NPS52-86-015

NAVAL POSTGRADUATE SCHOOL

Monterey, California

An Expert System Interface and Data Requirements
for the
Integrated Product Design and Manufacturing Process

MAJ Dana E. Madison

and

C. Thomas Wu

June 9, 1986

Prepared for:

Chief of Naval Research
Arlington, VA 22217

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Rear Admiral R. H. Shumaker
Superintendent

D. A. Schrady
Provost

The work reported herein was supported in part by the Foundation Research Program of the Naval Postgraduate School with funds provided by the Chief of Naval Research.

This report was prepared by:

C. Thomas Wu
Associate Professor
Computer Science

Reviewed by:

VINCENT Y. LUM
Chairman
Department of Computer Science

Released by:

KNEALE T. MARSHALL
Dean of Information and
Policy Science

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER NPS52-86-015	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* An Expert System Interface and Data Requirements for the Integrated Product Design and Manufacturing Process		5. TYPE OF REPORT & PERIOD COVERED 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) MAJ Dana E. Madison C. Thomas Wu		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, CA 93943-5100		10. PROGRAM ELEMENT, PROJECT, TASK & WORK UNIT NUMBERS 61152N; RR000-01 N0001486WR4E001
11. CONTROLLING OFFICE NAME AND ADDRESS Chief of Naval Research Arlington, VA 22217		12. REPORT DATE June 1986 13. NUMBER OF PAGES 35
14. MONITORING AGENCY NAME & ADDRESS*(if different from Controlling Office)*		15. SECURITY CLASS. *(of this report)* 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT *(of this Report)* Approved for public release; distribution unlimited		
17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*		

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

Database technology has been successfully applied to the traditional data processing environment where data are represented by well-formatted records. There is a growing interest in extending this database technology to more advanced application environments such as VLSI CAD/CAM, cartography, etc., where data are less structured and have very complex semantics. In this paper, we describe the data interactions in the design and manufacturing phases which are necessary to integrate the two phases automatically. These data requirements are part of an integrated information support system geared towards the "make to

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

order" design and manufacturing process. An expert system translator to carry out the integration is described and demonstrated in an example. An overall goal of our research is to develop a completely integrated information support system for generic design and manufacturing processes.

An Expert System Interface and Data Requirements for the Integrated Product Design and Manufacturing Process

MAJ Dana E. Madison

and

C. Thomas Wu

DEPARTMENT OF COMPUTER SCIENCE
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CA 93943

June 9, 1986

Abstract

Database technology has been successfully applied to the traditional data processing environment where data are represented by well-formatted records. There is a growing interest in extending this database technology to more advanced application environments such as VLSI CAD/CAM, cartography, etc., where data are less structured and have very complex semantics. In this paper, we describe the data interactions in the design and manufacturing phases which are necessary to integrate the two phases automatically. These data requirements are part of an integrated information support system geared towards the "make to order" design and manufacturing process. An expert system translator to carry out the integration is described and demonstrated in an example. An overall goal of our research is to develop a completely integrated information support system for generic design and manufacturing processes.

1. Introduction

One of the current trends in database research involves supporting more advanced engineering environments such as VLSI design. Considerable effort has been expended in developing specific data models to support the VLSI CAD process [BAT085], [MCLE83]. There is a growing interest in expanding the use of database technology to support the generic product design and manufacturing process. One very important benefit of this support is the potential for reduction in errors. In particular, automation of component counts, physical dimension measurements, and routine calculations could lead to reduction of errors in cost estimates, raw material requirements, and ordering of components and raw materials. Another benefit is the ability to automatically maintain logical relationships between objects as a design is manipulated. For example, automated support could automatically adjust connecting walls, windows, and doors if a wall is moved. This type of support frees a designer from responsibility for detailed adjustments of a design to maintain consistency, producing significant reductions in design errors and inconsistencies. Significant advances have been made in the individual aspects of Computer-Aided Design (CAD) [BAT085], [MCLE83], [SU86a] and Computer-Aided Manufacturing (CAM) [SU86b], however, the integration of design information into the manufacturing phase of the product life cycle has been largely ignored because of a lack of standards for data integration between the two functions.

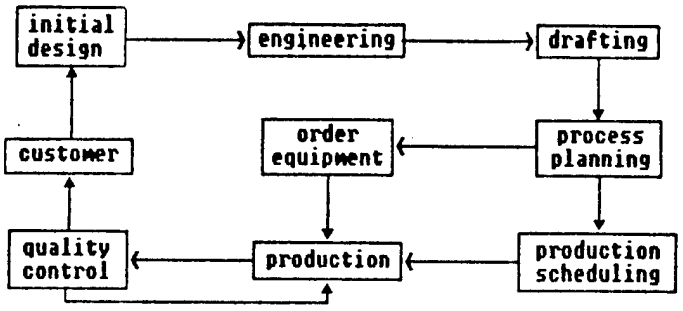
A major contribution of this paper is the description of the data requirements necessary for a fully integrated design and manufacturing system. The key to integrating the design and manufacturing phases lies in converting attribute data which is developed during the design process into information about the quantities and types of raw materials required in the manufacturing process. The portion of the manufacturing process which first uses this information about raw materials is the material requirements planning (MRP) phase, where raw materials are ordered and component parts production is planned. An expert system translator will be proposed which provides for the

integration of design information into the MRP phase of product manufacturing.

This paper is organized as follows: the product design and manufacturing process will be described followed by the data representation and integration requirements for MRP. The paper will conclude with a description of our proposed MRP translator and some directions for further research.

2. The Role of CAD/CAM in the Product Life Cycle

The role of CAD/CAM in the operations of a manufacturing company can be best portrayed by describing the various functions and activities involved in the design and manufacturing of a product. These functions and activities are known as the *product life cycle*. Figure 1 depicts the product life cycle as a series of activities, each interacting with one or more other activities in the cycle.



Product Life Cycle

figure 1

Input to the cycle consists of information about prospective markets and customers' desires, also known as the *demand* for the product. It is this demand which drives the decision-making process to determine in what ways the product life cycle will be activated and controlled. The extent of customer involvement in the design process will vary from product to product. In the case of house design, for example,

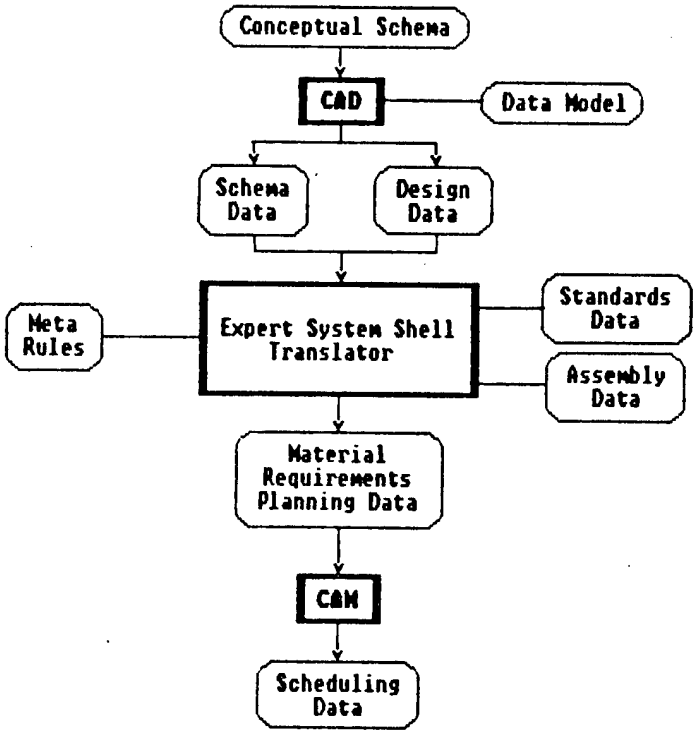
customers quite frequently supply specifications for a house, right down to the placement of wiring and plumbing runs within the walls. Other products, for example electronic products such as televisions, are designed by engineers working for the company manufacturing the product. Regardless of who actually does the design, the process begins with a concept or idea for a product. This concept is refined, analyzed, and improved by the design engineering process. The result of this process is a set of engineer drawings and specifications which detail how the product is to be made. At this point, the product moves from the design phase to the manufacturing phase of the life cycle.

The first activity in the manufacturing phase is the specification of the sequence of production operations necessary to make the product, known as the *process plan*. If new equipment and/or tools are required to make the product, they are purchased at this point in time. The process plan is used as input to the scheduling function, which attempts to satisfy the company's need to produce specific quantities of different products by specified dates. After the schedule is altered to include the new product, the product is put into production. Production and quality control perform their respective functions cyclically until the quality control standards are met or exceeded. At that point, the product is ready for delivery to the customer.

Recent advances in CAD/CAM have increased its use in the activities in the product life cycle. Computer-aided design, computer-aided drafting, and engineering documentation storage systems support the design phase, while most process planning and scheduling functions are automated to increase efficiency. Computers are used directly and indirectly to monitor and control the production operations and quality control functions in the manufacturing phase. CAD/CAM has traditionally supported the design and production activities as separate and distinct functions, and is now moving towards integration of the two using a technology known as *Computer Integrated Manufacturing (CIM)*.

3. Data Requirements for Computer Integrated Manufacturing

Webster's Dictionary defines "integrated" as *unified* or *united.* We maintain, therefore, that the "integrated" in Computer Integrated Manufacturing refers to the unification of the processes in the product life cycle through automation of the data interactions between these processes. Our use of the term Computer Integrated Manufacturing or CIM uses the word manufacturing in the broadest sense to mean the use of automation to support the entire product life cycle, not just the manufacturing phase of that cycle. Figure 2 depicts the data interactions in the life cycle from product design to the point where scheduling data is produced. Since our main objective in this paper is to provide an interface between the design data and the material requirements planning process, we will not concern ourselves with data requirements beyond the scheduling process.



Data Interaction in Computer Integrated Manufacturing

figure 2

The design phase is represented by the box labelled *CAD* which takes the conceptual schema as input and outputs schema and design data using the data model as a guiding mechanism. The *expert system shell translator* uses the schema and design data as input and produces material requirements planning data, which is used in the manufacturing phase, eventually being converted to scheduling data. We will discuss each of the data pools shown in figure 2 individually and tie them together by describing the interactions which occur.

3.1 Conceptual Schema

The conceptual schema will show the allowable type/subtype aggregations, component relationships, and the acceptable combinations of primitives to produce designs. Here, primitives for a particular product are defined. Primitives can be defined to any level of abstraction, and can be composite objects themselves. These primitives are the building blocks which the data model manipulates in the design process, therefore, the conceptual schema is product specific. A separate schema is produced for each different product line to be designed.

Each type and subtype shown in the conceptual schema will have a *prototype* associated with it. These prototypes will contain *slots* for attribute values, allow default values to be specified, and provide *inheritance* information. When instances are created, *extensions* of these prototypes are created, allowing for attribute values to be defined which are unique to that instance.

Figure 3 provides an example of a conceptual schema. This schema represents the hierarchy of type aggregations for a generic house. An instance of this schema would contain data for a specific house being designed.

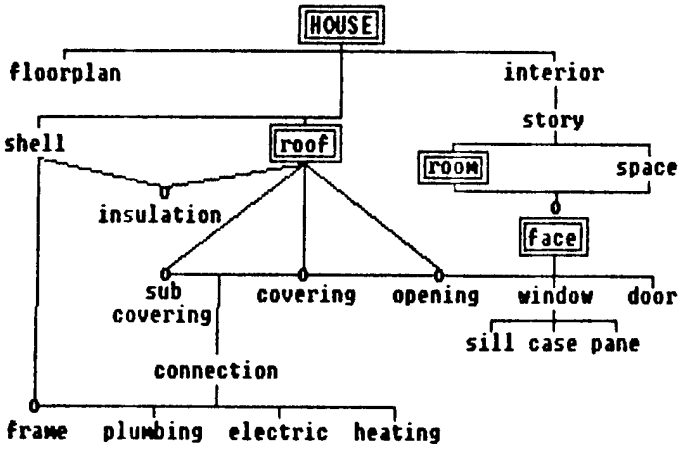


figure 3

House is the aggregation of a floor plan, a shell, a roof, and an interior. Each of shell, roof, and interior are further defined as aggregations of objects, some of which are shared. For example, both roof and shell have a component called "insulation".

The bubble notation in figure 3 represents an exclusive-or relationship among the types involved. A particular instance of insulation is either of type shell or roof, but not both. The insulation associated with the shell would be a separate instance and version from the insulation associated with the roof. The double rectangle notation represents types which have named subtypes. For example, room has subtypes named kitchen, den, bathroom, bedroom, etc., which can be instantiated to produce a specific configuration.

In summary, the conceptual schema provides the medium through which the data model captures the design data for a particular product. Together, the data model and conceptual schema determine the full range of design alternatives for a product.

3.2 The Role of the Data Model

In the traditional manual design process, data describing a design is placed on paper in the form of drawings and specifications. Both are revised and developed to higher levels of detail, potentially producing redundant and sometimes incomplete data. The redundant data leads to maintenance and consistency problems. The data which is produced in one design has little chance of being used in subsequent designs due to its manual nature. These problems and lack of reusable data prompted CAD developers to establish increased productivity as their major objective in the transition to an automated system for design functions. This objective is particularly important in developing countries, where shortages of technically skilled engineers can not keep pace with construction demands. In this case, CAD can achieve optimal use of scarce labor resources. The automation of routine calculations, data processing, word processing, and drafting functions leads to substantial productivity increases as technically skilled engineers are able to devote more of their time to technical duties and less time to administrative functions. Additionally, the declining cost of computer support makes the investment for integration of design and production functions attractive. This integration starts with the definition of a data model which supports the design process and provides a framework that facilitates use of design data in the manufacturing process.

Traditional hierarchical, relational, and network data models are oriented toward manipulation of logical records and do not support the CIM design environment. These traditional models lack facilities for handling the semantics which are a component of the design process. There is considerable interest in expanding the use of database technology to support data which is less structured, less formatted, with more complex data types, which would permit modeling of application semantics. Semantic data models attempt to provide high-level data structuring features to improve the expressiveness of database conceptual schemas. This is done by embedding the semantics of a particular application in the database schema. The overall

objective of the semantic models is to increase database accessibility by end users, many of whom are not trained in computer science.

In addition to providing for the representation of these semantics, the ideal CIM data model would provide other features which are not found in the traditional models. One of these features is the representation of design objects as primitives in the model, with prescribed "rules" for associating objects with one another. These objects could be the building blocks from which more complex objects could be built. Operations defined for the data model would include those for manipulating objects. These operations would include provisions for adding new objects and modifying existing ones. In this paper we will develop a data model which includes these desirable features.

Much of the application emphasis of CIM work done to date has been in the VLSI design process [BAT085], [LEE83], [MCLE83]. We believe that the integrated Information System concept can be extended to more generic applications, including integrated product design and manufacturing.

We will identify the abstraction concepts supported by our data model which are necessary for the design and manufacturing processes, and the types of support that a truly integrated system should provide.

Current semantic models include the Entity-Relationship (ER) Model, Functional Model, SHM+, SDM/Event Model, TAXIS, SAM*, and RM/T. All of these models use primitives such as entities, events, or simply objects. They also include provisions for composite objects and attribute specification among the supported features. Extended semantic models integrate a number of programming language concepts with database concepts. They also make use of advanced data type concepts such as abstract data types and strong typing. These extended models include SHM+, TAXIS, and the SDM/Event Model. Semantic modeling theory is now being applied to particular application areas such as office automation, VLSI, and cartography, as well as for traditional

data processing applications (inventory, insurance, banking). We will make use of many of the concepts from current semantic models in the description of our model.

The abstraction concepts supported by our model include molecular aggregation, generalization/specialization, version generalization, version hierarchy, instantiation, and instance hierarchy. We believe these abstraction concepts are necessary to support the design process, and therefore are useful for other advanced application areas as well. Each concept will be described in the remainder of this section.

Molecular aggregation is the abstraction of a set of objects and their relationships into a higher-level object [SMIT77]. This abstraction allows a view of objects from different levels of generality, each with its own level of detailed definition. A user interested in the overall design could use the topmost level of abstraction, which would hide the implementation details. This implements the "Information Hiding" principle commonly found in programming language design. The idea is to give the user only the amount of implementation detail he needs for a particular application. Figure 4 depicts a bathroom as a molecular aggregation of objects called floor, wall, ceiling, sink, toilet, and bathtub. All of these except toilet are molecular objects. Note that several levels of molecular aggregation abstraction are present in the figure.

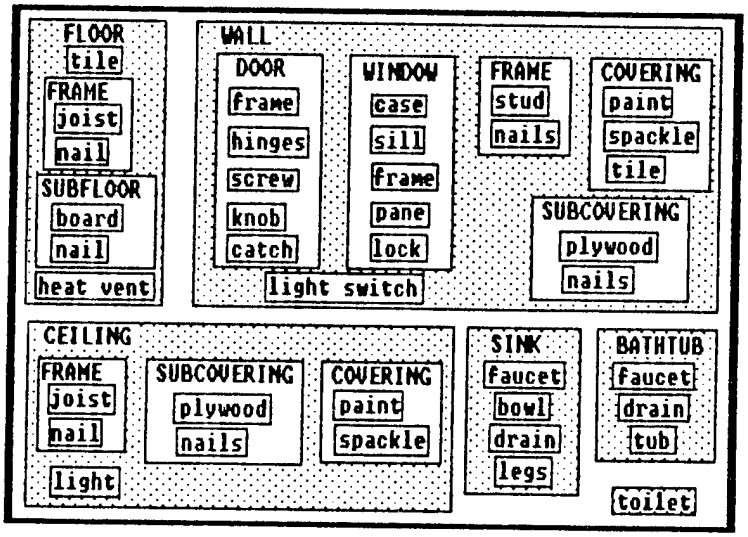


figure 4

The objects whose name appears in upper case are molecular aggregations. Those in lower case represent primitive objects in this example.

Molecular objects have two description components, an interface, and an implementation [BAT085]. The interface specifies the general function of the object and the implementation provides the details of the use of the object for a particular application. Attributes and relationships can be specified in either or both the interface and implementation.

The generalization/specialization concept of Smith and Smith [SMIT77] will be used in the model to provide the relationship between types and their subtypes. Types will be defined either as generalizations of a set of named subtypes, or as primitives from which versions and instances can be made directly. An example of generalization would be the creation of a type house from the subtypes colonial, duplex, ranch, tri-level, and rambling. The notion of subtype is important to the model because different subtypes will be permitted to have different sets of attributes.

Version generalization [BAT085] is used as the mechanism for specifying the relationship each object type has with its versions. A version is created by specifying implementation details for the object type. The difference between a version and an instance of a type/subtype is that a version is created at an intermediate point in a design, permitting future designs to begin at that point, with implementation details partially specified. A type/subtype is considered a starting point for a design, with no implementation details specified. The concept of parameterized versions [BAT085] arises from the need for allowing freedom in specifying the implementation details for a particular object. If an instance of an object type is defined instead of an instance of one of its versions, a parameterized version is created. Choosing an instance of an object type T creates a socket which will accommodate any version of type T. Using the terminology defined in [BAT085], the different versions are plugged into the socket, creating unique implementations.

The generalization concept of Smith and Smith [SMIT77] differs from version generalization in that the former takes two or more object sets and forms a higher level object set by taking their union, and in the latter, an object type (or subtype) is an abstraction of the common features of its versions, which is clearly not a union of object sets.

In our model, a version of a type (or subtype) will be defined to be a molecular object with interface details completely specified, but with implementation details in some stage of completion. This definition allows a version to be plugged, partially plugged, or unplugged. Figure 5 shows an object of type A with an object version V1 of type A. The object of type A has its interface defined, which is denoted by the shading of the interface block. The implementation details for this object are not specified, denoted by the unshaded implementation block.

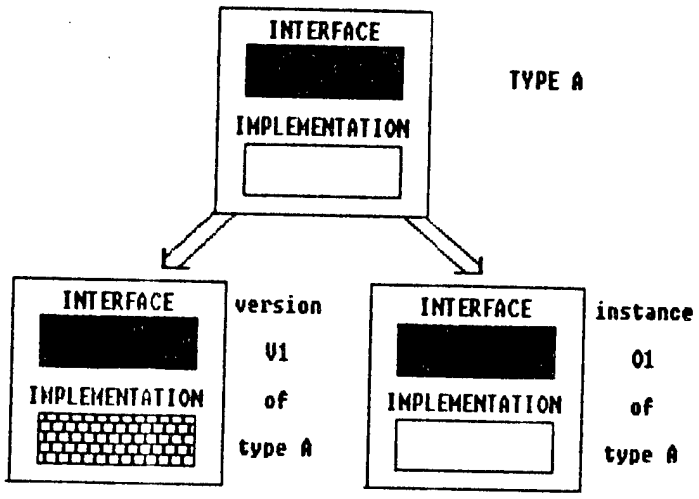


figure 5

Object V1 has the same interface details as its object type, and also has some implementation details specified, denoted by the partially shaded implementation block. Examples of this definition of version are the two, three, and four bedroom versions of a ranch house. In each of these examples, the interface (function) of the object is specified, but the implementation details (e.g. what are the sizes of the bedrooms?) are not specified completely.

Versions can have two distinct forms of attributes, those inherited from the object type, and those defined to be unique for each version. Attributes inherited from the object type reproduce the interface characteristics of the object type. Attributes defined to be version specific are the attributes which distinguish one version of a particular type from another version of the same type. Another way of describing version generalization is that it is a form of abstraction in which similar objects are related to a higher level object.

Instantiation [BATO85] occurs when an object is copied. Creating multiple instances of an object provides for distinction between the various copies. Both object types and object versions can be instantiated. The purpose of instantiating will be extended to provide meanings for instances of type and version. A version will be

instantiated to provide a local working copy of a previous design, which may be plugged to any level of detail. Types (or subtypes) will be instantiated to produce a working copy for design work from scratch, in cases where no existing design can be used. Figure 5 shows an object O1 which is an instance of type A. O1 would be produced to provide a working copy of type A as a starting point in this particular design. The fact that O1 is instantiated from its parent type tells us that the implementation specifications for the final product are not available and will be developed from scratch. If O1 were instantiated from V1 instead, the design would begin from the point in V1 where implementation details left off, indicating that some similarity exists between the implementation of O1 and V1.

A hierarchy is formed for the set of designs for a particular type/subtype, and is called a version hierarchy. In this hierarchy, going from a higher level to the next lower level, we find that more implementation details are specified. The difference between the type/subtype generalization and the version hierarchy is that different versions of an object have the same set of attributes, and not necessarily the same values, while different types (or subtypes) will have different sets of attributes from each other. Figure 6 depicts two version hierarchies. In this case, ranch and colonial are subtypes of type house. Each subtype can have its own version hierarchy. The blocks labelled two bedroom, three bedroom, and four bedroom are on the same level in the diagram because they represent mutually exclusive versions. Each block in the diagram is a potential starting point for future designs.

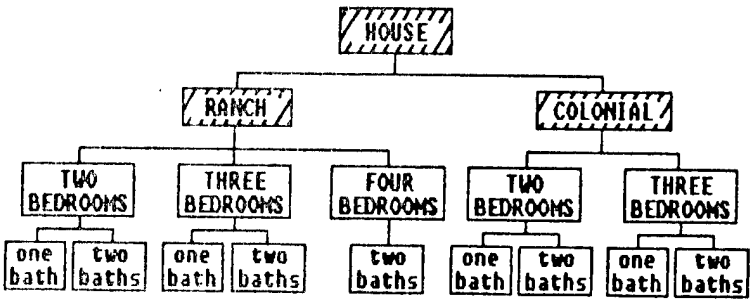


figure 6

An extension of the instantiation abstraction is the instance hierarchy. The purpose of this hierarchy is to record different design alternatives which are produced in the design process. Figure 7 is an example of an instance hierarchy for a house being designed for John Jones. Since Mr Jones is building this house from scratch, the design starting point was an instantiation from subtype ranch. In the course of designing his house, Mr Jones wasn't sure whether he wanted an attached or detached garage, two alternatives represented in the hierarchy. The reason for saving the hierarchy is that Mr Jones may decide on an attached garage, finish the design, and then change his mind. The hierarchy would permit him to go back to the point of the detached garage and re-complete the design. All of the information provided in the original design would be usable in the second design except for information about the garage itself.

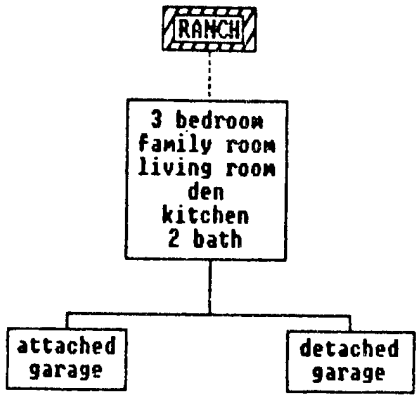


figure 7

Figure 8 summarizes the relationships between type, subtypes, versions, version hierarchies, and instances.

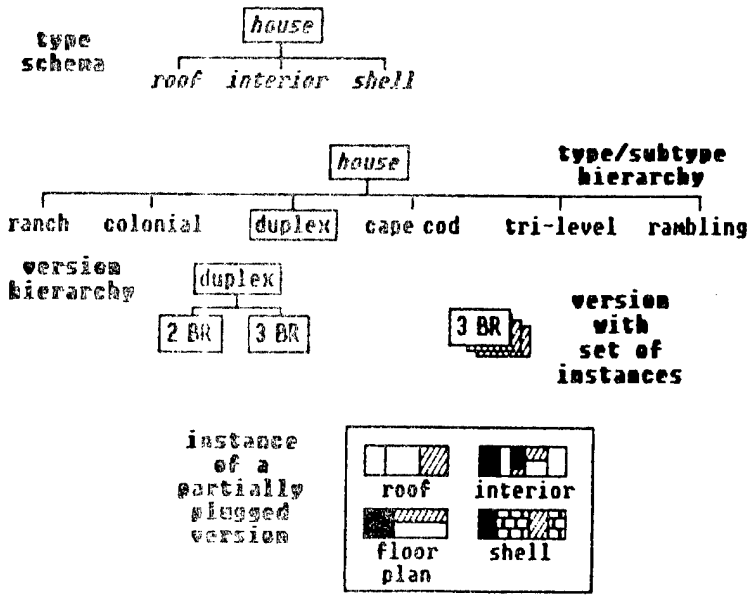


figure 8

The role of the data model in the design process will be to provide a standard which different product designs can use to ensure compatibility in the later stages of production. In particular, this standard will facilitate the integration of design data in the material requirements planning and scheduling phases.

3.3 Schema Data

The schema data consists of semantic network-type relationship information from the conceptual schema for a particular product. This schema data will be used by the expert system translator to associate design data according to the conceptual schema relationships. The relationships supported by our system are the IS-A and PART-OF [WINS84]. The IS-A relationship provides an attribute inheritance mechanism whereby the system can infer attribute values in cases where

those values were incompletely specified by the designer. Inheritance begins at the closest ancestor and continues up the ancestral hierarchy until a value is found. The relationships in the conceptual schema are stated in the form of facts, as shown in figure 9.

```
part_of(house,floorplan).   part_of(house,interior).
part_of(house,shell).       part_of(house,roof).
part_of(interior,story).    part_of(story,room).
part_of(story,space).       part_of(room,face).
part_of(space,face).        part_of(face,sub_cover).
```

figure 9

Our system distinguishes between schema data and the conceptual schema because the separation of these allows a user to modify the original conceptual schema in the design process without having to change the schema itself. This adds flexibility to the system and permits the conceptual schema to be implemented independently (i.e. can be represented in a form most appropriate for processing by CAD) of the schema data which will be used by the expert system translator. If no modification is made to the conceptual schema during the design process, the schema data does not have to be re-generated for each product.

3.4 Design Data

The design data consists of the instances of the prototypes created during the design process. All slots are filled in, either with default, inherited, or specified attribute values. As prototypes are instantiated, IS-A facts are asserted which associate the instance with the type from which it was created. At this point in the process, the design is considered to be complete. Any revision work would have been done prior to the design data being prepared for processing by the translator.

3.5 Translator Meta Rules

The translator meta rules, combined with the standards data, assembly data, and schema data will determine how the design data for a particular product will be transformed into material requirements planning data. These rules will enforce the standards given in the standards data, and provide the actual translation mechanism which produces the material requirements planning data. Figure 10 provides a sample of meta rules for the house design and construction example.

```
raw_materials_needed :-
  is_a(Extens,Intens),
  property(Extens,finish_type,Material),
  property(Extens,finish_color,Fcolor),
  liquid(Material,Ltype,Covers,Cunits,Lunits,Cost),
  dimension(Extens,height,Ht,Htunits),
  dimension(Extens,width,Wd,Wdunits),
  convert(Ht,Htunits,Height,Cunits),
  convert(Wd,Wdunits,Width,Cunits),
  Area = Height * Width * 2,
  Amt_needed = Area / Covers,
  Tot_cost = Amt_needed * Cost,
  assertz(liquid_list(Material,Ltype,Fcolor,Amt_needed,
          Lunits,Tot_cost)),fail.
```

figure 10

These meta rules will assert new facts which represent requirements for specific raw materials. Note that the materials list is refined for items such as paint, nails, caulking, etc., whose requirements are expressible as a function of the dimension of the object.

3.6 Standards Data

Design and manufacturing systems have to take into account a wide variety of Federal, State, local, Occupational Safety and Health (OSHA), quality assurance, and other standards prior to manufacturing a product. For example, a design could call for a 1/4" inside diameter pipe in a specific location, but a local building code may specify a 3/8" minimum inside diameter. In this case, the design specification must be changed to reflect the regulatory requirement. For a given

product, thousands of interactions are possible between existing standards and specifications generated from the design process.

These standards are represented in the system by Prolog-style rules to facilitate their enforcement by the expert system translator. Figure 11 gives an example of the implementation of a regulatory requirement.

```
maximum(pipe,plastic12,diameter,3,inches).
minimum(pipe,plastic12,diameter,1,inches).
passed(pipe,Type,Dimension,Z,Units) :-
  minimum(pipe,Type,Dimension,X,Unitx),
  maximum(pipe,Type,Dimension,Y,Unity),
  convert(X,Unitx,Min,Units),
  convert(Y,Unity,Max,Units),
  check_standards(pipe,Type,Dimension,Z,Units,Min,Max).
```

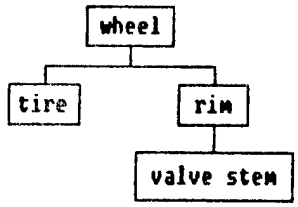
figure 11

The maximum and minimum facts shown on the first two lines provide the limits for a particular type of pipe. The passed predicate indicates that the minimum and maximum values with their respective units will be checked against the design values, indicated by the variable Z and units variable Units. The convert predicate converts the standards units of measure to the units in which the design object is measured. The check_standards predicate would compare all three measurements to a common unit of measurement and verify that the standard was met.

3.7 Assembly Data

Assembly data includes sequencing information for assembly of composite objects, or subassemblies, according to the relationships shown in the conceptual schema. This assembly data covers all conceptual schemata for a given application domain. In addition, information on standard material types and acceptable substitutes is included, with their costs. The system could take advantage of fluctuating costs with the substitution information to produce an optimal cost product.

The sequencing information will be represented in Prolog-style rules. Figure 12 provides an example of a portion of a conceptual schema with the sequencing rule to be included in the assembly data for the given product.




```
assemble(W,wheel) :- property(W,wheel,Wtype),
   part_of(W,T), part_of(W,R), part_of(R,V),
   property(T,tire,Ttype), property(R,rim,Rtype),
   property(V,valve_stem,Vtype),
   assertz(operation(Rtype,assemble,valve_stem,Vtype)),
   assertz(operation(Ttype,assemble,rim,Rtype)),fail.
```

figure 12

The first operation fact to be asserted provides for inserting the valve stem into the rim. The second operation inserts the rim into the appropriate tire. Note that operation information includes details of specific tires, rims, and valve stems. The assembly rule will produce a set of operation facts for each wheel defined in the design. Each wheel will be separately identifiable.

In the object-oriented approach, the assembly rules would be considered part of the operations encapsulated with each data type. We choose to separate these rules for the following reasons. First, the separation allows us to abstract out the implementation details so that the conceptual schema isn't tied to the rule-based implementation imposed by the assembly data. The separation also functionally aligns the conceptual schema and assembly data with the people responsible for maintaining them. The conceptual schema can be developed by users with little technical expertise or familiarity with the implementation considerations necessary to manufacture a product. The assembly data can be maintained by the manufacturing experts who are familiar with

implementation details, material properties that may lead to more cost effective substitutions of components, and the sequences of operations used in the manufacturing process. Another reason we separate them is that they serve different functions. The conceptual schema is used by designers, while the assembly rules are part of the expert system translator. The conceptual schema represents one product, but the assembly data represents all the conceptual schemata in the application domain. The assembly data could also contain information about the way the factory chooses to do assembly, which is independent of any particular product.

3.8 Material Requirements Planning Data

The main output of the translator will be a bill of materials containing information on the assembly of components into subassemblies and quantities of raw materials required for manufacture of component parts. This is known as material requirements planning data. It provides all the necessary information for the production of a product.

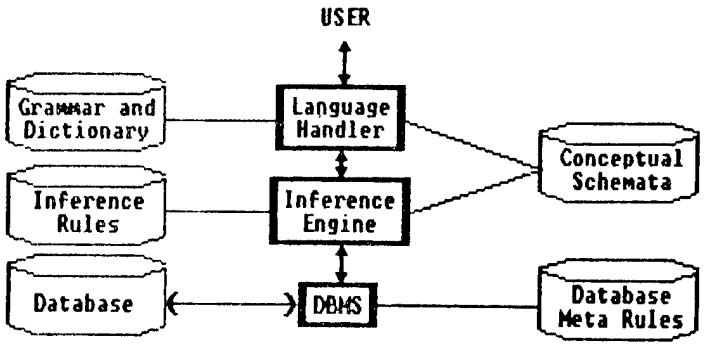
3.9 Scheduling Data

After the requirements for a new product have been determined, the new requirements data can be combined with existing production requirements in the scheduling phase. At this point, priority information is used by the system to determine how to integrate the new requirements into the existing workload. The scheduling data includes assembly data which will be used to coordinate construction of subassemblies with production of components and ordering of raw materials and purchased parts.

4. Expert System Shell Translator

The basic task of the expert system shell translator is to automatically conclude the quantities, types, and assembly sequences of raw materials needed to manufacture a product from design data. Before we examine this translator in detail, we will present some background information on expert systems.

Expert systems belong to a class of artificial intelligence applications known as *knowledge-based systems*. The thing that makes expert systems unique is that their performance depends on utilizing facts and heuristics used by human experts in similar situations. One of the characteristics of these systems is their large solution space wherein the number of reasonable solutions is usually a small percentage of the number of possible solutions. Figure 13 depicts the components of a knowledge-based system.



A Typical Knowledge-Based System

figure 13

Conceptually, these systems employ a representation scheme and some reasoning method. The representation scheme, determined by the *language handler*, permits expression of generalizations in the absence of complete information. Early systems used formal logic as the representation scheme and deductive reasoning, but abandoned logic in pursuit of more efficient representations. These efforts led to the development of knowledge-based and expert systems characterized by the

use of production rules and knowledge representation, i.e., object-oriented techniques. The reasoning method used by the knowledge-based system is determined by the inference rules manipulated by the *inference engine*.

Several benefits can be obtained by using a knowledge-based system to enhance data management. First, the inference engine is able to produce information that is not explicitly stored, but can be inferred from the known facts. The inference engine also permits users to work with the system without considering file structure and other implementation details. The ability of an inference engine to generate an audit trail or *line of reasoning* is invaluable in debugging a system. The inference engine can also act on *fuzzy* data, or data which is not completely specified, and produce results with the same degree of accuracy.

The translator we propose as the interface mechanism between the design and manufacturing processes will make use of the aforementioned benefits. In addition, the translator will provide for resolution in the event that it receives conflicting data. An example would be the preference of standards data over design data, in situations where standards would otherwise not be met. The translator will be opportunistic, that is, it will use substitution criteria whenever possible to lower cost without sacrificing quality. The translator will use the schema and assembly data as guidance to control the deduction process and limit the space of possible solutions.

The fact that this translator and most of the data used by it are rule-based introduces a variety of issues which should be addressed. Among these issues are inconsistencies, redundancies, and incompleteness in the rule base. Inconsistency is the predominant worry in rule-based systems because conflicting consequents inferred from the same set of facts (evidence) can result in faulty performance of the system. Remedies for inconsistency range from altering or removing part of a rule to major reorganization of the rule base. Our system will prevent inconsistency by checking new rules against the existing rule base using forward-chaining inference to derive possible

consequents. Users can be immediately informed if inconsistencies exist, and the new rule can be redefined.

Redundancies arise when different rules acting on the same set of facts arrive at the same conclusion. Incompleteness results from a failure of a rule base to derive a consequent from a given set of facts. Redundancy can be avoided by using a check similar to the consistency check. Incompleteness will be avoided by performing a test of the system in which conceptual schema data is passed through the translator.

Other major issues include possible difficulties in entering the rules into the system. Users should not have to concern themselves with the exact syntax of these rules and should be provided with an interface to facilitate the management of rules in the system.

4.1 An Example

The expert system translator we propose can be illustrated by use of a scaled-down example. A Prolog program for this example is presented as an appendix to the paper. We will limit our example by following the data from one prototype instantiated during the design process through the translator, producing a bill-of-materials. To begin, we will use the type *door* depicted in the conceptual schema in figure 3. The following figure is a prototype as it would appear after the data values are entered. When the prototype is first instantiated, the rightmost columns are either blank or contain default values. The leftmost columns are filled in as part of the conceptual schema definition process.

type door		
name	door1	
properties:		
material type	wood	
finish type	paint16	
finish color	brown	
knob type	round32	
hinge type	square3in	
height	78	inches
width	28	inches
depth	2.5	inches

figure 14

When the design process is complete, the following design data facts are created to represent the object door1 to the expert system translator:

```
is_a(door1,door).

property(door1,material_type,wood).
property(door1,finish_type,paint16).
property(door1,finish_color,brown).
property(door1,knob_type,round32).
property(door1,hinge_type,square3in).

dimension(door1,height,78,inches).
dimension(door1,width,28,inches).
dimension(door1,depth,2.5,inches).
```

figure 15

The is_a fact relates the instantiated object door1 to the type door. The property facts correspond directly to the property entries in the prototype and the dimension facts correspond to the dimension information. In the case of dimension attributes, both the value of the dimension and the unit of measure are variable data to be supplied during the design process.

A portion of the schema data to be used by the translator for the example conceptual schema includes the following facts and rules:

```
part_of(house,floorplan).
part_of(house,interior).
part_of(house,shell).
part_of(house,roof).
part_of(interior,story).
part_of(story,room).
part_of(room,face).
part_of(face,door).
trans_partof(X,Y) :- part_of(X,Y).
trans_partof(X,Y) :- part_of(X,Z),
                     trans_partof(Z,Y).
```

figure 16

The last two rules provide the recursion necessary for expressing transitive hierarchical relationships between two objects as defined in the conceptual schema.

Standards data, to be used by the translator, would include the following rules which would enforce fire, safety, and architectural standards:

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

begin_stds_check :-  is_a(Entens,Intens),
  dimension(Extens,Dimension,Z,Units),
  minimum(Intens,Extens,Dimension,X,Unitx),
  maximum(Intens,Extens,Dimension,Y,Unity),
  convert(X,Unitx,Min,Units),
  convert(Y,Unity,Max,Units),
  check_stds(Extens,Intens,Dimension,Z,Units,Min,Max).
```

figure 17

The format for maximum and minimum standards allows flexibility in specifying that different versions of a type could have different standards. For example, casement windows could have a minimum width of 18 inches, while bay windows could have a 36 inch minimum width. The begin_stds_check rule works on one dimension of an object at a time,

but is general enough to apply to any dimension of any object, avoiding redundant rules.

Assembly data, pertinent to this example, would consist of the following rules:

```
finish(Extens) :-  property(Extens,finish_type,Ftype),
   property(Extens,finish_color,Fcolor),
   assertz(operation(Extens,finish,Ftype,Fcolor)).

assemble(Extens,knob) :-
   property(Extens,knob_type,Ktype),
   assertz(operation(Extens,assemble,knob,Ktype)).

material(door1,wood,7,feet,3,feet,2.5,inches,
         0,feet,0,feet,15.75).

liquid(paint16,paint,1200,feet,gallon,7.50).
```

figure 18

The order of invocation of these rules will determine the order of operations to be performed. The material fact provides the dimensions for a standard piece of material of the given type, in this case wood for door type door1, and the cost of one piece of this material with the standard dimensions. Information on the density of fasteners can also be provided (note the zeros for the 9th and 11th parameters) in the event that this material needs nails, screws, etc, for piecing it together. The density information allows for different materials of the same size to be fastened appropriately. For example, board lumber which comes in 8 feet by 1 foot planks could have a height density of .5 per foot and width density of 1 per foot, which means that fasteners would be placed every 2 feet along the height dimension and every foot along the width dimension. Particle board planks with the same dimensions could have a different density for fasteners due to the binding properties of the material, i.e. height density of .75 per foot and width density of 2 per foot. The rules invoked to determine

the number of fasteners required for a piece of material would use the following formula:

number required =

((height X height density) + 1) X ((width X width density) + 1)

Similarly, the liquid fact says that one gallon of part number paint16 will cover 1200 square feet and costs $7.50.

The following report is produced by the expert system translator, and represents the results of standards verification plus the material requirements planning data for the object door1 of type door used in this example.

Standards check for door door1

door door1 passed - height
door door1 failed minimum - width

Production Sequence Report

door1	finish	paint16	brown
door1	assemble	knob	round32
door1	assemble	hinge	square3in

Raw Materials Report

wood	78 inches	28 inches	2.5 inches	$15.75
knob	round32	1		$3.20
hinge	square3in	3		$.75

liquids

paint	paint16	brown	0.03 gallon	$.19

5. Conclusion

The successful integration of product design and manufacturing functions requires a complete understanding of the relationships of data produced and used throughout the product life cycle and some mechanism to translate product design data into a form which is useful in the manufacturing process. In this paper, we have described this data and demonstrated an expert system translator which integrates the design and manufacturing functions. The data was classified according to its role in the integration process, and consists of a data model, conceptual schema, standards data, assembly data, schema data, design data, and translator meta rules.

The role of the data model is to provide a standard for compatibility across product design boundaries which will facilitate the material requirements planning process for a multitude of products. A conceptual schema will be required for each different product design, and will characterize the structure of the design data for a product at any point in time. The standards data should be viewed as constraints that design data must obey to be acceptable for production. Assembly data includes sequencing information for assembly of composite objects and equivalence data to increase cost effectiveness of a design without violating the standards constraints. The schema data is just the conceptual schema in rule form, with the added flexibility of temporarily modifying the conceptual schema for a specific product. The design data consists of instantiated prototypes for each object used in the design, and includes attribute inheritance features when necessary. The translator meta rules are used to produce the MRP data from the available design, schema, standards, and assembly data.

The expert system translator is a rule-based reasoning machine which produces information about the quantities and types of raw materials required to produce a product. The translator also provides assembly sequence information for use in later stages of production.

Directions for further research include the design of a user interface for managing the rules in the system and the development of a graphics interface for defining the conceptual schema to the system. An additional research area deals with possible internal representations of design objects using syntax directed editors with context-free [RAB185] or context-sensitive [LEE83] grammar rules. The use of such an editor will ensure that the design satisfies integrity constraints (which will keep a window from being placed in a floor).

The translator we have proposed is currently under development at the Laboratory for Database Management at the Naval Postgraduate School. The Appendix contains a listing of the Prolog code for the example given in this paper. We expect this integrated system project to spawn several other projects including user-interfaces for data modeling, database management support for rule-based systems, and multi-media data models.

References

[BATO85] BATORY, D.S. and KIM, W. Modeling Concepts for VLSI CAD Objects. *ACM Trans. Database Syst. 10,3* (Sep. 1985), 322-346.

[BROD82] BRODY, M.L. On the Development of Data Models. *On Conceptual Modeling*, Springer-Verlag, 1982, 19-47.

[CHEN76] CHEN, P.P.S. The Entity-Relationship Model-Toward a Unified View of Data. *ACM Trans. Database Syst. 1,1* (Mar. 1976),9-36.

[LEE83] LEE, Y.C. and FU, K.S. A CSG Based DBMS for CAD/CAM and its Supporting Query Language. *Databases for Engineering Applications*, 1983, 123-130.

[MCLE83] MCLEOD, D., NARAYANASWAMY, K., BAPA RAO, K. V. An Approach to Information Management for CAD/VLSI Applications. *Databases for Engineering Applications*, 1983, 39-50.

[RABI85] RABITTI, F. A Model for Multimedia Documents. *Office Automation*, Springer-Verlag, 1985, 227-250.

[SMIT77] SMITH, J.M. and SMITH, D.C.P Database Abstractions: Aggregation and Generalization. *ACM Trans. Database Syst. 2,2* (Jun 1977), 105-133

[SU86a] SU, S.Y.W., Modeling Integrated Manufacturing Data with SAM*, *IEEE Computer*, Jan 86, 34-49

[SU86b] SU, S.Y.W., et al, The Architecture and Prototype Implementation of an Integrated Manufacturing Database Administration System, *IEEE Computer Society International Conference*, 1986, 287-296

[WINS84] WINSTON, P.H., *Artificial Intelligence*, Addison-Wesley, 1984

Appendix

```
/* Expert System Translator */


start :-
        writedevice(printer),
        not(begin_stds_check),
        not(begin_operations),
        not(operations_report),
        not(raw_materials_needed),
        not(materials_report),
        writedevice(screen).

begin_stds_check :- is_a(Extens,Intens),
        write("Standards check for ",Intens,Extens),nl,nl,
        dimension(Extens,Dimens,Z,Units),
        check(Intens,Extens,Dimens,Z,Units),fail.

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

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

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

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

begin_operations :- is_a(Extens,Intens),
        not(do_finish(Extens)),
        not(do_assembly(Extens)),fail.

do_finish(Extens) :- finish(Extens),fail.

do_assembly(Extens) :- assemble(Extens,Notused),fail.

operations_report :- nl,nl,nl,
        write("Production Sequence Report"),nl,nl,
        operation(Extens,Function,Attribute1,Attribute2),
        write(Extens,"   ",Function,"   ",Attribute1,
           "   ",Attribute2),nl,fail.
```

```
raw_materials_needed :-
        is_a(Extens,door),
        material(Extens,Material,_,_,_,_,_,_,Hdens,Hdunits,
           Wdens,Wdunits,Cost),
        dimension(Extens,height,Ht,Htunits),
        dimension(Extens,width,Wd,Widunits),
        dimension(Extens,depth,Dp,Dpunits),
        assertz(material_list(Material,Ht,Htunits,Wd,
           Widunits,Dp,Dpunits,Cost)),fail.

raw_materials_needed :-
        is_a(Extens,Intens),
        property(Extens,finish_type,Material),
        property(Extens,finish_color,Fcolor),
        liquid(Material,Ltype,Covers,Cunits,Lunits,Cost),
        dimension(Extens,height,Ht,Htunits),
        dimension(Extens,width,Wd,Wdunits),
        convert(Ht,Htunits,Height,Cunits),
        convert(Wd,Wdunits,Width,Cunits),
        Area = Height * Width * 2,
        Amt_needed = Area / Covers,
        Tot_cost = Amt_needed * Cost,
        assertz(liquid_list(Material,Ltype,Fcolor,Amt_needed,Lunits,
           Tot_cost)),fail.

materials_report :- nl,nl,nl,write("Raw Materials Report"),nl,nl,
        material_list(Material,Ht,Htunits,Wd,Widunits,Dp,
                Dpunits,Cost),
        write(Material,"  ",Ht,"  ",Htunits,"  ",Wd,"  ",Widunits,
                "  ",Dp,"  ",Dpunits,"  $",Cost),nl,fail.

materials_report :- nl,nl,write("liquids"),nl,nl,
        liquid_list(Material,Ltype,Fcolor,Amt_needed,Lunits,Tot_cost),
        write(Material,"  ",Ltype,"  ",Fcolor,"  "),
        writef("%3.2",Amt_needed),
        write("  ",Lunits,"  $"),
        writef("%3.2",Tot_cost),nl,fail.

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).

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

Design Data

```
is_a(door1,door).

property(door1,material_type,wood).
property(door1,finish_type,paint16).
property(door1,finish_color,brown).
property(door1,knob_type,round32).
property(door1,hinge_type,square3in).
dimension(door1,height,78,inches).
dimension(door1,width,28,inches).
dimension(door1,depth,2.5,inches).
```

Schema Data

```
part_of(house,floorplan).
part_of(house,interior).
part_of(house,shell).
part_of(house,roof).
part_of(interior,story).
part_of(story,room).
part_of(story,space).
part_of(room,face).
part_of(space,face).
part_of(face,door).

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

Standards 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).
```