	$900	75
sheath_paper24	$64	0.850277
shingle12	$2020	1616
brick88	$4224	3673.2
paint9	$8	1.0317
paint17	$4	0.551818
paint21	$12	0.923095

```
**********************************
*                                *
 Total material cost is $14079
*                                *
**********************************
```

Figure 33. Raw Materials Report with substitution.

IV. Conclusions and Recommendations

A. CONCLUSIONS

The goals of this research were to determine the design requirements for a generic CAD to CAM translator, design and implement a CAD to CAM translator for a particular product and in the process determine data requirements for CAD output and CAM input.

A conceptual schema is a useful tool with which to model the product to be constructed. A simple hierarchical structure for the conceptual schema results in a design schema in which the translator can easily move from part to part, whether the part is abstract or real. Prototypes provide an ideal abstract model of the product design data to be used as input to the translator.

Artificial intelligence (AI) oriented languages such as Prolog can readily use prototype structured data, even using slot inheritance to fill in unspecified values. In addition, many search methods have been designed and implemented using AI methods and they can provide powerful solutions to difficult problems such as the positioning of integrated circuits on an electronic circuit board. Their two drawbacks, when compared to other languages currently in use today are speed and availability. This is currently being remedied with the recent releases of affordable compiled versions of artificial intelligence languages designed for micro-computers.

The translator is capable of performing certain standards checks on the design data, passing assembly information through to CAM and also

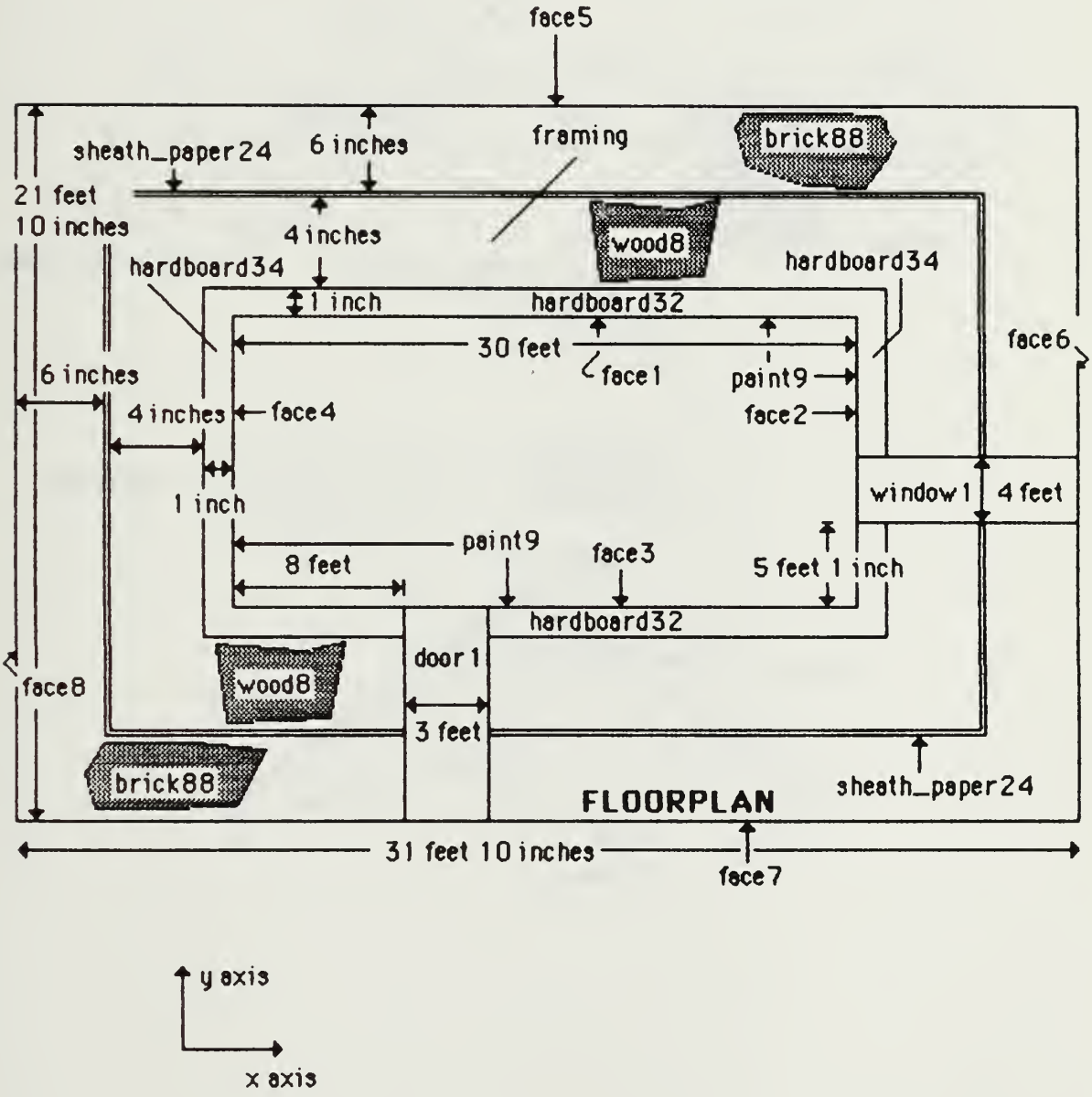
material requirements. The translator can also pass information that is useful to CAM at the time of production but cannot be verified prior to actual product construction such as temperature specifications. In addition, material substitutions can be recommended. Using AI's search techniques, the translator can search for the best material combination while at the same time checking for the effects of material substitution.

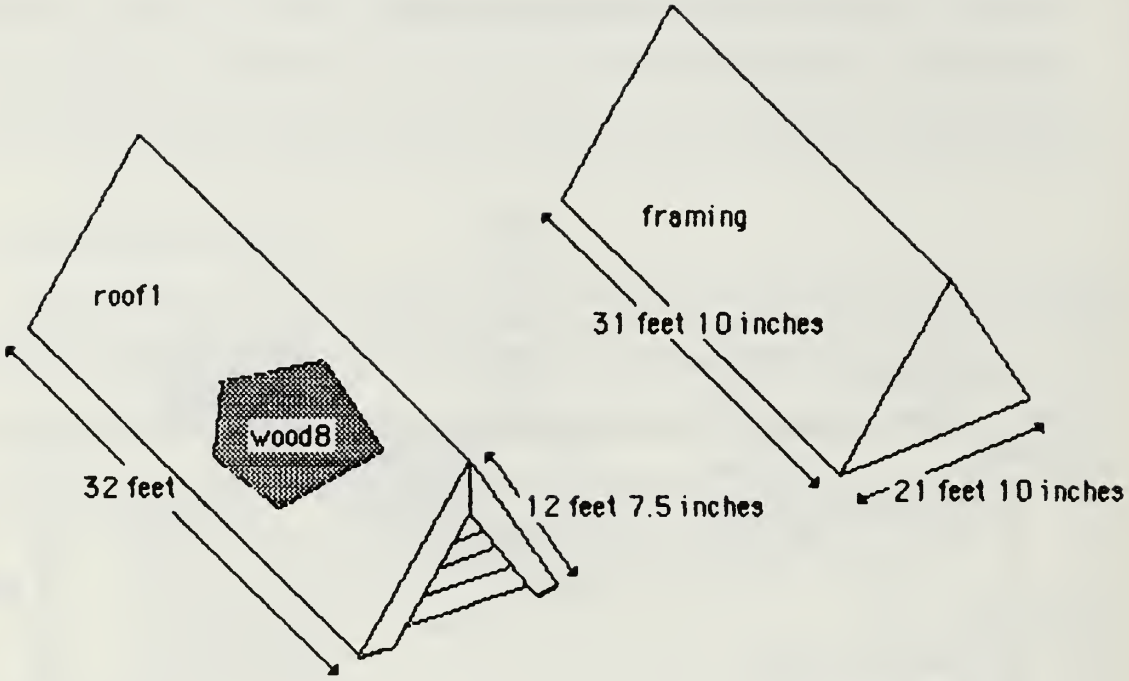
B. RECOMMENDATIONS

The next step in this research is to use an actual operating CAD system to generate the design data and schema data for input to an expert translator. The test products designed by that CAD system should be ones for which there exists a product CAM system for additional research.

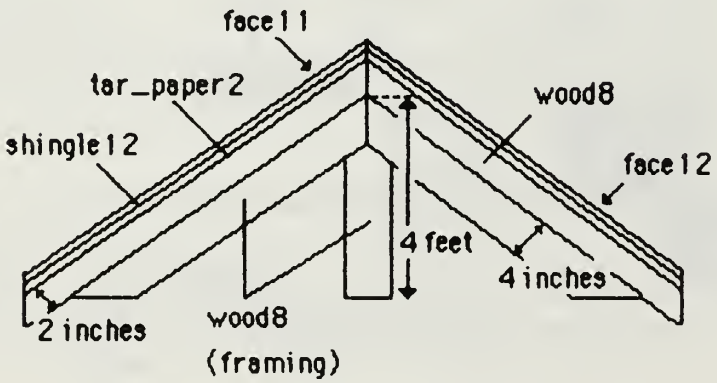
APPENDIX A

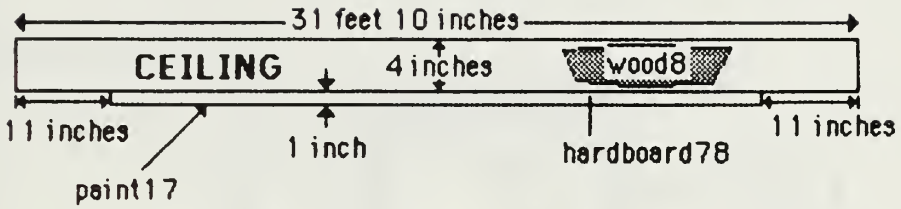
HOUSE1 DESIGN



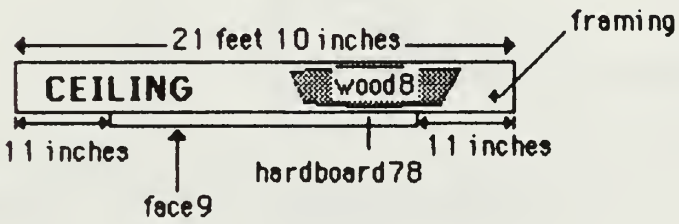


roof for house1

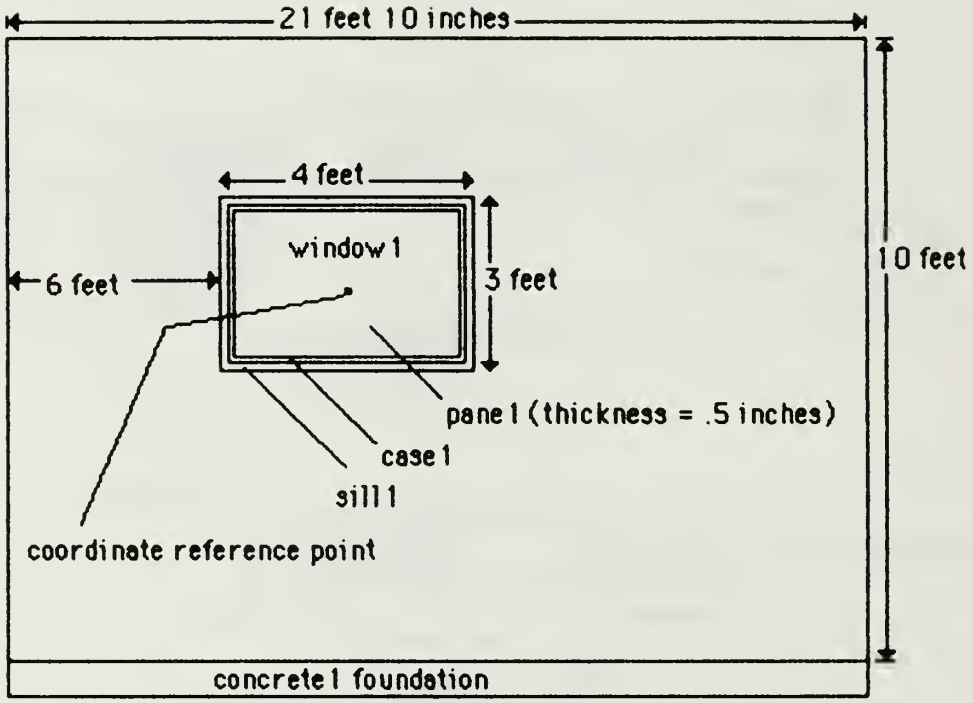




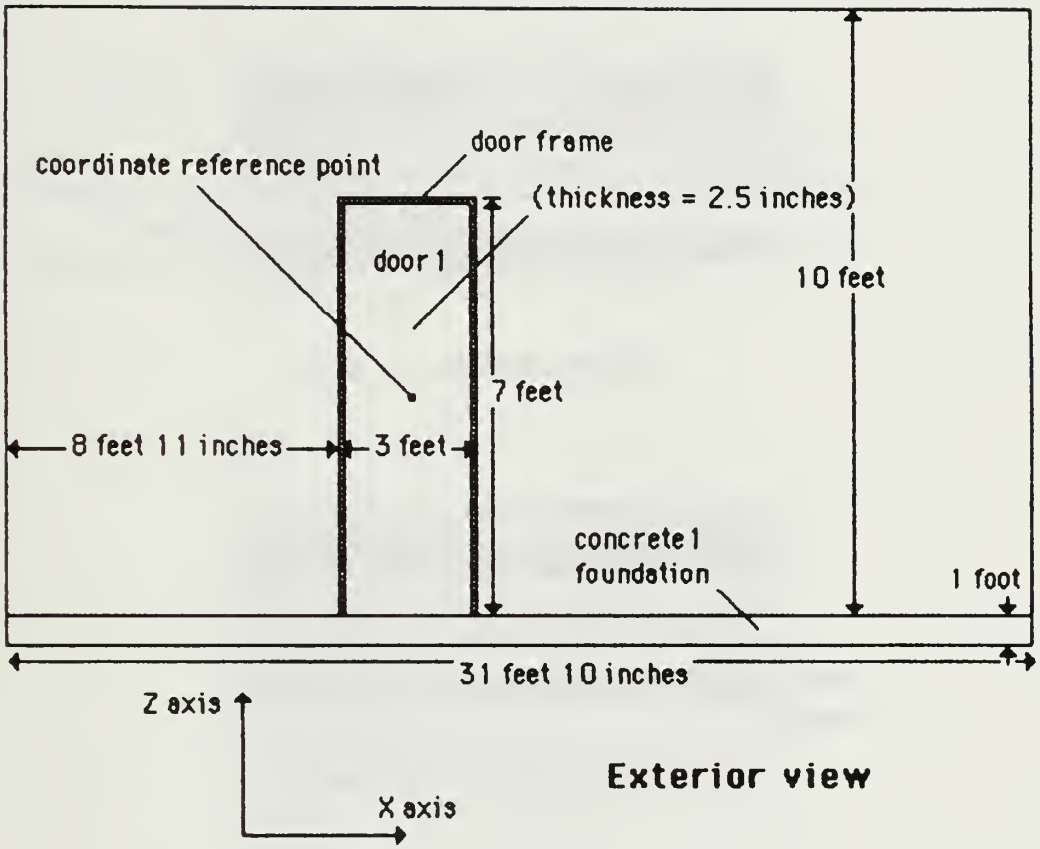
Parallel to X axis



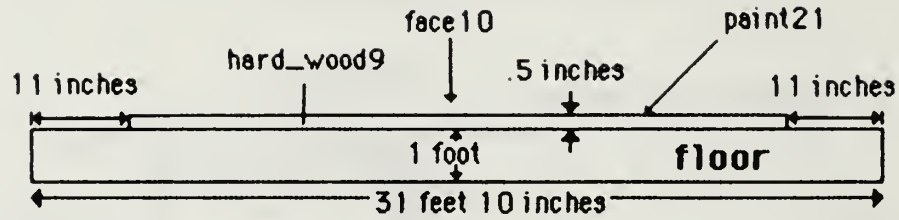
Parallel to Y axis



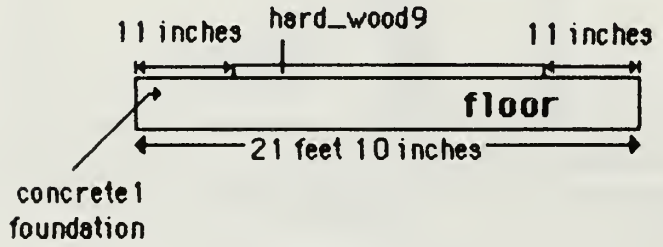
Exterior view



Exterior view



Parallel to X axis



Parallel to Y axis

APPENDIX B

PROTOTYPES

type house	
name :	
properties :	
** subtype	
contains	[...]

** => attribute is optional

type space	
name :	
properties :	
** subtype	
coordinates_X	
coordinates_Y	
coordinates_Z	
contains	[...]
part of	

** => attribute is optional

type roof	
name :	
properties :	
* finish color	
contains	[...]
part of	

* => attribute may be inherited
from face prototype

type exterior	
name :	
properties :	
* finish color	
contains	[...]
part of	

* => attribute may be inherited
from cover prototype

type room	
name :	
properties :	
** subtype	
coordinates_X	
coordinates_Y	
coordinates_Z	
contains	[...]
part of	

** => attribute is optional

type face	
name :	
properties :	
* finish color	
dimensions :	
height	
width	
depth	
** contains	[...]
normal_X	
normal_Y	
normal_Z	
part of	

* => attribute may be inherited
from cover prototype

** => items made from the same prototype
are listed according to their position
in the face

type door	
name :	
properties :	
material type	
finish type	
finish color	
knob type	
hinge type	
dimensions :	
height	
width	
depth	
face	
face	
coordinates_X	
coordinates_Y	
coordinates_Z	
part of	

type opening	
name :	
properties :	
geometry	
dimensions :	
** radius	
** height	
** width	
* depth	
face	
** face	
part of	

* => attribute may be inherited
from face prototype

** => attribute is optional

type sub_cover	
name :	
properties :	
material type	
** finish color	
dimensions :	
* height	
* width	
** depth	
part of	

* => attribute may be inherited
from face prototype

** => attribute is optional

type insulation	
name :	
properties :	
material type	
dimensions :	
* height	
* width	
depth	
face	
** face	
part of	

* => attribute may be inherited
from face prototype

** => attribute is optional

type cover	
name:	
properties:	
material type	
** finish color	
dimensions:	
* height	
* width	
** depth	
part of	

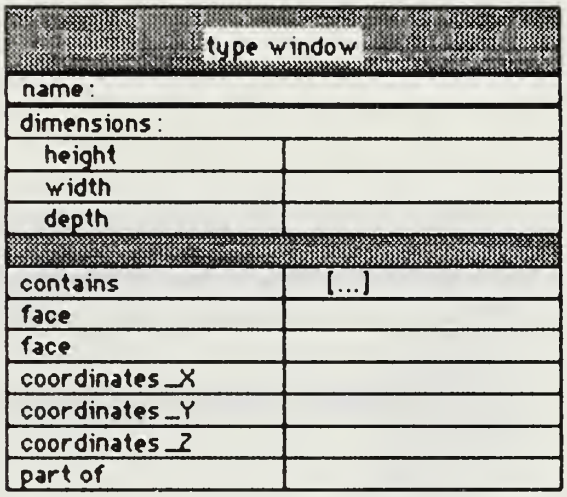
* => attribute may be inherited
from face prototype

** => attribute is optional

type frame	
name:	
properties:	
material type	
* finish color	
dimensions:	
* height	
* width	
depth	
face	
** face	
part of	

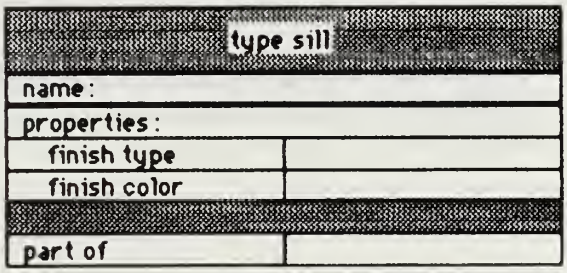
* => attribute may be inherited
from face prototype

** => attribute is optional


type window
name :
dimensions :
height
width
depth
contains
[...]
face
face
coordinates_X
coordinates_Y
coordinates_Z
part of

type pane
name :
part of

type case
name :
part of

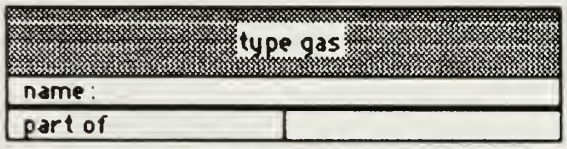

type sill
name :
properties :
finish type
finish color
part of

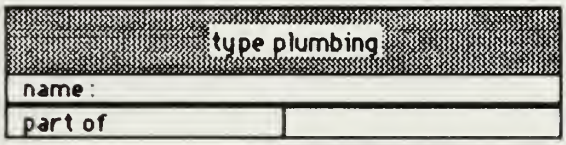
type connection	
name :	
properties :	
geometry	
dimensions :	
** radius	
** height	
** width	
* depth	
face	
** face	
part of	

* => attribute may be inherited
from face prototype

** => attribute is optional

type electric	
name :	
part of	


type gas
name :
part of


type plumbing
name :
part of

type heating
name :
part of

APPENDIX C

```
/* Interface File */

start :-
  not(begin_stds_check),
  not(begin_operations),
  not(set_neg_area),
  not(raw_materials_needed),
  not(materials_report),
  not(report_subst).

begin_stds_check :- is_a(Extens,Intens),
  write(' check for '),write(Intens),write(' '),write(Extens),nl,nl,
  check(Extens,Intens),fail.

check(Extens,Intens) :-
  property(Extens,material_type,Material),
  material(Material,Spec_Mat,_,_,_,_,_,_,_,_,_,_,_),
  comment_for(Intens,Spec_Mat,Class),
  comment(Class,Comment),
  write('     '),write(Comment),nl,nl.

check(Extens,Intens) :-
  property(Extens,material_type,Material),
  material(Material,Spec_Mat,_,_,_,_,_,_,_,_,_,_,_),
  check_for(Intens,Spec_Mat,Class),
  member(Material,Class),
  write('     '),write(Intens),write(' '),write(Extens),
  write(' meets requirements; allowed substitutes are:'),nl,nl,
  member(Other_Mat,Class),
  not(Other_Mat = Material),
  write('     - '),write(Other_Mat),nl,nl,
  assertz(substitute(Extens,Other_Mat)).

check(Extens,Intens) :-
  property(Extens,material_type,Material),
  material(Material,Spec_Mat,_,_,_,_,_,_,_,_,_,_,_),
  check_for(Intens,Spec_Mat,Class),
  not(member(Material,Class)),
  write('     '),write(Intens),write(' '),write(Extens),
  write(' does not meet requirements; allowed substitutes are:'),nl,nl,
  member(Other_Mat,Class),
  write('     - '),write(Other_Mat),nl,nl,
  assertz(substitute(Extens,Other_Mat)).
```

```
check(Extens,door) :-
  dimension(Extens,Dimension,Z,Units),
  minimum(door,Extens,Dimension,X,Unitx),
  maximum(door,Extens,Dimension,Y,Unity),
  convert(X,Unitx,Min,Units),
  convert(Y,Unity,Max,Units),
  check_standards(door,Extens,Dimension,Z,Min,Max).

check(Extens,pane) :-
  property(Extens,quality,Value),
  minimum(pane,Extens,quality,Min),
  check_standards(pane,Extens,Value,Min).

check_standards(Intens,Extens,Dimension,Value,Min,Max) :-
  not(Min > Value), not(Value > Max),
  write('     '),write(Intens),write(' '),write(Extens),
  write(' passed - '),write(Dimension),nl,nl,!.

check_standards(Intens,Extens,Dimension,Value,Min,Max) :-
  Min > Value,
  write('     '),write(Intens),(' '),write(Extens),
  write(' failed minimum - '),
  write(Dimension),nl,nl,!.

check_standards(Intens,Extens,Dimension,Value,Min,Max) :-
  Value > Max,
  write('     '),write(Intens),write(' '),write(Extens),
  write(' failed maximum - '),
  write(Dimension),nl,nl,!.

check_standards(pane,Extens,Value,Min) :-
  not(Min > Value),
  write('     '),write('pane '),write(Extens),
  write(' passed quality check'),nl,nl,!.

check_standards(pane,Extens,Value,Min) :-
  Min > Value,
  part_of(Extens,Window),
  is_a(Window,window),
  write('     '),write('pane '),write(Extens),
  write(' failed minimum quality check'),nl,
  write('       - part of '),write(Window),nl,nl,!.

begin_operations :-
  is_a(H,house),
  not(do_assembly(H)),
  not(operations_report(H)),
  fail.

do_assembly(H) :- assemble(H,house),fail.
```

```
operations_report(H) :-
  nl,nl,
  write('*************************************************'),nl,
  write('*                                               *'),nl,
  write(' Production Sequence Report for '),
  write(H),nl,nl,
  print_style(H),
  write('*                                               *'),nl,
  write('*************************************************'),nl,nl,!,
  operation(Extens,Function,Attribute1,Attribute2),
  print_operation(Extens,Function,Attribute1,Attribute2),
  fail.

print_operation(comment,Comment,_,_) :-
  nl,
  write('*************************************************'),nl,
  write('*                                               *'),nl,
  write(' comment : '),
  write(Comment),
  nl,
  write('*                                               *'),nl,
  write('*************************************************'),nl,nl,!.

print_operation(Extens,Attribute1,Attribute2,Attribute3) :-
  write(Extens),
  name(Extens,L1),
  length(L1,N1),
  tab(15 - N1),
  write(Attribute1),
  get_name_len(Attribute1,N2),
  tab(15 - N2),
  write(Attribute2),
  get_name_len(Attribute2,N3),
  tab(17 - N3),
  write(Attribute3),nl,!.

get_name_len(Name,Len) :-
  number(Name),
  not(integer(Name)),
  name(Name,L1),
  length(L1,N1),
  Len is (N1 - 4),!.

get_name_len(Name,Len) :-
  name(Name,L1),
  length(L1,Len),!.
```

```
print_style(H) :-
  property(H,subtype,Hstyle),
  write('  - house style is '),
  write(Hstyle),nl,
  write('    and consists of '),
  contains(H,L),
  write(L),nl,!.

print_style(H).

/* routines to calculate surface area of faces taken up by */
/* doors, windows, openings, and connections                */
set_neg_area :-
  is_a(Extens,face),
  set_neg_area2(Extens,[],0,feet),fail.

set_neg_area2(Face,L,Area,Units) :-
  face(Extens,Face),
  not(member(Extens,L)),
  is_a(Extens,window),
  dimension(Extens,height,Ht,Htunits),
  dimension(Extens,width,Wd,Wdunits),
  convert(Ht,Htunits,New_Ht,Units),
  convert(Wd,Wdunits,New_Wd,Units),
  New_Area is (Area + (New_Ht * New_Wd)),
  set_neg_area2(Face,[Extens|L],New_Area,Units),!.

set_neg_area2(Face,L,Area,Units) :-
  face(Extens,Face),
  not(member(Extens,L)),
  is_a(Extens,door),
  dimension(Extens,height,Ht,Htunits),
  dimension(Extens,width,Wd,Wdunits),
  convert(Ht,Htunits,New_Ht,Units),
  convert(Wd,Wdunits,New_Wd,Units),
  New_Area is (Area + (New_Ht * New_Wd)),
  set_neg_area2(Face,[Extens|L],New_Area,Units),!.

set_neg_area2(Face,L,Area,Units) :-
  face(Extens,Face),
  not(member(Extens,L)),
  is_a(Extens,connection),
  geometry(Extens,rectangle),
  dimension(Extens,height,Ht,Htunits),
  dimension(Extens,width,Wd,Wdunits),
  convert(Ht,Htunits,New_Ht,Units),
  convert(Wd,Wdunits,New_Wd,Units),
  New_Area is (Area + (New_Ht * New_Wd)),
  set_neg_area2(Face,[Extens|L],New_Area,Units),!.
```

```
set_neg_area2(Face,L,Area,Units) :-
  face(Extens,Face),
  not(member(Extens,L)),
  is_a(Extens,connection),
  geometry(Extens,square),
  dimension(Extens,height,Ht,Htunits),
  convert(Ht,Htunits,New_Ht,Units),
  New_Area is (Area + (New_Ht * New_Ht)),
  set_neg_area2(Face,[Extens|L],New_Area,Units),!.

set_neg_area2(Face,L,Area,Units) :-
  face(Extens,Face),
  not(member(Extens,L)),
  is_a(Extens,connection),
  geometry(Extens,square),
  dimension(Extens,width,Wd,Wdunits),
  convert(Wd,Wdunits,New_Wd,Units),
  New_Area is (Area + (New_Wd * New_Wd)),
  set_neg_area2(Face,[Extens|L],New_Area,Units),!.

set_neg_area2(Face,L,Area,Units) :-
  face(Extens,Face),
  not(member(Extens,L)),
  is_a(Extens,connection),
  geometry(Extens,circle),
  dimension(Extens,radius,Rd,Rdunits),
  convert(Rd,Rdunits,New_Rd,Units),
  Pi is 3.14159,
  New_Area is (Area + (Pi * New_Rd * New_Rd)),
  set_neg_area2(Face,[Extens|L],New_Area,Units),!.

set_neg_area2(Face,L,Area,Units) :-
  face(Extens,Face),
  not(member(Extens,L)),
  is_a(Extens,opening),
  geometry(Extens,rectangle),
  dimension(Extens,height,Ht,Htunits),
  dimension(Extens,width,Wd,Wdunits),
  convert(Ht,Htunits,New_Ht,Units),
  convert(Wd,Wdunits,New_Wd,Units),
  New_Area is (Area + (New_Ht * New_Wd)),
  set_neg_area2(Face,[Extens|L],New_Area,Units),!.
```

```
set_neg_area2(Face,L,Area,Units) :-
  face(Extens,Face),
  not(member(Extens,L)),
  is_a(Extens,opening),
  geometry(Extens,square),
  dimension(Extens,height,Ht,Htunits),
  convert(Ht,Htunits,New_Ht,Units),
  New_Area is (Area + (New_Ht * New_Ht)),
  set_neg_area2(Face,[Extens|L],New_Area,Units),!.

set_neg_area2(Face,L,Area,Units) :-
  face(Extens,Face),
  not(member(Extens,L)),
  is_a(Extens,opening),
  geometry(Extens,square),
  dimension(Extens,width,Wd,Wdunits),
  convert(Wd,Wdunits,New_Wd,Units),
  New_Area is (Area + (New_Wd * New_Wd)),
  set_neg_area2(Face,[Extens|L],New_Area,Units),!.

set_neg_area2(Face,L,Area,Units) :-
  face(Extens,Face),
  not(member(Extens,L)),
  is_a(Extens,opening),
  geometry(Extens,circle),
  dimension(Extens,radius,Rd,Rdunits),
  convert(Rd,Rdunits,New_Rd,Units),
  Pi is 3.14159,
  New_Area is (Area + (Pi * New_Rd * New_Rd)),
  set_neg_area2(Face,[Extens|L],New_Area,Units),!.

set_neg_area2(Face,L,Area,Units) :-
  assertz(get_neg_area(Face,Area,Units)).

get_area(Extens,Area,Units) :-
  part_of(Extens,Face),
  dimension(Extens,height,Ht,Htunits),
  dimension(Extens,width,Wd,Wdunits),
  get_neg_area(Face,Neg_Area,Units),
  convert(Ht,Htunits,New_Ht,Units),
  convert(Wd,Wdunits,New_Wd,Units),
  Area is ((New_Ht * New_Wd) - Neg_Area),!.

get_area(Extens,Area,Units) :-
  part_of(Extens,Face),
  dimension(Face,height,Ht,Htunits),
  dimension(Face,width,Wd,Wdunits),
  get_neg_area(Face,Neg_Area,Units),
  convert(Ht,Htunits,New_Ht,Units),
  convert(Wd,Wdunits,New_Wd,Units),
  Area is ((New_Ht * New_Wd) - Neg_Area),!.
```

```
get_area(Extens,Area,Units) :-
  part_of(Extens,Face),
  dimension(Face,height,Ht,Htunits),
  dimension(Extens,width,Wd,Wdunits),
  get_neg_area(Face,Neg_Area,Units),
  convert(Ht,Htunits,New_Ht,Units),
  convert(Wd,Wdunits,New_Wd,Units),
  Area is ((New_Ht * New_Wd) - Neg_Area),!.

get_area(Extens,Area,Units) :-
  part_of(Extens,Face),
  dimension(Extens,height,Ht,Htunits),
  dimension(Face,width,Wd,Wdunits),
  get_neg_area(Face,Neg_Area,Units),
  convert(Ht,Htunits,New_Ht,Units),
  convert(Wd,Wdunits,New_Wd,Units),
  Area is ((New_Ht * New_Wd) - Neg_Area),!.

materials_report :-
  assertz(mat_cost(0)),
  nl,nl,nl,write('          Raw Materials Report'),nl,nl,
  write('  Item          Cost        Units Required'),nl,nl,
  material_list(Material,Num_Units,Item_Cost),
  New_Cost is floor(Item_Cost),
  print_mat_report(Material,Num_Units,New_Cost),
  update_mat_cost(New_Cost),fail.

materials_report :-
  mat_cost(Total),nl,nl,
  write('*************************************'),nl,
  write('*                                   *'),nl,
  write(' Total material cost is $'),
  write(Total),nl,
  write('*                                   *'),nl,
  write('*************************************'),nl,nl,nl,
  fail.

update_mat_cost(Item_Cost) :-
  retract(mat_cost(Total)),
  New_Total is (Total + Item_Cost),
  assertz(mat_cost(New_Total)),!.
```

```
print_mat_report(Material,Num_Units,Tot_Cost) :-
  write(Material),
  name(Material,L1),
  length(L1,N1),
  tab(17 - N1),
  write('$'),write(Tot_Cost),
  name(Tot_Cost,L2),
  length(L2,N2),
  tab(15 - N2),
  write(Num_Units),nl,nl,!.

report_subst :-
  nl,nl,nl,
  write('  Start Raw Materials Report (w/ substitute)'),
  nl,nl,nl,fail.

report_subst :-
  substitute(Extens,Subst_Mat),
  replace_data(Extens,Subst_Mat),
  not(raw_materials_needed),
  not(materials_report),
  restore_data,fail.

replace_data(Extens,Subst_Mat) :-
  retract(mat_cost(_)),
  retract(material_list(_,_,_)),fail.

replace_data(Extens,Subst_Mat) :-
  retract(substitute(Extens,Subst_Mat)),
  retract(property(Extens,material_type,Material)),
  write('*************************************************************'),nl,
  write('*                                                           *'),nl,
  write(' '),write(Extens),write(': substitute '),
  write(Subst_Mat),write(' for '),write(Material),nl,
  write('*                                                           *'),nl,
  write('*************************************************************'),nl,nl,
  assertz(property(Extens,material_type,Subst_Mat)),
  assertz(temp(Extens,material_type,Material)),!.

restore_data :-
  retract(temp(Extens,material_type,Material)),
  retract(property(Extens,material_type,_)),
  assertz(property(Extens,material_type,Material)),!.
```

```
/* Standards File */

minimum(door,door1,width,32,inches).
minimum(door,door1,height,6,feet).
maximum(door,door1,width,4,feet).
maximum(door,door1,height,7,feet).
minimum(door,_,depth,2,inches).
maximum(door,_,depth,3,inches).

minimum(pane,_,quality,3).

comment(masonry,'approved methods must be used for building masonry walls
     when outside air temperature drops below 40 degrees farenheit').
comment_for(cover,brick,masonry).
comment_for(cover,concrete_block,masonry).
comment_for(sub_cover,brick,masonry).
comment_for(sub_cover,concrete_block,masonry).

comment(framing,'grade marks must be clearly visible on all framing
     members for inspection').
comment_for(frame,wood,framing).

check_for(sub_cover,tar_paper,[tar_paper1,tar_paper2,tar_paper3]).
```

```
/* Assembly File */

/* start with information on face normals */
assemble(H,house) :-
  assertz(operation(comment,'normal for each face listed',_,_)),
  assertz(operation('FACE','X','Y','Z')),
  assertz(operation('_____','__','__','__')),
  is_a(Face,face),
  normal_X(Face,X),
  normal_Y(Face,Y),
  normal_Z(Face,Z),
  assertz(operation(Face,X,Y,Z)).

/* start with frame */
assemble(H,house) :-
  assertz(operation(comment,'erect foundation and frame',_,_)).

/* do foundation frame */
assemble(H,house) :-
  is_a(Yface,face),
  trans_partof(Yface,H),
  normal_Z(Yface,1),
  contains(Yface,L),
  member(Frame,L),
  is_a(Frame,frame),
  property(Frame,material_type,Mtype),
  assertz(operation(Frame,assemble,'material type: ',Mtype)).

/* do frame perpendicular to ground */
assemble(H,house) :-
  is_a(Yface,face),
  trans_partof(Yface,H),
  normal_Y(Yface,0),
  normal_Z(Yface,0),
  contains(Yface,L),
  member(Frame,L),
  is_a(Frame,frame),
  property(Frame,material_type,Mtype),
  assertz(operation(Frame,assemble,'material type: ',Mtype)).

assemble(H,house) :-
  is_a(Yface,face),
  trans_partof(Yface,H),
  normal_X(Yface,0),
  normal_Z(Yface,0),
  contains(Yface,L),
  member(Frame,L),
  is_a(Frame,frame),
  property(Frame,material_type,Mtype),
  assertz(operation(Frame,assemble,'material type: ',Mtype)).
```

```
/* ceiling frame */
assemble(H,house) :-
  is_a(Yface,face),
  trans_partof(Yface,H),
  normal_Z(Yface,-1),
  contains(Yface,L),
  member(Frame,L),
  is_a(Frame,frame),
  property(Frame,material_type,Mtype),
  assertz(operation(Frame,assemble,'material type: ',Mtype)).

/* roof frame */
assemble(H,house) :-
  is_a(Roof,roof),
  trans_partof(Roof,H),
  is_a(Yface,face),
  trans_partof(Yface,Roof),
  contains(Yface,L),
  member(Frame,L),
  is_a(Frame,frame),
  property(Frame,material_type,Mtype),
  assertz(operation(Frame,assemble,'material type: ',Mtype)).

/* now put doors in place */
assemble(H,house) :-
  assertz(operation(comment,'put door framing in place',_,_)).

assemble(H,house) :-
  is_a(Door,door),
  trans_partof(Door,H),
  property(Door,material_type,Mtype),
  assertz(operation(Door,assemble,'material type: ',Mtype)),
  get_faces(Door,Face1,Face2),
  assertz(operation('','- attach to:',Face1,Face2)),
  part_of(Door,Face3),
  assertz(operation('','- location','relative to',Face3)),
  coordinates_X(local,Door,X,Units_X),
  coordinates_Y(local,Door,Y,Units_Y),
  coordinates_Z(local,Door,Z,Units_Z),
  assertz(operation('',' X coordinate',X,Units_X)),
  assertz(operation('',' Y coordinate',Y,Units_Y)),
  assertz(operation('',' Z coordinate',Z,Units_Z)).

/* put window sills in place */
assemble(H,house) :-
  assertz(operation(comment,'put window framing in place',_,_)).
```

```
assemble(H,house) :-
  is_a(W,window),
  trans_partof(W,H),
  contains(W,L),
  member(Sill,L),
  is_a(Sill,sill),
  assertz(operation(Sill,assemble,'window sill for: ',W)),
  get_faces(W,Face1,Face2),
  assertz(operation('','- attach to: ',Face1,Face2)),
  part_of(W,Face3),
  assertz(operation('','- location','relative to',Face3)),
  coordinates_X(local,W,X,Units_X),
  coordinates_Y(local,W,Y,Units_Y),
  coordinates_Z(local,W,Z,Units_Z),
  assertz(operation('',' X coordinate',X,Units_X)),
  assertz(operation('',' Y coordinate',Y,Units_Y)),
  assertz(operation('',' Z coordinate',Z,Units_Z)).

/* put up exterior siding */
assemble(H,house) :-
  assertz(operation(comment,'put up exterior siding',_,_)).

assemble(H,house) :-
  is_a(E,exterior),
  trans_partof(E,H),
  contains(E,L),
  assemble(L,face).

/* put up roof */
assemble(H,house) :-
  assertz(operation(comment,'put up roof',_,_)).

assemble(H,house) :-
  is_a(R,roof),
  trans_partof(R,H),
  contains(R,L),
  assemble(L,face).

/* put up faces for each room */
assemble(H,house) :-
  assertz(operation(comment,'put up faces for each room',_,_)).

assemble(H,house) :-
  is_a(R,room),
  trans_partof(R,H),
  contains(R,L),
  assemble(L,face).

/* put up windows */
assemble(H,house) :-
  assertz(operation(comment,'put windows in place',_,_)).
```

```
assemble(H,house) :-
  is_a(W,window),
  trans_partof(W,H),
  contains(W,L),
  member(P,L),
  is_a(P,pane),
  member(C,L),
  is_a(C,case),
  assertz(operation(W,'complete using',P,C)).

/* take care of finish on windows and doors */
assemble(H,house) :-
  assertz(operation(comment,'put finish on windows and doors',_,_)).

assemble(H,house) :-
  finish.

/* take care of door knobs and hinges */
assemble(H,house) :-
  assertz(operation(comment,'put on door knobs and hinges',_,_)).

assemble(H,house) :-
  is_a(D,door),
  trans_partof(D,H),
  assemble(D,door).

/* take care of paint on faces */
assemble(H,house) :-
  assertz(operation(comment,'put final paint on faces',_,_)).

assemble(H,house) :-
  is_a(R,roof),
  trans_partof(R,H),
  contains(R,L),
  paint_face(L).

assemble(H,house) :-
  is_a(E,exterior),
  trans_partof(E,H),
  contains(E,L),
  paint_face(L).

assemble(H,house) :-
  is_a(R,room),
  trans_partof(R,H),
  contains(R,L),
  paint_face(L).
```

```
/* routines to put up sub_covers and covers for a given     */
/* list of faces supplied as first argument; these routines */
/* look for common materials to help set priority; all      */
/* sub_covers are handled prior to covers;                  */
/* covers which are paint are left to be performed at a     */
/* later time; all sub_covers and covers                    */
/* associated with the floor are performed last             */

assemble(L,face) :-
  assemble1(L,[],face).

assemble(L,face) :-
  member(Face,L),
  normal_Z(Face,1),
  assertz(operation(comment,'build floor as last step',_,_)),
  contains(Face,L1),
  assemble2([L1],[L1],face).

assemble1(L,L1,face) :-
  member(Face,L),
  not(normal_Z(Face,1)),
  delete(Face,L,L2),
  contains(Face,L3),
  assemble1(L2,[L3|L1],face),!.

assemble1(L,L1,face) :-
  assemble2(L1,L1,face),!.

assemble2(Full_L,L,face) :-
  member(Face,L),
  delete(Face,L,L1),
  member(Item,Face),
  is_a(Item,sub_cover),
  property(Item,material_type,Mtype),
  operation(Y,_,_,Mtype),
  member(Face1,Full_L),
  member(Y,Face1),
  assertz(operation(Item,assemble,'material type: ',Mtype)),
  delete(Item,Face,Face2),
  assemble2(Full_L,[Face2|L1],face),!.

assemble2(Full_L,L,face) :-
  member(Face,L),
  delete(Face,L,L1),
  member(Item,Face),
  is_a(Item,sub_cover),
  property(Item,material_type,Mtype),
  assertz(operation(Item,assemble,'material type: ',Mtype)),
  delete(Item,Face,Face1),
  assemble2(Full_L,[Face1|L1],face),!.
```

```
assemble2(Full_⊥,L,face) :-
  member(Face,L),
  delete(Face,L,L1),
  member(Item,Face),
  is_a(Item,cover),
  property(Item,material_type,Mtype),
  not(liquid(Mtype,paint,_,_,_,_,_)),
  operation(Y,_,_,Mtype),
  member(Face1,Full_⊥),
  member(Y,Face1),
  assertz(operation(Item,assemble,'material type: ',Mtype)),
  delete(Item,Face,Face2),
  assemble2(Full_⊥,[Face2|L1],face),!.

assemble2(Full_⊥,L,face) :-
  member(Face,L),
  delete(Face,L,L1),
  member(Item,Face),
  is_a(Item,cover),
  property(Item,material_type,Mtype),
  not(liquid(Mtype,paint,_,_,_,_,_)),
  assertz(operation(Item,assemble,'material type: ',Mtype)),
  delete(Item,Face,Face1),
  assemble2(Full_⊥,[Face1|L1],face),!.

assemble2(Full_⊥,L,face).

/* take care of finishes */
finish :-
  property(F,finish_type,Ftype),
  property(F,finish_color,Fcolor),
  assertz(operation(F,finish,Ftype,Fcolor)).

/* assemble door knob */
assemble(D,door) :-
  property(D,knob_type,Ktype),
  assertz(operation(D,assemble,knob,Ktype)).

/* assemble door hinges */
assemble(D,door) :-
  property(D,hinge_type,Htype),
  assertz(operation(D,assemble,hinge,Htype)).
```

```
/* routines to apply paint to faces; acts on covers only */
paint_face(L) :-
  member(Face,L),
  normal_Z(Face,-1),
  contains(Face,L1),
  member(Cover,L1),
  is_a(Cover,cover),
  property(Cover,material_type,Mtype),
  liquid(Mtype,paint,_,_,_,_,_),
  assertz(operation(Cover,paint,'material type: ',Mtype)),
  delete(Face,L,L2),
  paint_face(L2),!.

paint_face(L) :-
  member(Face,L),
  normal_Y(Face,0),
  normal_Z(Face,0),
  contains(Face,L1),
  member(Cover,L1),
  is_a(Cover,cover),
  property(Cover,material_type,Mtype),
  liquid(Mtype,paint,_,_,_,_,_),
  assertz(operation(Cover,paint,'material type: ',Mtype)),
  delete(Face,L,L2),
  paint_face(L2),!.

paint_face(L) :-
  member(Face,L),
  normal_X(Face,0),
  normal_Z(Face,0),
  contains(Face,L1),
  member(Cover,L1),
  is_a(Cover,cover),
  property(Cover,material_type,Mtype),
  liquid(Mtype,paint,_,_,_,_,_),
  assertz(operation(Cover,paint,'material type: ',Mtype)),
  delete(Face,L,L2),
  paint_face(L2),!.

paint_face(L) :-
  member(Face,L),
  normal_Z(Face,-1),
  contains(Face,L1),
  member(Cover,L1),
  is_a(Cover,cover),
  property(Cover,material_type,Mtype),
  liquid(Mtype,paint,_,_,_,_,_),
  assertz(operation(Cover,paint,'material type: ',Mtype)),
  delete(Face,L,L2),
  paint_face(L2),!.
```

```
paint_face(L) :-
  member(Face,L),
  contains(Face,L1),
  member(Cover,L1),
  is_a(Cover,cover),
  property(Cover,material_type,Mtype),
  liquid(Mtype,paint,_,_,_,_,_),
  assertz(operation(Cover,paint,'material type: ',Mtype)),
  delete(Face,L,L2),
  paint_face(L2),!.

paint_face(L).

/* routine to get the two faces which an item is associated with */
get_faces(Item,Face1,Face2) :-
  face(Item,Face1),
  face(Item,Face2),
  not(Face1 = Face2),!.
```

```
/* Materials File */

/* materials for doors */
raw_materials_needed :-
  is_a(Extens,door),
  material(Extens,_,_,_,_,_,_,_,_,_,_,_,Cost),
  add_material(Extens,1,Cost),fail.

raw_materials_needed :-
  is_a(Extens,door),
  property(Extens,finish_type,Paint),
  liquid(Paint,_,Area_Cov,Area_Units,_,_,Cost),
  dimension(Extens,height,Org_Ht,Ht_Units),
  dimension(Extens,width,Org_Wd,Wd_Units),
  dimension(Extens,depth,Org_Dp,Dp_Units),
  convert(Org_Ht,Ht_Units,New_Ht,Area_Units),
  convert(Org_Wd,Wd_Units,New_Wd,Area_Units),
  convert(Org_Dp,Dp_Units,New_Dp,Area_Units),
  Area is ((2 * New_Ht * New_Wd) + (2 * New_Ht * New_Dp) +
    (2 * New_Wd * New_Dp)),
  Num_Units is (Area / Area_Cov),
  Tot_Cost is (Num_Units * Cost),
  add_material(Paint,Num_Units,Tot_Cost),fail.

/* materials for windows */
raw_materials_needed :-
  is_a(Extens,window),
  material(Extens,_,_,_,_,_,_,_,_,_,_,_,Cost),
  add_material(Extens,1,Cost),fail.

raw_materials_needed :-
  is_a(Window,window),
  part_of(Extens,Window),
  is_a(Extens,sill),
  property(Extens,finish_type,Paint),
  liquid(Paint,_,Area_Cov,Area_Units,_,_,Cost),
  convert(Area_Cov,Area_Units,New_Area,feet),
  New_Area2 is ((New_Area * New_Area) / Area_Cov),
  dimension(Window,height,Org_Ht,Ht_Units),
  dimension(Window,width,Org_Wd,Wd_Units),
  part_of(Window,Face),
  dimension(Face,depth,Org_Dp,Dp_Units),
  convert(Org_Ht,Ht_Units,New_Ht,Area_Units),
  convert(Org_Wd,Wd_Units,New_Wd,Area_Units),
  convert(Org_Dp,Dp_Units,New_Dp,Area_Units),
  Area is ((2 * New_Ht * New_Dp) + (2 * New_Wd * New_Dp)),
  Num_Units is (Area / Area_Cov),
  Tot_Cost is (Num_Units * Cost),
  add_material(Paint,Num_Units,Tot_Cost),fail.
```

```
/* materials for frames; assumes 1 square foot of area */
/* requires a 1 foot length of frame wood              */
raw_materials_needed :-
  is_a(Extens,frame),
  dimension(Extens,height,Height,Ht_Units),
  dimension(Extens,width,Width,Wd_Units),
  convert(Height,Ht_Units,New_Height,feet),
  convert(Width,Wd_Units,New_Width,feet),
  property(Extens,material_type,Material),
  material(Material,wood,Ht,Htunits,Wd,Wdunits,Dp,Dpunits,_,_,_,_,Cost),
  longest_dimension(Ht,Htunits,Wd,Wdunits,Dp,Dpunits,Len,Lenunits),
  convert(Len,Lenunits,New_Len,feet),
  Area is (New_Height * New_Width),
  Num_Units is (Area / New_Len),
  Tot_Cost is (Num_Units * Cost),
  add_material(Material,Num_Units,Tot_Cost),fail.

raw_materials_needed :-
  is_a(Extens,frame),
  not(dimension(Extens,width,_,_)),
  not(dimension(Extens,height,_,_)),
  property(Extens,material_type,Material),
  material(Material,wood,Ht,Htunits,Wd,Wdunits,Dp,Dpunits,_,_,_,_,Cost),
  get_area(Extens,Area,Units),
  convert(Area,Units,New_Area,feet),
  New_Area2 is ((New_Area * New_Area) / Area),
  longest_dimension(Ht,Htunits,Wd,Wdunits,Dp,Dpunits,Len,Lenunits),
  convert(Len,Lenunits,New_Len,feet),
  Num_Units is (Area / New_Len),
  Tot_Cost is (Num_Units * Cost),
  add_material(Material,Num_Units,Tot_Cost),fail.
```

```
raw_materials_needed :-
  normal_Z(Face,-1),
  part_of(Extens,Face),
  is_a(Extens,frame),
  property(Extens,material_type,Material),
  material(Material,wood,Ht,Htunits,Wd,Wdunits,Dp,Dpunits,_,_,_,_,Cost),
  longest_dimension(Ht,Htunits,Wd,Wdunits,Dp,Dpunits,Len,Lenunits),
  convert(Len,Lenunits,New_Len,feet),
  is_a(Roof,roof),
  part_of(Face2,Roof),
  is_a(Face2,face),
  part_of(Extens2,Face2),
  is_a(Extens2,frame),
  normal_Z(Face2,CosZ),
  dimension(Extens2,height,Ht_face,Ht_face_units),
  convert(Ht_face,Ht_face_units,New_Ht_face,feet),
  dimension(Extens2,width,Wd_face,Wd_face_units),
  convert(Wd_face,Wd_face_units,New_Wd_face,feet),
  SinZ is (sqrt(1 - (CosZ * CosZ))),
  Area is (SinZ * New_Ht_face * New_Wd_face),
  Area2 is (SinZ * New_Ht_face * CosZ * New_Ht_face * 2),
  Tot_Area is Area + Area2,
  Num_Units is (Tot_Area / New_Len),
  Tot_Cost is (Num_Units * Cost),
  add_material(Material,Num_Units,Tot_Cost),fail.
```

```
raw_materials_needed :-
  normal_Z(Face,-1),
  part_of(Extens,Face),
  is_a(Extens,frame),
  property(Extens,material_type,Material),
  material(Material,wood,Ht,Htunits,Wd,Wdunits,Dp,Dpunits,_,_,_,_,Cost),
  longest_dimension(Ht,Htunits,Wd,Wdunits,Dp,Dpunits,Len,Lenunits),
  convert(Len,Lenunits,New_Len,feet),
  is_a(Roof,roof),
  part_of(Face2,Roof),
  is_a(Face2,face),
  part_of(Extens2,Face2),
  is_a(Extens2,frame),
  normal_Z(Face2,CosZ),
  not(dimension(Extens2,height,_,_)),
  dimension(Face2,height,Ht_face,Ht_face_units),
  convert(Ht_face,Ht_face_units,New_Ht_face,feet),
  dimension(Face2,width,Wd_face,Wd_face_units),
  convert(Wd_face,Wd_face_units,New_Wd_face,feet),
  SinZ is (sqrt(1 - (CosZ * CosZ))),
  Area is (SinZ * New_Ht_face * New_Wd_face),
  Area2 is (SinZ * New_Ht_face * CosZ * New_Wd_face),
  Tot_Area is Area + Area2,
  Num_Units is (Tot_Area / New_Len),
  Tot_Cost is (Num_Units * Cost),
  add_material(Material,Num_Units,Tot_Cost),fail.

/* frame material of type "filler" */
raw_materials_needed :-
  is_a(Extens,frame),
  property(Extens,material_type,Material),
  filler(Material,_,Vol,Volunits,_,_,Cost),
  get_area(Extens,Area,Units),
  convert(Vol,Volunits,New_Vol,Units),
  New_Vol2 is ((New_Vol * New_Vol * New_Vol)/(Vol * Vol)),
  dimension(Extens,depth,Dp,Dpunits),
  convert(Dp,Dpunits,New_Dp,Units),
  Num_Units is (Area * New_Dp / New_Vol2),
  Tot_Cost is (Num_Units * Cost),
  add_material(Material,Num_Units,Tot_Cost),fail.
```

```
raw_materials_needed :-
  is_a(Extens,sub_cover),
  dimension(Extens,depth,Th,Thunits),
  property(Extens,material_type,Material),
  material(Material,_,Ht,Htunits,Wd,Wdunits,Dp,Dpunits,_,_,_,_,Cost),
  match(Ht,Htunits,Wd,Wdunits,Dp,Dpunits,Th,Thunits,Act_Ht,Units1,
    Act_Wd,Units2),
  get_area(Extens,Area,Units),
  convert(Act_Ht,Units1,Act_Ht2,Units),
  convert(Act_Wd,Units2,Act_Wd2,Units),
  Num_Units is (Area / (Act_Ht2 * Act_Wd2)),
  Tot_Cost is (Num_Units * Cost),
  add_material(Material,Num_Units,Tot_Cost),fail.

raw_materials_needed :-
  is_a(Extens,sub_cover),
  not(dimension(Extens,depth,Th,Thunits)),
  property(Extens,material_type,Material),
  material(Material,_,Ht,Htunits,Wd,Wdunits,Dp,Dpunits,_,_,_,_,Cost),
  get_area(Extens,Area,Units),
  convert(Ht,Htunits,New_Ht,Units),
  convert(Wd,Wdunits,New_Wd,Units),
  Num_Units is (Area / (New_Ht * New_Wd)),
  Tot_Cost is (Num_Units * Cost),
  add_material(Material,Num_Units,Tot_Cost),fail.

raw_materials_needed :-
  is_a(Extens,sub_cover),
  property(Extens,material_type,Paint),
  liquid(Paint,_,Area_Cov,Area_Units,_,_,Cost),
  get_area(Extens,Area,Units),
  convert(Area_Cov,Area_Units,New_Area,Units),
  New_Area2 is ((New_Area * New_Area) / Area_Cov),
  Num_Units is (Area / New_Area2),
  Tot_Cost is (Num_Units * Cost),
  add_material(Paint,Num_Units,Tot_Cost),fail.

raw_materials_needed :-
  is_a(Extens,cover),
  dimension(Extens,depth,Th,Thunits),
  property(Extens,material_type,Material),
  material(Material,_,Ht,Htunits,Wd,Wdunits,Dp,Dpunits,_,_,_,_,Cost),
  match(Ht,Htunits,Wd,Wdunits,Dp,Dpunits,Th,Thunits,Act_Ht,Units1,
    Act_Wd,Units2),
  get_area(Extens,Area,Units),
  convert(Act_Ht,inches,Act_Ht2,Units),
  convert(Act_Wd,inches,Act_Wd2,Units),
  Num_Units is (Area / (Act_Ht2 * Act_Wd2)),
  Tot_Cost is (Num_Units * Cost),
  add_material(Material,Num_Units,Tot_Cost),fail.
```

```
raw_materials_needed :-
  is_a(Extens,cover),
  not(dimension(Extens,depth,Th,Thunits)),
  property(Extens,material_type,Material),
  material(Material,_,Ht,Htunits,Wd,Wdunits,Dp,Dpunits,_,_,_,_,Cost),
  get_area(Extens,Area,Units),
  convert(Ht,Htunits,New_Ht,Units),
  convert(Wd,Wdunits,New_Wd,Units),
  Num_Units is (Area / (New_Ht * New_Wd)),
  Tot_Cost is (Num_Units * Cost),
  add_material(Material,Num_Units,Tot_Cost),fail.

raw_materials_needed :-
  is_a(Extens,cover),
  property(Extens,material_type,Paint),
  liquid(Paint,_,Area_Cov,Area_Units,_,_,Cost),
  get_area(Extens,Area,Units),
  convert(Area_Cov,Area_Units,New_Area,Units),
  New_Area2 is ((New_Area * New_Area) / Area_Cov),
  Num_Units is (Area / New_Area2),
  Tot_Cost is (Num_Units * Cost),
  add_material(Paint,Num_Units,Tot_Cost),fail.

add_material(Material,Num_Units,Tot_Cost) :-
  retract(material_list(Material,Old_Num_Units,Old_Cost)),
  New_Num_Units is (Old_Num_Units + Num_Units),
  New_Cost is (Old_Cost + Tot_Cost),
  assertz(material_list(Material,New_Num_Units,New_Cost)),!.

add_material(Material,Num_Units,Tot_Cost) :-
  assertz(material_list(Material,Num_Units,Tot_Cost)),!.

/*****************************************************/

/* material data */

material(shingle12,shingle,12,inches,6,inches,0.25,inches,0,feet,0,feet,1.25).

material(tar_paper2,tar_paper,72,inches,240,inches,0.25,inches,0,feet,0,
  feet,125.00).

material(tar_paper1,tar_paper,72,inches,240,inches,0.25,inches,0,feet,0,
  feet,150.00).

material(tar_paper3,tar_paper,72,inches,240,inches,0.25,inches,0,feet,0,
  feet,110.00).

material(sheath_paper24,sheath_paper,12,feet,100,feet,0.1,inches,
        0,feet,0,feet,75.65).

material(wood8,wood,144,inches,4,inches,2,inches,0,feet,0,feet,8.25).
```

```
material(hard_wood9,hard_wood,4,inches,24,feet,0.5,inches,
        0,feet,0,feet,12.00).

material(hardboard32,hardboard,36,feet,10,feet,1,inches,0,feet,0,feet,136.55).
material(hardboard78,hardboard,36,feet,24,feet,1,inches,0,feet,0,feet,289.00).
material(hardboard34,hardboard,24,feet,10,feet,1,inches,0,feet,0,feet,95.35).

/* use brick 10x4x6 effective size */
material(brick88,brick,10,inches,4,inches,6,inches,
        0,feet,0,feet,1.15).

liquid(paint9,paint,900,feet,1,gallon,8.00).
liquid(paint21,paint,700,feet,1,gallon,13.55).
liquid(paint17,paint,1100,feet,1,gallon,8.25).

/* 10 lb per 2 cubic feet */
filler(concrete1,concrete,2,feet,10,lb,5.00).

material(door1,_,_,_,_,_,_,_,0,feet,0,feet,16.00).

material(window1,_,_,_,_,_,_,_,0,feet,0,feet,30.50).
```

```
/* House1 File */

/*************************************************/

/* house data */

is_a(house1,house).

property(house1,subtype,single_room).

contains(house1,[roof1,exterior1,room1]).

/*************************************************/

/* exterior data */

is_a(exterior1,exterior).

contains(exterior1,[face5,face6,face7,face8]).
part_of(exterior1,house1).

/*************************************************/

/* roof data */

is_a(roof1,roof).

contains(roof1,[face11,face12]).
part_of(roof1,house1).

/*************************************************/

is_a(face11,face).

dimension(face11,height,151.5,inches).
dimension(face11,width,384,inches).
dimension(face11,depth,6.5,inches).

contains(face11,[frame1,sub_cover2,sub_cover1,cover1]).
normal_X(face11,0).
normal_Y(face11,0.34).
normal_Z(face11,0.94).
part_of(face11,roof1).
```

```
/*-----------------------------------------------*/

is_a(frame1,frame).

property(frame1,material_type,wood8).

dimension(frame1,height,139.5,inches).
dimension(frame1,width,382,inches).
dimension(frame1,depth,4,inches).

face(frame1,face11).

part_of(frame1,face11).

/*-----------------------------------------------*/

is_a(sub_cover2,sub_cover).

property(sub_cover2,material_type,wood8).

dimension(sub_cover2,depth,2,inches).

part_of(sub_cover2,face11).

/*-----------------------------------------------*/

is_a(sub_cover1,sub_cover).

property(sub_cover1,material_type,tar_paper2).

dimension(sub_cover1,depth,0.25,inches).

part_of(sub_cover1,face11).

/*-----------------------------------------------*/

is_a(cover1,cover).

property(cover1,material_type,shingle12).
property(cover1,finish_color,brown).

dimension(cover1,depth,0.25,inches).

part_of(cover1,face11).
```

```
/*************************************************/

is_a(face12,face).

dimension(face12,height,151.5,inches).
dimension(face12,width,384,inches).
dimension(face12,depth,6.5,inches).

contains(face12,[frame2,sub_cover13,sub_cover14,cover12]).
normal_X(face12,0).
normal_Y(face12,-0.34).
normal_Z(face12,0.94).
part_of(face12,roof1).

/*-----------------------------------------------*/

is_a(frame2,frame).

property(frame2,material_type,wood8).

dimension(frame2,height,139.5,inches).
dimension(frame2,width,382,inches).
dimension(frame2,depth,4,inches).

face(frame2,face12).

part_of(frame2,face12).

/*-----------------------------------------------*/

is_a(sub_cover13,sub_cover).

property(sub_cover13,material_type,wood8).

dimension(sub_cover13,depth,2,inches).

part_of(sub_cover13,face12).

/*-----------------------------------------------*/

is_a(sub_cover14,sub_cover).

property(sub_cover14,material_type,tar_paper2).

dimension(sub_cover14,depth,0.25,inches).

part_of(sub_cover14,face12).
```

```
/*-----------------------------------------------*/

is_a(cover12,cover).

property(cover12,material_type,shingle12).
property(cover12,finish_color,brown).

dimension(cover12,depth,0.25,inches).

part_of(cover12,face12).

/***********************************************/

/* room1 */

is_a(room1,room).

coordinates_X(product,room1,0,inches).
coordinates_Y(product,room1,0,inches).
coordinates_Z(product,room1,12,inches).
contains(room1,[face1,face2,face3,face4,face9,face10]).
part_of(room1,house1).

/***********************************************/

/* face1 */

is_a(face1,face).

dimension(face1,height,115,inches).
dimension(face1,width,302,inches).
dimension(face1,depth,1,inches).

contains(face1,[sub_cover3,cover2]).
normal_X(face1,0).
normal_Y(face1,-1).
normal_Z(face1,0).
part_of(face1,room1).

/*-----------------------------------------------*/

is_a(sub_cover3,sub_cover).

property(sub_cover3,material_type,hardboard32).

dimension(sub_cover3,depth,1,inches).

part_of(sub_cover3,face1).
```

```
/*-----------------------------------------------*/

is_a(cover2,cover).

property(cover2,material_type,paint9).
property(cover2,finish_color,yellow).

part_of(cover2,face1).

/***********************************************/

/* face2 */

is_a(face2,face).

dimension(face2,height,115,inches).
dimension(face2,width,240,inches).
dimension(face2,depth,1,inches).

contains(face2,[sub_cover4,cover3]).
normal_X(face2,-1).
normal_Y(face2,0).
normal_Z(face2,0).
part_of(face2,room1).

/*-----------------------------------------------*/

is_a(sub_cover4,sub_cover).

property(sub_cover4,material_type,hardboard34).

dimension(sub_cover4,depth,1,inches).

part_of(sub_cover4,face2).

/*-----------------------------------------------*/

is_a(cover3,cover).

property(cover3,material_type,paint9).
property(cover3,finish_color,yellow).

part_of(cover3,face2).
```

```
/*************************************************/

/* face3 */

is_a(face3,face).

dimension(face3,height,115,inches).
dimension(face3,width,362,inches).
dimension(face3,depth,1,inches).

contains(face3,[sub_cover5,cover4]).
normal_X(face3,0).
normal_Y(face3,1).
normal_Z(face3,0).
part_of(face3,room1).

/*-----------------------------------------------*/

is_a(sub_cover5,sub_cover).

property(sub_cover5,material_type,hardboard32).

dimension(sub_cover5,depth,1,inches).

part_of(sub_cover5,face3).

/*-----------------------------------------------*/

is_a(cover4,cover).

property(cover4,material_type,paint9).
property(cover4,finish_color,yellow).

part_of(cover4,face3).

/*************************************************/

/* face4 */

is_a(face4,face).

dimension(face4,height,115,inches).
dimension(face4,width,240,inches).
dimension(face4,depth,1,inches).

contains(face4,[sub_cover6,cover51]).
normal_X(face4,1).
normal_Y(face4,0).
normal_Z(face4,0).
part_of(face4,room1).
```

```
/*-----------------------------------------------*/

is_a(sub_cover6,sub_cover).

property(sub_cover6,material_type,hardboard34).

dimension(sub_cover6,depth,1,inches).

part_of(sub_cover6,face4).

/*-----------------------------------------------*/

is_a(cover5,cover).

property(cover5,material_type,paint9).
property(cover5,finish_color,yellow).

part_of(cover5,face4).

/*****************************************/

/* face5 */
/* use brick 10x4x6 effective size */

is_a(face5,face).

dimension(face5,height,120,inches).
dimension(face5,width,382,inches).
dimension(face5,depth,6,inches).

contains(face5,[frame3,sub_cover7,cover6]).
normal_X(face5,0).
normal_Y(face5,1).
normal_Z(face5,0).
part_of(face5,exterior1).

/*-----------------------------------------------*/

is_a(frame3,frame).

property(frame3,material_type,wood8).

dimension(frame3,depth,4,inches).

face(frame3,face5).
face(frame3,face1).

part_of(frame3,face5).
```

```
/*-----------------------------------------------*/

is_a(sub_cover7,sub_cover).

property(sub_cover7,material_type,sheath_paper24).

part_of(sub_cover7,face5).

/*-----------------------------------------------*/

is_a(cover6,cover).

property(cover6,material_type,brick88).
property(cover6,finish_color,red).

dimension(cover6,depth,6,inches).

part_of(cover6,face5).

/***********************************************/

/* face6 */

is_a(face6,face).

dimension(face6,height,120,inches).
dimension(face6,width,250,inches).
dimension(face6,depth,6,inches).

contains(face6,[frame4,sub_cover8,cover7,window11]).
normal_X(face6,1).
normal_Y(face6,0).
normal_Z(face6,0).
part_of(face6,exterior1).

/*-----------------------------------------------*/

is_a(frame4,frame).

property(frame4,material_type,wood8).

dimension(frame4,depth,4,inches).

face(frame4,face6).
face(frame4,face2).

part_of(frame4,face6).
```

```
/*-----------------------------------------------*/

is_a(sub_cover8,sub_cover).

property(sub_cover8,material_type,sheath_paper24).

part_of(sub_cover8,face6).

/*-----------------------------------------------*/

is_a(cover7,cover).

property(cover7,material_type,brick88).
property(cover7,finish_color,red).

dimension(cover7,depth,6,inches).

part_of(cover7,face6).

/*-----------------------------------------------*/

is_a(window1,window).

dimension(window1,height,36,inches).
dimension(window1,width,48,inches).
dimension(window1,depth,0.5,inches).

contains(window1,[pane1,sill1,case11).
face(window1,face2).
face(window1,face6).
coordinates_X(local,window1,96,inches).
coordinates_Y(local,window1,0,inches).
coordinates_Z(local,window1,66,inches).
part_of(window1,face6).

/*-----------------------------------------------*/

is_a(pane1,pane).

property(pane1,quality,4).

part_of(pane1,window1).

/*-----------------------------------------------*/

is_a(sill1,sill).

property(sill1,finish_type,paint17).
property(sill1,finish_color,white).

part_of(sill1,window1).
```

```
/*-----------------------------------------------*/

is_a(case1,case).

part_of(case1,window1).

/***********************************************/

/* face7 */

is_a(face7,face).

dimension(face7,height,120,inches).
dimension(face7,width,382,inches).
dimension(face7,depth,6,inches).

contains(face7,[frame5,sub_cover9,cover8,door11]).
normal_X(face7,0).
normal_Y(face7,-1).
normal_Z(face7,0).
part_of(face7,exterior1).

/*-----------------------------------------------*/

is_a(frame5,frame).

property(frame5,material_type,wood8).

dimension(frame5,depth,4,inches).

face(frame5,face7).
face(frame5,face3).

part_of(frame5,face7).

/*-----------------------------------------------*/

is_a(sub_cover9,sub_cover).

property(sub_cover9,material_type,sheath_paper24).

part_of(sub_cover9,face7).
```

```
/*-----------------------------------------------*/

is_a(cover8,cover).

property(cover8,material_type,brick88).
property(cover8,finish_color,red).

dimension(cover8,depth,6,inches).

part_of(cover8,face7).

/*-----------------------------------------------*/

is_a(door1,door).

property(door1,material_type,wood5).
property(door1,finish_type,paint21).
property(door1,finish_color,brown).
property(door1,knob_type,round32).
property(door1,hinge_type,square3in).

dimension(door1,height,84,inches).
dimension(door1,width,36,inches).
dimension(door1,depth,2.5,inches).

face(door1,face3).
face(door1,face7).
coordinates_X(local,door1,125,inches).
coordinates_Y(local,door1,0,inches).
coordinates_Z(local,door1,42,inches).
part_of(door1,face7).

/*************************************************/

/* face8 */

is_a(face8,face).

dimension(face8,height,120,inches).
dimension(face8,width,250,inches).
dimension(face8,depth,6,inches).

contains(face8,[frame6,sub_cover10,cover91).
normal_X(face8,-1).
normal_Y(face8,0).
normal_Z(face8,0).
part_of(face8,exterior1).
```

```
/*-----------------------------------------------*/

is_a(frame6,frame).

property(frame6,material_type,wood8).

dimension(frame6,depth,4,inches).

face(frame6,face8).
face(frame6,face4).

part_of(frame6,face8).

/*-----------------------------------------------*/

is_a(sub_cover10,sub_cover).

property(sub_cover10,material_type,sheath_paper24).

part_of(sub_cover10,face8).

/*-----------------------------------------------*/

is_a(cover9,cover).

property(cover9,material_type,brick88).
property(cover9,finish_color,red).

part_of(cover9,face8).

/*************************************************/

/* face9 */

is_a(face9,face).

dimension(face9,height,20,feet).
dimension(face9,width,30,feet).
dimension(face9,depth,1,inches).

contains(face9,[frame7,sub_cover11,cover10]).
normal_X(face9,0).
normal_Y(face9,0).
normal_Z(face9,-1).
part_of(face9,room1).
```

```
/*-----------------------------------------------*/

is_a(frame7,frame).

property(frame7,material_type,wood8).

face(frame7,face9).

part_of(frame7,face9).

/*-----------------------------------------------*/

is_a(sub_cover11,sub_cover).

property(sub_cover11,material_type,hardboard78).

dimension(sub_cover11,depth,1,inches).

part_of(sub_cover11,face9).

/*-----------------------------------------------*/

is_a(cover10,cover).

property(cover10,material_type,paint17).
property(cover10,finish_color,white).

part_of(cover10,face9).

/***********************************************/

/* face10 */

is_a(face10,face).

dimension(face10,height,382,inches).
dimension(face10,width,262,inches).
dimension(face10,depth,12.5,inches).

contains(face10,[frame8,sub_cover12,cover11]).
normal_X(face10,0).
normal_Y(face10,0).
normal_Z(face10,1).
part_of(face10,room1).
```

```
/*-----------------------------------------------*/

is_a(frame8,frame).

property(frame8,material_type,concrete1).

dimension(frame8,depth,12,inches).

face(frame8,face10).

part_of(frame8,face10).

/*-----------------------------------------------*/

is_a(sub_cover12,sub_cover).

property(sub_cover12,material_type,hard_wood9).

dimension(sub_cover12,height,20,feet).
dimension(sub_cover12,width,30,feet).
dimension(sub_cover12,depth,0.5,inches).

part_of(sub_cover12,face10).

/*-----------------------------------------------*/

is_a(cover11,cover).

property(cover11,material_type,paint21).
property(cover11,finish_color,brown).

dimension(cover11,height,20,feet).
dimension(cover11,width,30,feet).

part_of(cover11,face10).
```

```
/* Schema File */

part_of(house,floorplan).
part_of(house,exterior).
part_of(house,room).
part_of(house,roof).
part_of(house,space).

part_of(roof,face).

part_of(room,face).

part_of(space,face).

part_of(exterior,face).

part_of(face,door).
part_of(face,window).
part_of(face,opening).
part_of(face,covering).
part_of(face,sub_covering).
part_of(face,frame).
part_of(face,insulation).
part_of(face,connection).

part_of(connection,plumbing).
part_of(connection,electric).
part_of(connection,heating).
part_of(connection,gas).

part_of(window,sill).
part_of(window,case).
part_of(window,pane).

trans_partof(X,Y) :- part_of(X,Y),!.
trans_partof(X,Y) :- part_of(X,Z),

trans_partof(Z,Y),!.
```

```
/* Conversion File */

converts(A,feet,B,feet) :- B = A.
converts(A,inches,B,inches) :- B = A.
converts(A,feet,B,inches) :- B = A * 12.
converts(A,inches,B,feet) :- B = A / 12.
converts(A,feet,B,yards) :- B = A / 3.
converts(A,yards,B,feet) :- B = A * 3.

convert(A,Dimension1,B,Dimension2) :-
  converts(A,Dimension1,B,Dimension2),!.

convert(A,Dimension1,B,Dimension2) :-
  converts(A,Dimension1,X,Dimensionx),
  not(equal(Dimension1,Dimensionx)),
  convert(X,Dimensionx,B,Dimension2).
```

```
/* Routines File */

/* find longest dimension of three passed in */
longest_dimension(Ht,Htunits,Wd,Wdunits,Dp,Dpunits,Len,Htunits) :-
  convert(Wd,Wdunits,New_Wd,Htunits),
  convert(Dp,Dpunits,New_Dp,Htunits),
  maximum(Ht,New_Wd,Max),
  maximum(Max,New_Dp,Len),!.

maximum(A,B,A) :-
  A > B,!.

maximum(A,B,B).

/* have match if within .25 inches */
match(A,A_Units,B,B_Units,C,C_Units,D,D_Units,A,A_Units,B,B_Units) :-
  convert(A,A_Units,New_A,inches),
  convert(B,B_Units,New_B,inches),
  convert(C,C_Units,New_C,inches),
  convert(D,D_Units,New_D,inches),
  ((New_D - New_C) < 0.25),
  ((New_D - New_C) > - 0.25),!.

match(A,A_Units,B,B_Units,C,C_Units,D,D_Units,A,A_Units,C,C_Units) :-
  convert(A,A_Units,New_A,inches),
  convert(B,B_Units,New_B,inches),
  convert(C,C_Units,New_C,inches),
  convert(D,D_Units,New_D,inches),
  ((New_D - New_B) < 0.25),
  ((New_D - New_B) > - 0.25),!.

match(A,A_Units,B,B_Units,C,C_Units,D,D_Units,B,B_Units,C,C_Units) :-
  convert(A,A_Units,New_A,inches),
  convert(B,B_Units,New_B,inches),
  convert(C,C_Units,New_C,inches),
  convert(D,D_Units,New_D,inches),
  ((New_D - New_A) < 0.25),
  ((New_D - New_A) > - 0.25),!.

match(A,A_Units,B,B_Units,C,C_Units,D,D_Units,A,A_Units,B,B_Units) :-
  nl,write('Error! No match found during raw material calculations.'),fail.

/* routine to get member of list */
member(X,[X|L]).
member(X,[Y|L]) :- member(X,L).
```

```
/* routine to delete member of list */
delete(X,[],[]).
delete(X,[X|L],L) :- !.
delete(X,[Y|L],[Y|M]) :- delete(X,L,M).

equal(A,B) :- B = A.
```

APPENDIX D

```
| ?- start.
check for house house1

check for exterior exterior1

check for roof roof1

check for face face11

check for frame frame1

    grade marks must be clearly visible on all framing
    members for inspection

check for sub_cover sub_cover2

check for sub_cover sub_cover1

    sub_cover sub_cover1 meets requirements; allowed substitutes are:

    - tar_paper1

    - tar_paper3

check for cover cover1

check for face face12

check for frame frame2

    grade marks must be clearly visible on all framing
    members for inspection

check for sub_cover sub_cover13

check for sub_cover sub_cover14

    sub_cover sub_cover14 meets requirements; allowed substitutes are:

    - tar_paper1

    - tar_paper3

check for cover cover12
```

```
check for room room1

check for face face1

check for sub_cover sub_cover3

check for cover cover2

check for face face2

check for sub_cover sub_cover4

check for cover cover3

check for face face3

check for sub_cover sub_cover5

check for cover cover4

check for face face4

check for sub_cover sub_cover6

check for cover cover5

check for face face5

check for frame frame3

    grade marks must be clearly visible on all framing
    members for inspection

check for sub_cover sub_cover7

check for cover cover6

    approved methods must be used for building masonry walls
    when outside air temperature drops below 40 degrees farenheit

check for face face6

check for frame frame4

    grade marks must be clearly visible on all framing
    members for inspection

check for sub_cover sub_cover8
```

```
check for cover cover7

    approved methods must be used for building masonry walls
    when outside air temperature drops below 40 degrees farenheit

check for window window1

check for pane pane1

    pane pane1 passed quality check

check for sill sill1

check for case case1

check for face face7

check for frame frame5

    grade marks must be clearly visible on all framing
    members for inspection

check for sub_cover sub_cover9

check for cover cover8

    approved methods must be used for building masonry walls
    when outside air temperature drops below 40 degrees farenheit

check for door door1

    door door1 passed - height

    door door1 passed - width

    door door1 passed - depth

check for face face8

check for frame frame6

    grade marks must be clearly visible on all framing
    members for inspection

check for sub_cover sub_cover10

check for cover cover9

    approved methods must be used for building masonry walls
    when outside air temperature drops below 40 degrees farenheit
```

```
check for face face9

check for frame frame7

    grade marks must be clearly visible on all framing
    members for inspection

check for sub_cover sub_cover11

check for cover cover10

check for face face10

check for frame frame8

check for sub_cover sub_cover12

check for cover cover11
```

```
*********************************************
*                                           *
 Production Sequence Report for house1

  - house style is single_room
    and consists of [roof1,exterior1,room1]
*                                           *
*********************************************


*********************************************
*                                           *
 comment : normal for each face listed
*                                           *
*********************************************

FACE          X             Y             Z
____          _             _             _
face11        0             0.34          0.94
face12        0             -0.34         0.94
face1         0             -1            0
face2         -1            0             0
face3         0             1             0
face4         1             0             0
face5         0             1             0
face6         1             0             0
face7         0             -1            0
face8         -1            0             0
face9         0             0             -1
face10        0             0             1

*********************************************
*                                           *
 comment : erect foundation and frame
*                                           *
*********************************************

frame8        assemble      material type:   concrete1
frame4        assemble      material type:   wood8
frame6        assemble      material type:   wood8
frame3        assemble      material type:   wood8
frame5        assemble      material type:   wood8
frame7        assemble      material type:   wood8
frame1        assemble      material type:   wood8
frame2        assemble      material type:   wood8
```

```
*******************************************
*                                         *
 comment : put door framing in place
*                                         *
*******************************************

door1           assemble        material type:   wood5
                - attach to:    face3            face7
                - location      relative to      face7
                 X coordinate   125              inches
                 Y coordinate   0                inches
                 Z coordinate   42               inches

*******************************************
*                                         *
 comment : put window framing in place
*                                         *
*******************************************

sill1           assemble        window sill for: window1
                - attach to:    face2            face6
                - location      relative to      face6
                 X coordinate   96               inches
                 Y coordinate   0                inches
                 Z coordinate   66               inches

*******************************************
*                                         *
 comment : put up exterior siding
*                                         *
*******************************************

sub_cover10     assemble        material type:   sheath_paper24
sub_cover9      assemble        material type:   sheath_paper24
sub_cover8      assemble        material type:   sheath_paper24
sub_cover7      assemble        material type:   sheath_paper24
cover6          assemble        material type:   brick88
cover7          assemble        material type:   brick88
cover8          assemble        material type:   brick88
cover9          assemble        material type:   brick88
```

```
*********************************************
*                                           *
 comment : put up roof
*                                           *
*********************************************

sub_cover13     assemble        material type:   wood8
sub_cover2      assemble        material type:   wood8
sub_cover1      assemble        material type:   tar_paper2
sub_cover14     assemble        material type:   tar_paper2
cover12         assemble        material type:   shingle12
cover1          assemble        material type:   shingle12

*********************************************
*                                           *
 comment : put up faces for each room
*                                           *
*********************************************

sub_cover11     assemble        material type:   hardboard78
sub_cover6      assemble        material type:   hardboard34
sub_cover4      assemble        material type:   hardboard34
sub_cover5      assemble        material type:   hardboard32
sub_cover3      assemble        material type:   hardboard32

*********************************************
*                                           *
 comment : build floor as last step
*                                           *
*********************************************

sub_cover12     assemble        material type:   hard_wood9

*********************************************
*                                           *
 comment : put windows in place
*                                           *
*********************************************

window1         complete using pane1            case1

*********************************************
*                                           *
 comment : put finish on windows and doors
*                                           *
*********************************************

sill1           finish          paint17         white
door1           finish          paint21         brown
```

```
*********************************************
*                                           *
 comment : put on door knobs and hinges
*                                           *
*********************************************

door1          assemble       knob            round32
door1          assemble       hinge           square3in

*********************************************
*                                           *
 comment : put final paint on faces
*                                           *
*********************************************

cover10        paint          material type:  paint17
cover3         paint          material type:  paint9
cover5         paint          material type:  paint9
cover2         paint          material type:  paint9
cover4         paint          material type:  paint9
cover11        paint          material type:  paint21
```

Raw Materials Report

Item	Cost	Units Required
door1	$16	1
window1	$30	1
concrete1	$1737	347.514
wood8	$3582	434.194
tar_paper2	$841	6.73333
hardboard32	$211	1.54776
hardboard34	$147	1.54722
hardboard78	$200	0.694444
hard_wood9	$900	75
sheath_paper24	$64	0.850277
shingle12	$2020	1616
brick88	$4224	3673.2
paint9	$8	1.0317
paint17	$4	0.551818
paint21	$12	0.923095

```
**************************************
*                                    *
 Total material cost is $13996
*                                    *
**************************************
```

Start Raw Materials Report (w/ substitute)

```
**********************************************************
*                                                        *
 sub_cover1: substitute tar_paper1 for tar_paper2
*                                                        *
**********************************************************
```

Raw Materials Report

Item	Cost	Units Required
door1	$16	1
window1	$30	1
concrete1	$1737	347.514
tar_paper1	$504	3.36666
wood8	$3582	434.194
tar_paper2	$420	3.36666
hardboard32	$211	1.54776
hardboard34	$147	1.54722
hardboard78	$200	0.694444
hard_wood9	$900	75
sheath_paper24	$64	0.850277
shingle12	$2020	1616
brick88	$4224	3673.2
paint9	$8	1.0317
paint17	$4	0.551818
paint21	$12	0.923095

```
************************************
*                                  *
 Total material cost is $14079
*                                  *
************************************
```

```
*****************************************************
*                                                   *
 sub_cover1: substitute tar_paper3 for tar_paper2
*                                                   *
*****************************************************
```

Raw Materials Report

Item	Cost	Units Required
door1	$16	1
window1	$30	1
concrete1	$1737	347.514
tar_paper3	$370	3.36666
wood8	$3582	434.194
tar_paper2	$420	3.36666
hardboard32	$211	1.54776
hardboard34	$147	1.54722
hardboard78	$200	0.694444
hard_wood9	$900	75
sheath_paper24	$64	0.850277
shingle12	$2020	1616
brick88	$4224	3673.2
paint9	$8	1.0317
paint17	$4	0.551818
paint21	$12	0.923095

```
************************************
*                                  *
 Total material cost is $13945
*                                  *
************************************
```

```
*********************************************************
*                                                       *
 sub_cover14: substitute tar_paper1 for tar_paper2
*                                                       *
*********************************************************
```

Raw Materials Report

Item	Cost	Units Required
door1	$16	1
window1	$30	1
concrete1	$1737	347.514
tar_paper2	$420	3.36666
wood8	$3582	434.194
tar_paper1	$504	3.36666
hardboard32	$211	1.54776
hardboard34	$147	1.54722
hardboard78	$200	0.694444
hard_wood9	$900	75
sheath_paper24	$64	0.850277
shingle12	$2020	1616
brick88	$4224	3673.2
paint9	$8	1.0317
paint17	$4	0.551818
paint21	$12	0.923095

```
************************************
*                                  *
 Total material cost is $14079
*                                  *
************************************
```

```
**********************************************************
*                                                        *
 sub_cover14: substitute tar_paper3 for tar_paper2
*                                                        *
**********************************************************
```

Raw Materials Report

Item	Cost	Units Required
door1	$16	1
window1	$30	1
concrete1	$1737	347.514
tar_paper2	$420	3.36666
wood8	$3582	434.194
tar_paper3	$370	3.36666
hardboard32	$211	1.54776
hardboard34	$147	1.54722
hardboard78	$200	0.694444
hard_wood9	$900	75
sheath_paper24	$64	0.850277
shingle12	$2020	1616
brick88	$4224	3673.2
paint9	$8	1.0317
paint17	$4	0.551818
paint21	$12	0.923095

```
**************************************
*                                    *
 Total material cost is $13945
*                                    *
**************************************
```

LIST OF REFERENCES

1. Ness, Philip H., "Managing Integrated CAD/CAM in a Large Aerospace Company for Fun and Profit," Automation Technology for Management and Productivity Advancements through CAD/CAM and Engineering Data Handling, p. 181, Printice-Hall, 1983.

2. Beeby, William D., "Applications of Computer Aided Design on the 767," Automation Technology for Management and Productivity Advancements through CAD/CAM and Engineering Data Handling, p. 181, Printice-Hall, 1983.

3. Su, S. Y. W., and others, "The Architecture and Prototype Implementation of an Integrated Manufacturing Database Administration System," IEEE Computer Society International Conference, pp. 287-289, Computer Science Press, 1986.

4. Abi-Ezzi, Salim S., and Kader, Steven E., "Phigs in CAD," Computers in Mechanical Engineering, v. 5, pp. 28-31, July 1986.

5. Madison, Dana E., and Wu, C. Thomas, An Expert System Interface and Data Requirements for the Integrated Product Design and Manufacturing Process, report prepared for Chief of Naval Research, Naval Postgraduate School, Monterey, California, June 9, 1986.

6. Conaway, Jack, "What's in a Name: Plain Talk About CIM," Computers in Mechanical Engineering, v. 4, pp. 23-31, November 1985.

7. Rutherford, S. L., "Examples of Automated Design Data Used in Manufacturing Processes," Automation Technology for Management and Productivity Advancements through CAD/CAM and Engineering Data Handling, pp. 234-238, Printice-Hall,1983.

8. Su, S. Y. W., "Modeling Integrated Manufacturing Data With SAM," IEEE Computer, pp. 34-49, Computer Science Press, January 1986.

9. Rowe, N. C., AI through Prolog, chapter 12, Printice_Hall, 1987.

10. Khokhani, Kanti, and others, "Advanced Algorithms Enhance Board Layout," Digital Design, v. 16, pp. 56-60, June 1986.

11. Winston, Patrick Henry, Artificial Intelligence, pp. 87-132, Addison-Wesley, 1984.

12. Hayes-Roth, F., Lenat, D. B., and Waterman, D. A., Building Expert Systems, pp. 219-235, Addison-Wesley, 1983.

13. Porter, Ed, "AI Emerges as Design Automation Force," Digital Design, v. 16, pp. 48-52, June 1986.

14. Davis, H. F. and Snider, A. D., Introduction to Vector Analysis, pp. 1-11, Allyn and Bacon, Inc., 1975.

Initial Distribution List

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 142 Naval Postgraduate School Monterey, California 93943-5002	2
3.	MAJ Dana E. Madison Department of Computer Science, Code 52Wq Naval Postgraduate School Monterey, California 93943-5000	1
4.	Associate Professor C. Thomas Wu Department of Computer Science, Code 52Wq Naval Postgraduate School Monterey, California 93943-5000	12

18070 2

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA 93943-5002

Thesis
W58507 Wilbur
c.1 Expert database support of the integrated manufacturing system.

6 OCT 89 35757

Thesis
W58507 Wilbur
c.1 Expert database support of the integrated manufacturing system.

DUDLEY KNOX LIBRARY
3 2768 00305679 7