A View of Next Generation Tools for Design

by

Sarosh Talukdar and Arthur W. Westerberg

EDRC 05-17-88

A VIEW OF NEXT GENERATION TOOLS FOR DESIGN

by
Sarosh Talukdar
Arthur W. Westerberg

Engineering Design Research Center
Carnegie Mellon University
Pittsburgh, PA 15213

Presented as Paper 23a, 1988 Spring National Meeting, AIChE, New Orleans, LA, March 6-10, 1988

ABSTRACT

In this paper we examine the characteristics of current tools that support the design activity. Based on their shortcomings, we propose a set of characteristics that future tools should have. To illustrate the nature of the issues involved, we suggest one form in which the design activity itself might be modeled and then discuss some fundamental issues related to structuring tools to support this model.

INTRODUCTION

This work is being pursued as a part of the activities of the Engineering Design Research Center (EDRC) at Carnegie Mellon University. The EDRC is one of fourteen Engineering Research Centers funded by the National Science Foundation in the last three years. Our vision is to provide leadership in the *development* and *integration* of design methodologies that will make U.S. industry preeminent in design practice. There is no doubt that this is formidable charter.

Among the items listed for our strategic plan, we include the creation and demonstration of new design methodologies and the integration of previously autonomous design activities and tools.

For administrative purposes the center is partitioned into three laboratories. Projects in each laboratory typically deal with several research issues; however, each laboratory emphasizes one key research thrust, as illustrated in Figure 1. Three of our current projects are "plotted" to show that each covers more than the issues of a single laboratory.

Figure 1: Laboratories in Engineering Design Research Center.
Three projects illustrate how each in fact deals with issues from
all the laboratories

- The **Design for Manufacturabilfty Laboratory [DFM]** is developing ways of bringing life cycle concerns (e.g., manufacturability, flexibility, safety) back to the early design stages.
- The **Synthesis Laboratory [SY]** is building methods for the automatic generation and selection of design alternatives based on incomplete information at the preliminary design stages.

- The **Design Environments Laboratory [DE]** is creating significantly improved next generation design environments as opposed to incrementally improving current environments.

Design methodologies developed in the DE Laboratory and the other laboratories will eventually become the basis for new tools and organizations to support them. The DE laboratory is studying what the form for these tools and organizations should be by developing prototype design environments in specific domains, e.g., auto parts and buildings. This activity has led us to examine the characteristics of current computer tools which support the design process. In the next section we list their weaknesses as a basis for starting a discussion of the characteristics we desire for the next generation of tools that should be created.

UPGRADING DESIGN PROCESSES

The important properties of artifacts like cars, computers and microelectronic chips, are determined by their designs. Manufacturers of such artifacts must maintain competitive design systems, if they are to preserve their market shares. When existing systems have considerable embedded value, as is often the case, the logical approach to keeping them competitive is through redesign and upgrading, rather than through complete replacement. The two dimensions along which improvements are usually needed are *speed* with which designs can be completed and *quality* of the designs. Cost is a consideration but of lesser importance because design cost is often only a small fraction of the total cost of an artifact.

Virtually every large design project has three stages: the decomposition of the project into loosely coupled, partially ordered tasks (as illustrated in Figure 2 for the design steps one might propose for a chemical process design), the performance of the tasks, and the integration of the results. In mature industries the decomposition stage is often fixed by tradition and varies more in the selection of who is to perform the tasks than in the tasks themselves. Take the case of the design of a new car in the U.S. The traditional tasks are: design the outer shape of the car; then design components to conform to the outer shape; then design processes for manufacturing the components; and so on. The second stage-the actual performance of these tasks-is carried out by task forces using design processes that have been developed and refined over considerable periods of time, often decades. As a result, these task forces have developed distinct technical cultures that resist intrusions and integration. The purpose of the third stage is to transcend these barriers, at least to the extent of ensuring that the results from the task forces can be integrated. For instance, the engine must mate with the drive train. Also, the engine must be manufacturable, testable and maintainable. If this sort of integration among the results of task forces is to be achieved, the second and third stages must run in parallel. Moreover, agents from the third stage must be able to monitor and adjust the activities of the task forces of the second stage. This monitoring and adjustment process has come to be called simultaneous engineering or concurrent design.

Figure 2: Project Decomposition into Loosely Coupled, Partially Ordered Tasks

Second Stage Weaknesses

Some of the more profound weaknesses of existing design systems stem from their second stages. These weaknesses include:

- closed design processes that inhibit the monitoring and mid-course adjustments needed for effective simultaneous engineering;
- infrastructures that support only a fraction of the representations needed (While complex design processes call for wide varieties of representations, like geometrical models, differential equations and block diagrams, existing CAD systems seldom support more than one or two of these representations.);
- poor coverage of process activities by computer tools (While analysis and drafting activities are often well covered, synthesis and management activities are usually poorly covered.);
- few automatic ways for integrating computer tools. (Existing CAD tools tend to be large, stand-alone programs. They are difficult to interconnect, learn and use.)

To understand how to alleviate these weaknesses one must look more closely at the design processes of the second stage, as we will do in the next section.

DESIGN PROCESSES

Two dimensions of a design process-architecture and control-will be briefly discussed below.

Architectures and TAO Graphs

The three main components of a design process are aspects, operators and tests. An aspect is a perspective or view of an artifact from any point in its life cycle. Sketches on the back of an envelope, detailed blueprints, and full size prototypes are all aspects of a car. For the purpose of design processes, aspects can be divided into three categories:

- input aspects (given data),

- output aspects (goal states), and
- intermediate aspects (subgoal states or stepping stones to the outputs).

Operators calculate the intermediate and output aspects and can be either manual or automatic. Large processes usually require some of each.

Tests are special cases of operators. The usual function of a test is to compare two or more aspects for consistency and post the results of the comparison in another aspect. An example of a simple test is the comparison of the successive results from an iterative process to a given threshold. A more complex test would be needed to check the design of a car against specifications that call for the car to last at least ten years. As these examples illustrate, tests are task-specific and often subjective.

A convenient way to describe the architecture of a design process is by a directed graph whose nodes represent aspects and whose arcs represent operators and tests. We will call these graphs TAO (test-aspect-operator) graphs. Figure 3 illustrates a TAO diagram that one might create for the design of a chemical process.

Figure 3: Example TAO Diagram for Process Design

Control

The control problem can be stated as follows: given a TAO graph, select paths by which to calculate the output aspects from the input aspects. Some of the given inputs may be test results. That is, the paths may have to be selected so that certain tests are passed. Often this requires cycles (iterations), with the control scheme acting to reduce inconsistencies among aspects, much as a classical control system acts to reduce errors.

Even if the computerized operators in a process are constrained to running serially, the humans in the process are not, and therefore, the control scheme must accommodate distributed problem solving strategies.

Binding Constraints

We can now turn to the question of what architectural or control features are responsible for the "second stage weaknesses" mentioned earlier. From an examination of some processes in automobile, electronics and construction industries, we have come to suspect that the binding constraints on the performance of many, if not most, industrial design processes stem from their architectures rather than their control schemes. In seeking to relax these constraints one should consider the following factors:

1. aspect placement (Do the aspects decompose the overall task into manageable subtasks? Do they provide the information and connection points needed for simultaneous engineering?);
2. aspect capability (Do the data abstractions and representation schemes used by each aspect adequately capture and present the information needed by the operators that deal with the aspect and, just as important, hide the information that might confuse these operators?)
3. operator capabilities (Are the operators effective?)
4. history and explanations (Can the operators record and explain their intentions and actions?);
5. test adequacy (Do the tests make sense and are they appropriately placed?);
6. modularity and expandability (Can new aspects, operators and tests be readily added to the architecture?).

A CASE STUDY

This section outlines a project, called CASE, which has been jointly undertaken by the EDRC and Fisher Guide, to illustrate how a design process might be upgraded. The project deals with window regulators-devices that raise and lower the glass in automobile doors. The type of regulator considered has three main parts: a lift arm to move the glass; a sector and pinion combination to translate handle rotations into lift arm motion; and a backplate to fix the entire device to the inner door panel.

A TAO graph for the existing window regulator design process is shown in Fig.4. The examination of this graph reveals that:

Figure 4: TAO Diagram for Existing Window Regulator Design Process

- most of the operations are manual. Automation of the routine and well understood parts of these operations would produce savings in time.
- much of the generative reasoning is done in a single, manual operation that transforms the stick diagram into blueprints. This reasoning is relatively inaccessible which causes design changes to be difficult to make. New aspects, placed to break this single operation into smaller, more comprehensible, modular operations, would make the design process more open and changes easier to implement.
- there are no automatic aids for simultaneous engineering.

With these observations in mind, we have embarked on the first phase of an upgrade that involves the addition of aspects and automatic operators in the several areas of design: synthesis, optimization, analysis, and simultaneous engineering. The result is the TAO graph shown in Figure 5.

Figure 5: TAO Diagram for Revised Window Regulator Design Process

ABSTRACT VIEW OF TAO DIAGRAMS

Figure 6 provides a more abstract view of TAO diagrams and their relationship to design "spaces."[1] The term space is used in many different ways, but here we mean the (likely infinite) set of all possible instances of aspects that might occur for a particular level of representation for an artifact. A member of a space is a particular aspect for a particular artifact. The mapping of an aspect in one space into one in another more refined space is a *synthesis* operation. Typically it is a very complex activity and is accomplished by generating many alternatives and selecting one which is deemed to be "best" in some sense. Without the criterion to select a best alternative, the mapping becomes a one-to-many mapping where several refinements *satisfice* the more abstract aspect.

Figure 6: An Abstract View of TAO Diagrams

The mapping from a more refined aspect to a more abstract one is also very complex. It too is generally one-to-many unless there are some specific mapping rules which can be stated and applied. The mapping is similar to finding the question for which the answer is known.

There may already be an existing version of the neighboring aspect from an earlier pass through It. The operator could in this case attempt to choose the target aspect which is consistent with the more source aspect and which is *closest* to the current version of the target aspect. The aspect which will subsequently propagate the fewest changes to neighboring aspects could also be defined as *closest*

To test is to compare two aspects to see if they are mutually consistent. Testing can be very complex as there can be parts of the two aspects for which one may be unable to ascertain consistency without generating other aspects and, as illustrated above for the autoparts example, the testing may be subjective.

The operators in TAO diagrams have a fractal nature to them. Complex operators in TAO diagrams can be implemented by expanding them into sub-TAO diagrams, as Figure 7 illustrates.

Figure 7: "Fractal" Nature of TAO Operators

It is useful to compare TAO diagrams to chemical flowsheets to make clear the significant differences between them.

Consider the TAO diagram left hand side of Figure 8a. Think of this diagram as if it were a chemical engineering flowsheet with the nodes being "streams'* and the arcs being "unit operations," the inverse of the normal way to represent a flowsheet. Figure 8b reverses the roles of the arcs and units so we can view them with the more conventional representation. To reverse the roles in the diagram in Figure 8a to create 8b often requires that a node be represented by several arcs, as, for example, happens to node 2.

Figure 8: Perspectives of TAO Diagrams, (a TAO diagram - activity on edge network, (b) TAO diagram - activity on node network.)

Let us step through a design. The designer will provide as much of the highest level aspect as he/she can. Assume this level is a top level specification for the artifact. The operator "a," a complex operator, will generate or request defaults for the information not provided and then, through a synthesis operation, create aspect 2, a more refined representation of the artifact. In a similar fashion aspects 3 and 4 are created. We note that there is an operator that can translate aspect 4 back to 2. For solving a flowsheet, we would invoke it and iterate until there was consistency among the unit operations and streams. In a TAO network with "perfect" operators, the operators from aspect 2 to 3 and from 3 to 2 would be the inverse of each other; i.e., the aspect mapped from 2 to 3 would be recreated by mapping that result back to 2. A complete design (but not necessarily a good design) can thus be constructed by defining completely one aspect from which one or more directed paths exists to all other aspects.

The aspect shown as 5 in Figure 8a is for testing. One of the design specifications in aspect 1 may require the development of aspect 5 before one knows if that a specification in aspect 1 can be met by the design. Thus the (synthesis) steps leading from aspect 1 through to 4 may not be complete until this test succeeds. Iterations on these steps may be required.

Let us assume that during the developing of a design, the designer chooses to alter aspect 4 in Figure 8a. He/she would do this by creating a version 2 of the design, as in the right side of Figure 8a. This version is constructed by copying all aspects for version 1 plus the alteration. The alteration is then mapped onto the other aspects by executing operators e, b, and d in that order. No operator exists to map back to aspect 1. The designer would have to become the operator by making changes manually to aspect 1 and testing if the results are consistent with version 2 of aspect 4.

Operators will seldom be perfect in the above sense. Thus it is very likely that mapping from aspect 2 to 3 and back will result in a difference in the starting and final version of aspect 2. It is a serious question as to what to do in this case. As mentioned earlier, one could strive to create all operators so they make the fewest changes to the target aspect or make changes which will halt the propagation of changes subsequently to neighboring aspects when that aspect has been defined before. Thus the operator from 3 to 2 could first test if aspect 3 is compatible with aspect 2 and, if so, make no changes to aspect 2, in which case there would be no need to propagate changes manually back to aspect 1.

MODELING TAO'S

We need a tool that will allow a designer to create TAO's quickly and accurately. When designing a new type of artifact, the designer will have to construct many alternative approaches for completing the design. Slowly a network of convenient aspects will evolve, with the connecting operators initially being the manual intervention of the designer. As the design becomes more routine, the structure of the TAO becomes fixed and many of the operators will be automated. One must assume, however, that no design will be completely routine, and changes will always be needed to a TAO structure. Therefore, a tool to permit their rapid creation is clearly needed.

Of course changes in TAO's for routine design have to be done carefully so previously designed artifacts will still be correctly designed by the modified system. No one appreciates finding a computer code that worked yesterday does not work today.

What are desirable characteristics for this type of tool? We suggest the following.

- We should be able to create aspects without knowing about the operators that will translate them into other aspects. If we can do this, unanticipated operators can more readily be added later to connect existing aspects.
- Each operator must access information in aspects associated with it in a unique manner. Therefore, the interface cannot be anticipated by the aspects but rather must be defined by the operator.
- Most desirable would be if an operator could be implemented totally nonprocedural^. One way would be if the operator is a set of algebriac constraints among the variables of the two aspects. An equation solving package capable of solving the equations in either direction would allow the operator to propagate changes in both directions. The operator would be bidirectional. An aspect may be partially specified nonprocedural^, allowing the propagation of some changes in both directions using the same constraints.
- We would like an ^{M}elegant^{M} and uniform approach.
- For the present it appears the human designer will have to do the "wiring" together of such networks. We wonder if some operators can be automatically created, at least partially, through the use of pattern matching.

Are these properties possible? We conjecture that some of them will be. In the Design Environments Laboratory of the Engineering Design Research Center, we are developing the interactive ASCEND system for aiding the rapid building and solving of models described by algebraic equations [Piela and Westerberg, 1988]. ASCEND comprises a language, a compiler and a "solving engine." We shall step back and examine those features of this language which suggest how we might develop a TAO modeling system. These ideas are speculative.

The ASCEND Language

We show in Figure 9 three characteristics of languages that can be used to distinguish them. The vertical axis is the degree of structuring which is naturally supported within the language. The right to left axis is the degree to which the language deliberately separates the declarative aspects of the problem to be encoded from the procedural parts, and, finally, the front to back axis represents the degree to which information hiding is a part of the language. The placement of several languages on this diagram corresponds to our interpretation of their characteristics.

We place the assembly languages and older versions of FORTRAN at the origin of this space. The only structures supported in FORTRAN were vectors and arrays. While one could do list processing in FORTRAN IV, it had to be done by writing code. Subroutines and functions allowed a considerable degree of information hiding. Only the formal parameters listed in the subroutine or function

Figure 9: A Morphology of Computer Languages

statement could be seen by the user of them. Finally there is no attempt in FORTRAN or in assembly languages to allow for the declarative specification of relationships. Specifically, a statement of the type

A = **2.0*A,**

means that is to be replaced by twice A. In a purely declarative language, this statement would require A to be zero.

LISP is largely unstructured too, but it allows one to handle arrays and lists naturally and directly so we can place it slightly up the vertical axis. PASCAL permits the handling of arrays, lists (pointers are a part of the language), and finally "records.[1]* A record is a mixture of variable types associated together and given a name. One can then have, for example, an array of records. Records and lists give one powerful structures to use when programming.

Moving forward on the "hiding" axis from PASCAL, we can place ADA and MODULA II. These languages allow for data hiding. One can, for example, write code to do matrix manipulations **and** separately write code that describes how the matrices are to be stored. These codes can be linked together at execution time. Thus one can "hide" the storage details from the code that manipulates the matrices.

At the top of the vertical structure axis are several of the expert system shells such as Knowledge Craft, KEE and ART. These shells provide for the use of class definition and inheritance hierarchies to structure information in a very powerful manner. Figure 10 illustrates. Here we define a class called VEHICLE and another called MANUFACTURED-PRODUCT. Through "IS-A" links, we define AUTOMOBILES and TRUCKS to be subclasses of both the VEHICLE and MANUFACTURED-PRODUCT classes. We can further define subclasses of AUTOMOBILE and finally, through an "INSTANCE-OF" link, create a particular instance of a car, MY-CAR. At each level in this hierarchy, information can be attached which is generally true of all subclasses and instances attached below. For example, the notion that there are wheels can be attached to the VEHICLE class. Attached to AUTOMOBILE class can be that there are four wheels. Finally MY-CAR can have the information that the wheels are decorated with spoked hub caps. If one asks about how many wheels MY-CAR has, the answer is found automatically by these shells by moving up the hierarchy until the number four is found under the AUTOMOBILE class. This very powerful structuring and retrieval of information has a profound impact on the ease with which certain classes of programs can be created.

Figure 10: Example Class Definition and Inheritance Structure

Moving out on the hiding axis from the Expert System Shells, we find Object Oriented Languages (OOL), such as SMALLTALK, LOOPS and FLAVORS. Added to the ability to create class definition and inheritance hierarchies is the notion of complete hiding. A pure OOL program is made up of "objects" which communicate with each other through the passing of messages. Each object has a set of messages it can respond to and the internal information structure and code to permit computations to occur when responding. The object sending a message has no idea of how the object receiving the message computes its response. Thus, pure OOL programs are remarkable modular. Objects which handle the same messages can readily replace each other. There are many examples where very complex programs have been written with ease in object oriented languages, as much as five times more quickly than in FORTRAN.

Moving out the left to right axis from FORTRAN, we find spreadsheet languages (these are realty programming systems). Spreadsheets are arrays of cells. For example one may have an array of 500 by 1000 cells in a program. Each cell can contain a character string or a number. The number can be a constant or the result of executing a formula attached to the cell that says how to compute it in terms of the contents of other cells. The spreadsheet system automatically determines the sequence to execute the formulas, and, every time a new value is placed in one of the cells, this computational sequence is executed. As long as the computations fully precedence order, these programs work just fine. If there is a circular definition among the cells, then one is warned. Repeated execution is permitted and may cause the cell values to equilibrate. Generally, one should limit the formulas so they will precedence order.

At the end of the axis, we place TKISOLVER and PROLOG. TKISOLVER is an equation solving package that runs on the IBM-PC. One simply types in the equations - say 10 equations and 15 variables. After selecting and providing values for five of the variables, one may have to guess some of the remaining variables if the equations cannot be precedence ordered. One then triggers the system to solve. It automatically develops the solution procedure and solves, iteratively if needed. The programmer using TKISOLVER simply types in the equations. These must be true at the final solution. He/she does not develop the code to solve them. Thus TKISOLVER specifically partitions the declarative information (what must be true at the solution) from the procedural (how to find it). A statement such as

```
A = 2.0*A
```

will result, as indicated earlier, in A equal to zero.

PROLOG is to the solving of logical relationships as TKISOLVER is to the solving of systems of algebraic equations. One simply states what must be true, inserts some facts and PROLOG deduces all the other things that must be true. It in effect develops the solution procedure.

ASCEND is a modeling language like TK!SOLVER for setting up and solving algebraic models. However, it is for setting up and solving very large models comprising thousands of equations and variables. It moves along the "hiding" axis to the opposite extreme of Object Oriented Languages, i.e., to the extreme of no hiding. It supports powerful information structuring, and it partitions strongly the declarative information from the procedural information.

In ASCEND, there are two objects: models and instances of models. We can develop the information structure among these objects through the use of five relationships.

- Models can REFINE other models,
- IS-A allows one to declare an instance of a model,
- ARE-THE-SAME is a powerful equivalence operator that allows two instances to be declared equivalent to each other. Since models can be built out of other models, ARE-THE-SAME makes equivalent complex structures with all their relationships. It permits the wiring up of complex networks of models and will be one of the concepts we will explore for creating TAO networks. [The complexity of equivalencing two models which have been partially instantiated should not be underestimated; however, it is possible to define this operator so one can determine when it can and cannot be done.]
- ARE-ALIKE is an operator that allows one to require two or more instances to be similar to each other when writing a model. An example is to write a model for an equilibrium stage and to require the liquid stream into the stage and the liquid stream leaving the stage to be similar - e.g., have the same components and the same methods for computing liquid activity coefficients. [As with ARE-THE-SAME, care is needed for this operator to have a proper and implementable definition.]
- UNIVERSAL is a qualifier on a model that says all instances of this model are automatically equivalenced. There should only be one copy of the physical properties of methane in the system, for example.

The last notion we wish to expose for ASCEND is that ASCEND deliberately does not hide information. A model is build up of instances of other models, instances of atoms and relationships among them. For example Figure 11 indicates the structure of an ASCEND model corresponding to the class definition and inheritance hierarchy in Figure 10. In the first model defined for vehicle, we indicate the existence of an atom (think variable) called Q. In this example we have suppressed all other information in the model. The model for manufactured-product contains an atom A. The automobile model is a refinement of the vehicle model and thus inherits all the information for the vehicle model. We attach the automobile model to the manufactured product model by declaring within the automobile model the existenc of MP, an instance of a manufactured-product. Variable A in the manufactured-product model is available by simply naming it MP.A. It is NOT hidden. The decision to access variable A is made by the person creating the automobile model, not the person writing the manufactured-product model.

Figure 11: ASCEND Model for Structure in Figure 10

Language Concepts Useful for Creating TAO Diagrams

We ask now if any of the concepts described above for any of the computer languages might prove useful for creating TAO diagrams.

Aspects will typically use very different representations. We have already noted that operators are complex and procedural.

Figure 12 illustrates how we might use the some of the concepts in ASCEND to develop a TAO structure. The first useful property that we indicated above that we would like for a TAO modeling language is the we could write the aspects without worrying about what operators would later connect them together. We could imagine creating aspects A1 and A2 as ASCEND-like models. See Figure 12a.

Figure 12: ASCEND-like Modeling of TAO's. (a) is the code, (b) shows how OPER12 ^{H}owns^{M} two aspects, A and B of types A1 and A2 respectively.

The next property was that we wanted the operators to have access to any of the information within an aspect. We could write an operator OPER12 to connect aspects of type A1 and A2. OPER12 will contain instances of A1, called A, and A2, called B. We might think of OPER12 as owning A and B; Figure 12b illustrates pictorially. All variables within both A and B are now available because there is no hiding of variables. We might next create another operator that wires together an aspect of type A1, called C, to one of type A3, called D. Figure 13 illustrates. Equivalencing A with C, as indicated, wires the network together.

Figure 13: Wiring a TAO Network Together With Equivalencing

These examples show that one can wire such structures together using ASCEND type concepts. Indeed it almost looks easy to do. If the operators can be written nonprocedural^, our third very desired property, ASCEND solving concepts could even play a part in implementing them. It should be made clear though, that most operators will be procedural and very complex.

Structuring Archival Information

As long as we are being speculative, we can go one step further with the following. Part of the support needed for the design process is to provide information from the literature and company reports that support many of the approaches used and many of the decisions made. Is this type of support consistent with TAO diagrams and therefore with many of the language concepts we have been considering? We suggest that it is.

Whenever a new concept or methodology must be developed, the design team must go into a data gathering and sharing mode of operation. The literature and company files have to be scoured and the relevant information retrieved, interpreted and shared. We suggest this activity is a form of model building, where the model is of the information being gathered. Adding structure to the information, where analogies, similarities and equivalences are noted, is a form of interpreting the information for the problem at hand. We suggest this structuring can be done using concepts we have just examined. One could set up a *modelior* an article

MODEL Smith&Jones88 REFINES article.

Keywords could be modeled through an IS-A link;

distillation IS-A keyword

might be stated. One could label concepts with a statement of the type

Concept-1: this is a concept.

With this type of formal structuring, information in an article would be in a form that could be formally linked. One could note a relationship between two articles by creating a model of the form

MODEL Sj88-dt82

sj88 IS-A Smith&Jones88

dt82 IS-A Davis&Tims82

sj88.Concept1, dt82.Concept5 ARE-THE-SAME.

One would in this structuring be creating a model of the information. By asking about one article, one **would be lead through** equivalencing of concepts and/or keywords to other related articles. These articles could be formally included as references within aspect and operator models, again though a statement of the type

ref1 IS-A sj88;

Thus one could provide a trail through the relevant literature that could be mechanically searched, perhaps even finding related aspects and operators in a library through the literature referenced by both. Of interest would be the degree that connections among such references might be automatically generated, especially by discovering common or related keywords being used. A software system where this type of *modeling* of information is possible is the HyperCard programming system written for the Macintosh Computer.

CONCLUSIONS

In this article we have examined how one might approach the development of future generations of tools to aid the design process. We proposed one model for structuring the design process, the **TAO** diagram. We then examined issues of providing tools to support the rapid creation of such diagrams. We suggested that certain language concepts in a modeling language, ASCEND, might **prove** important in structuring such tools.

List of Figures

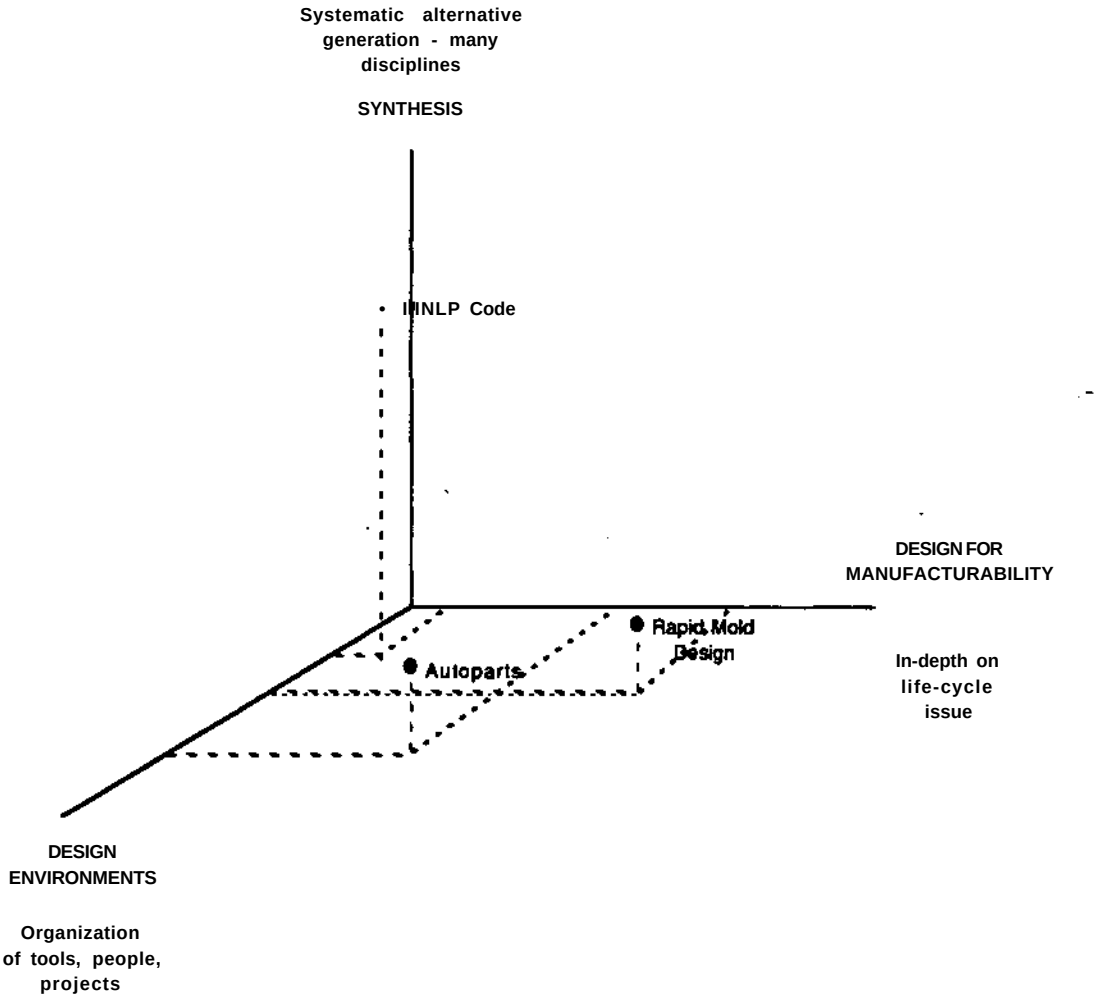


FIGURE 1 LABORATORIES IN ENGINEERING DESIGN RESEARCH CENTER. Three projects illustrate how each in fact deals with Issues from all the laboratories.

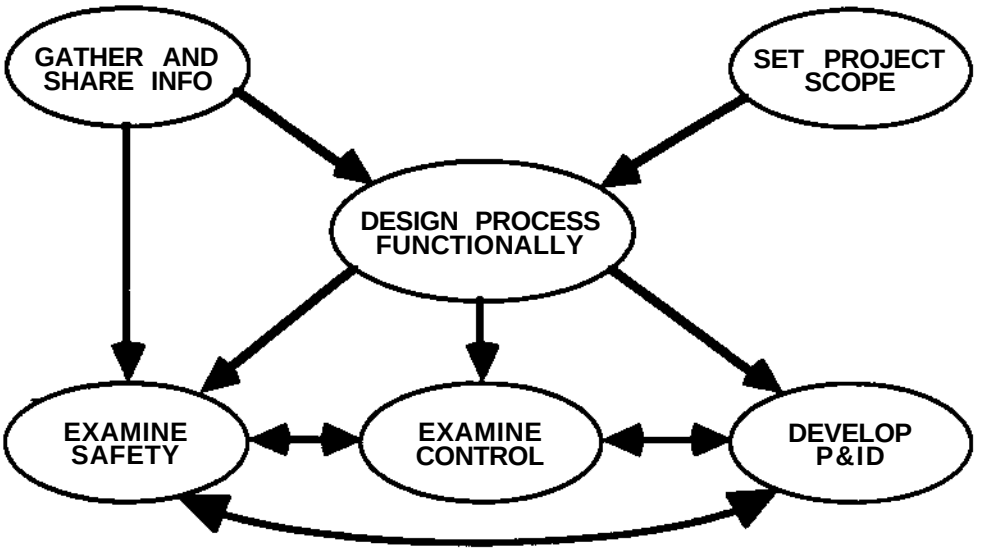


FIGURE 2 PROJECT DECOMPOSITION INTO LOOSELY COUPLED, PARTIALLY ORDERED TASKS

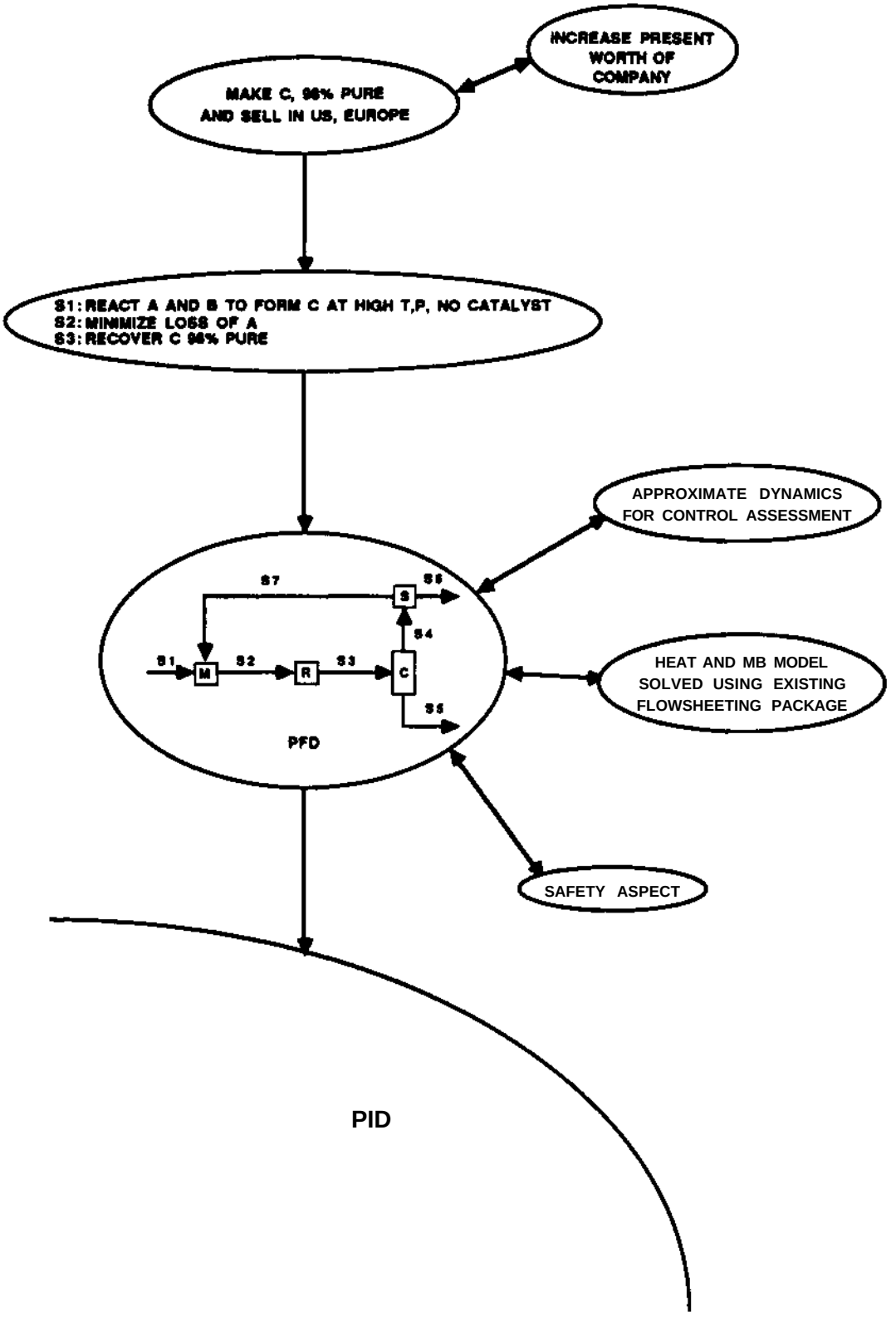


FIGURE 3 EXAMPLE TAO NETWORK FOR PROCESS DESIGN

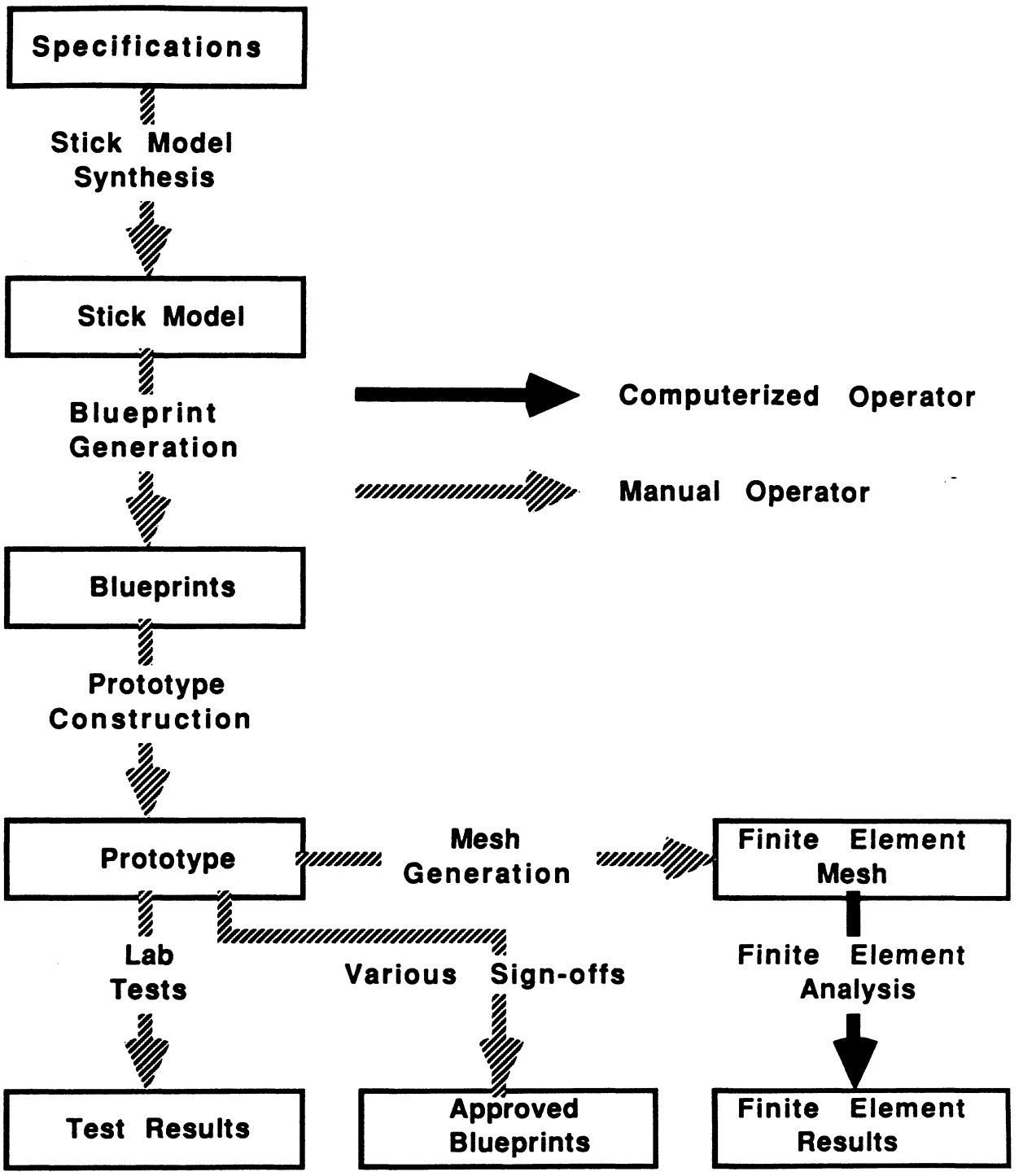


Figure 4: TAO Diagram for Existing Window Regulator Design Process

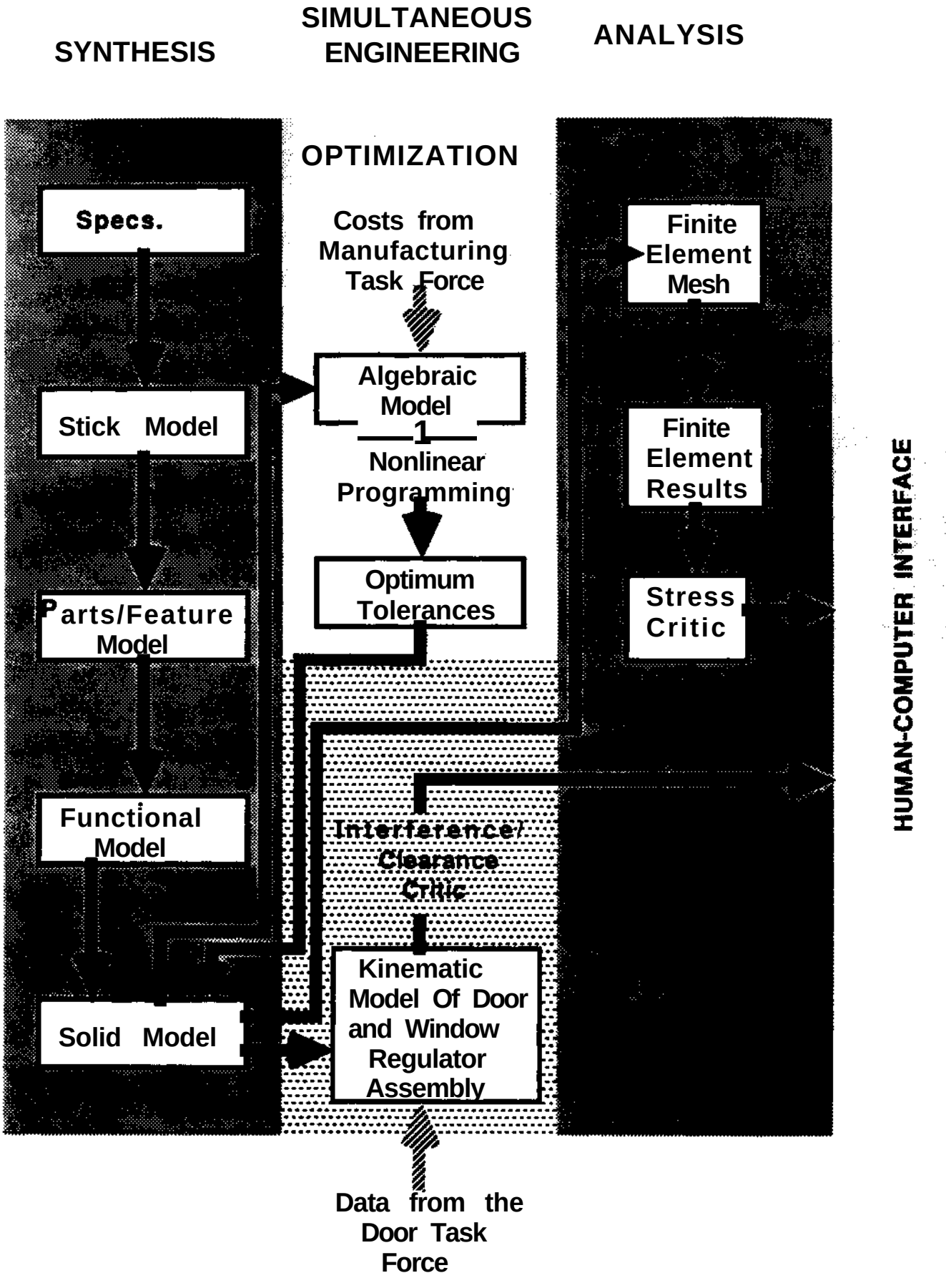


Figure 5: TAO Diagram for Revised Window Regulator Design Process

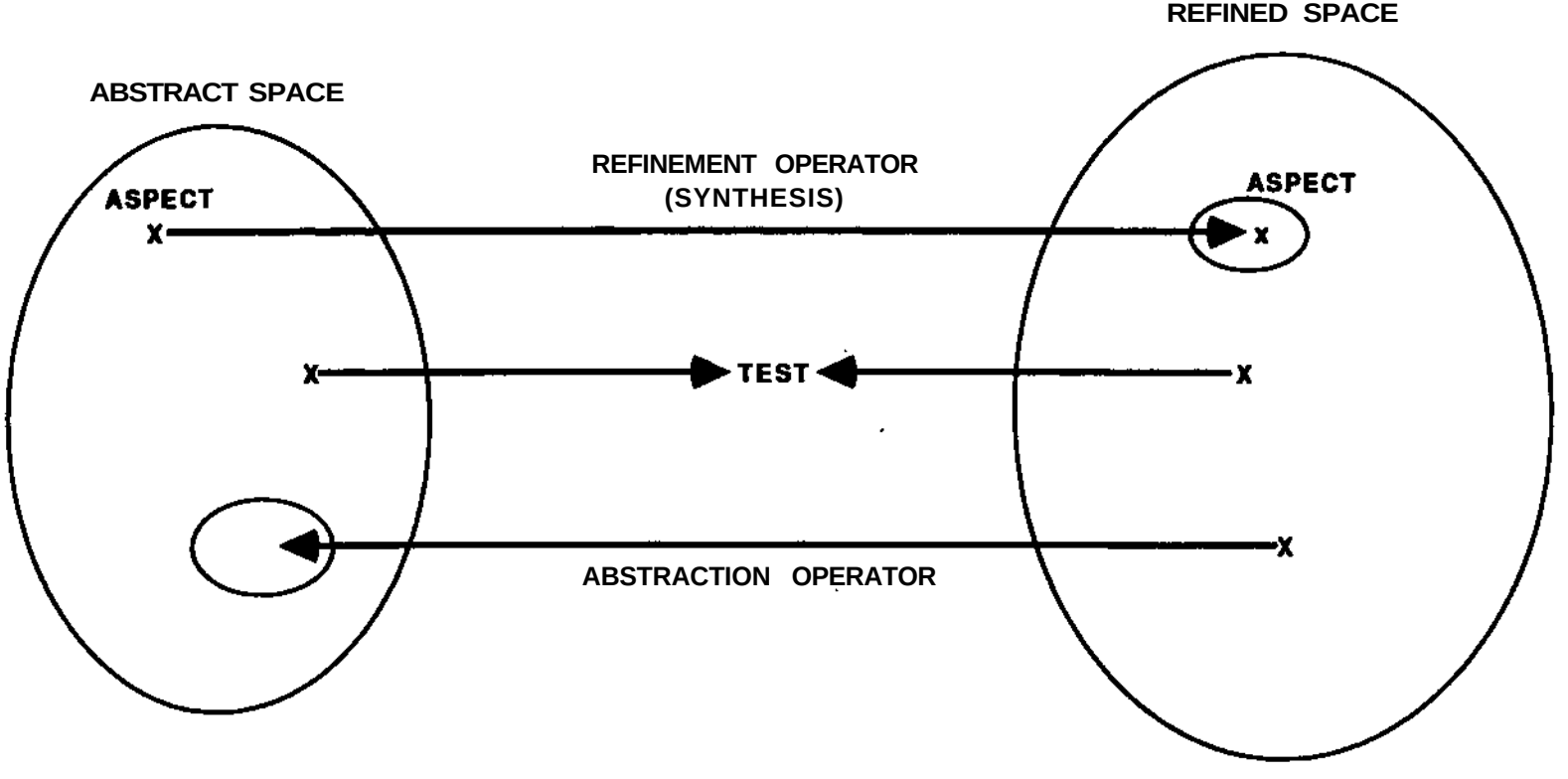
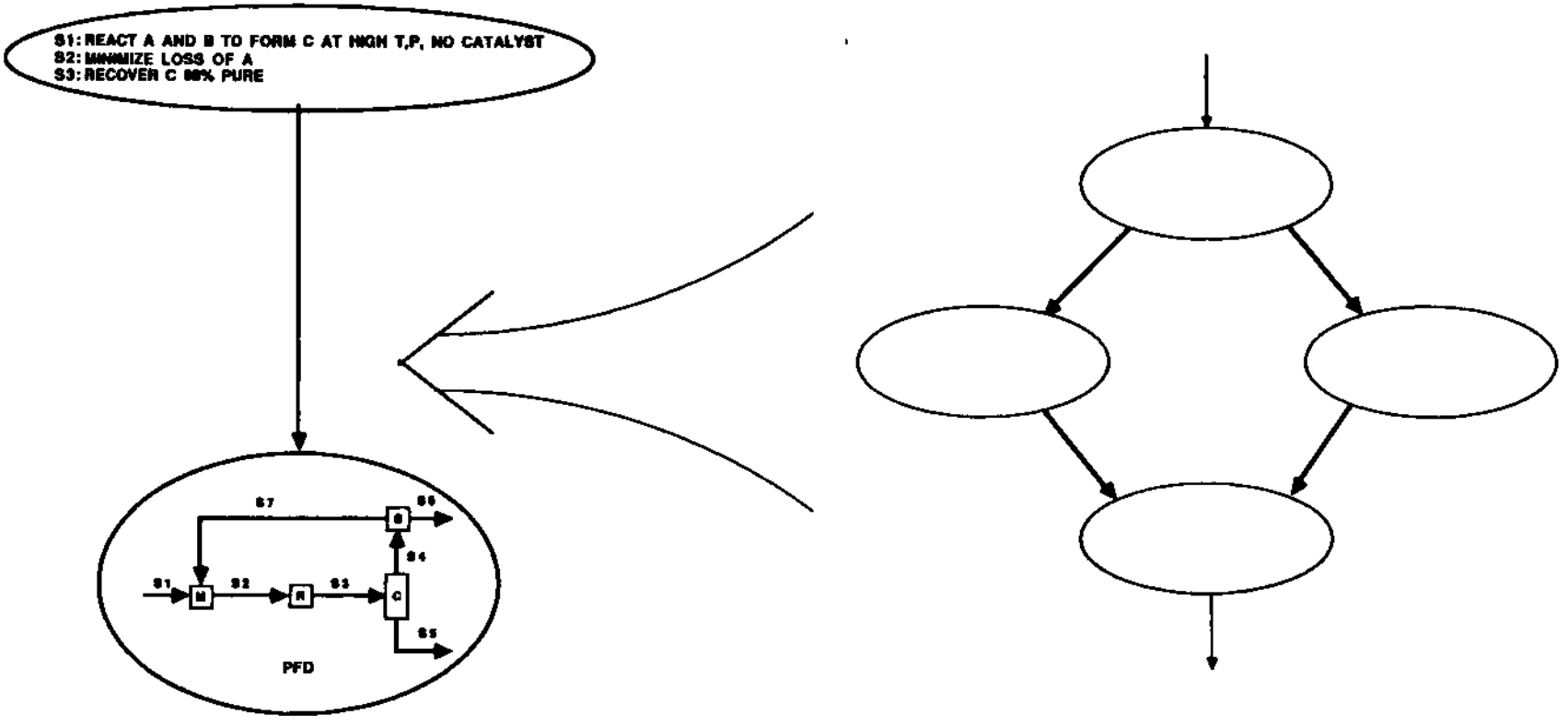
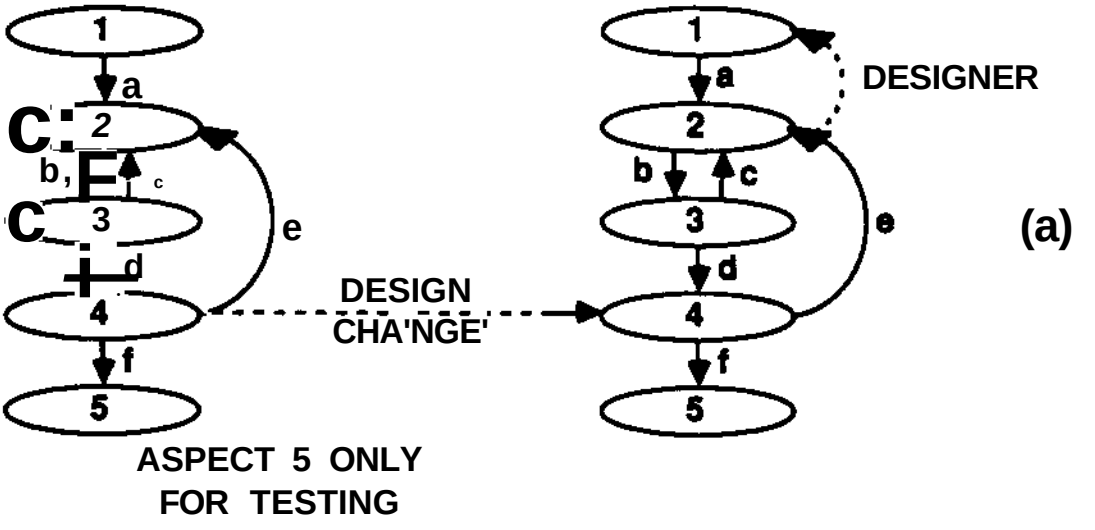


FIGURE 6 AN ABSTRACT VIEW OF TAO NETWORKS


S1: REACT A AND B TO FORM C AT HIGH T,P, NO CATALYST
S2: MINIMIZE LOSS OF A
S3: RECOVER C 99% PURE
S7
S6
S4
S1
S2
S3
S5
M
R
C
B
PFD

[illegible]



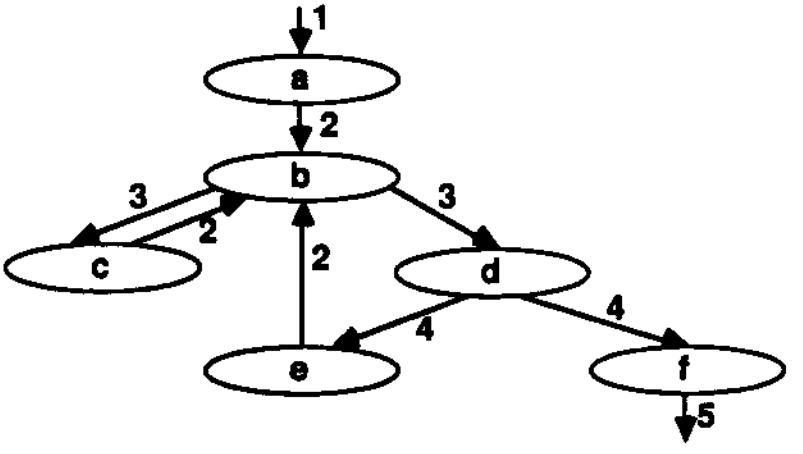


FIGURE 8 **PERSPECTIVES OF TAO DIAGRAMS**
(a) TAO diagram - activity on edge network
(b) TAO diagram - activity on node network

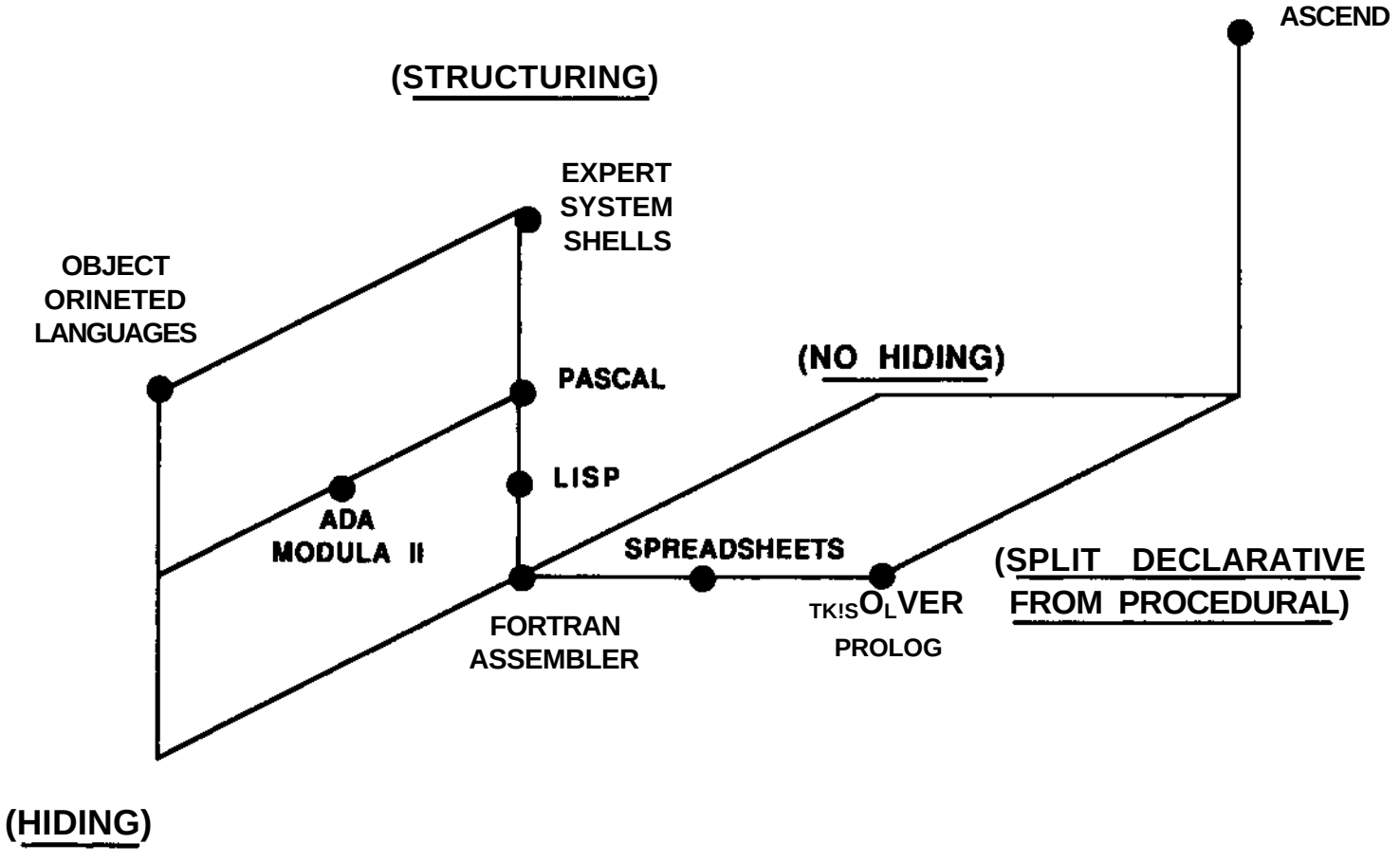


FIGURE 9 A MORPHOLOGY OF COMPUTER LANGUAGES

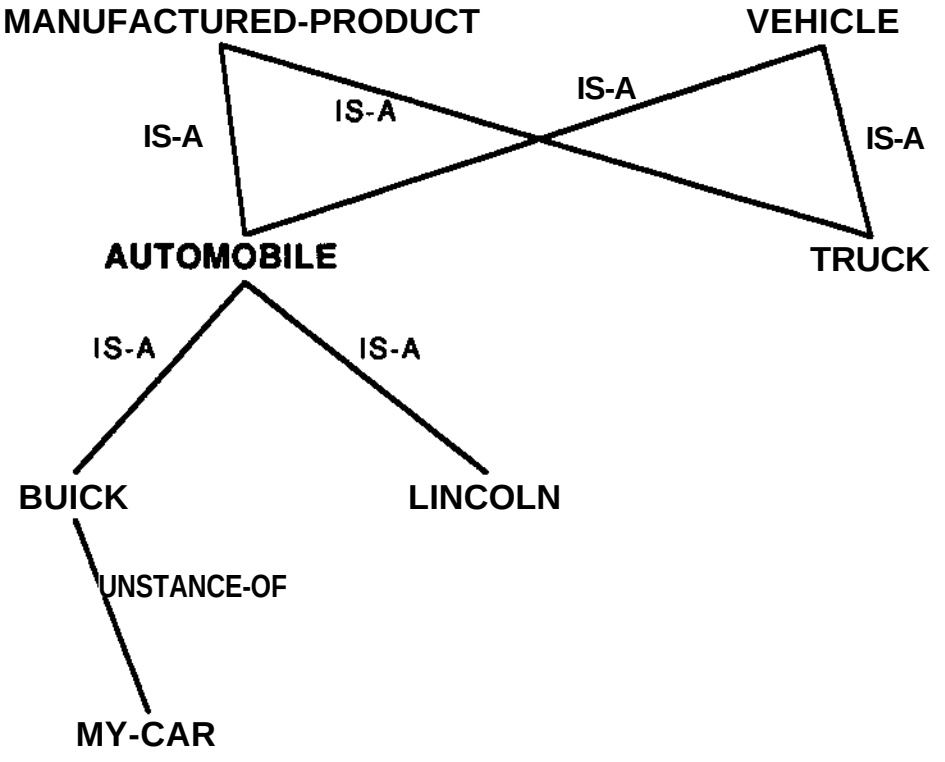
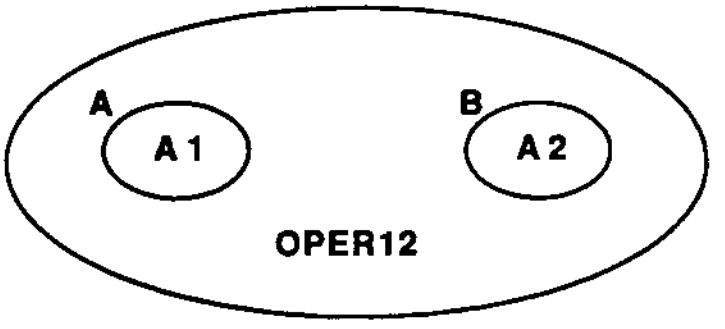


FIGURE 10 EXAMPLE CLASS DEFINITION AND INHERITANCE STRUCTURE

```
MODEL VEHICLE;
  Q IS-A ......

END;
   ***
MODEL MANUFACTURED-PRODUCT;
  A IS-A ......

END;
   ***
MODEL AUTOMOBILE REFINES VEHICLE;
  MP IS-A MANUFACTURED-PRODUCT;
  MP.A + Q = 7.2

END;

                etc.
```

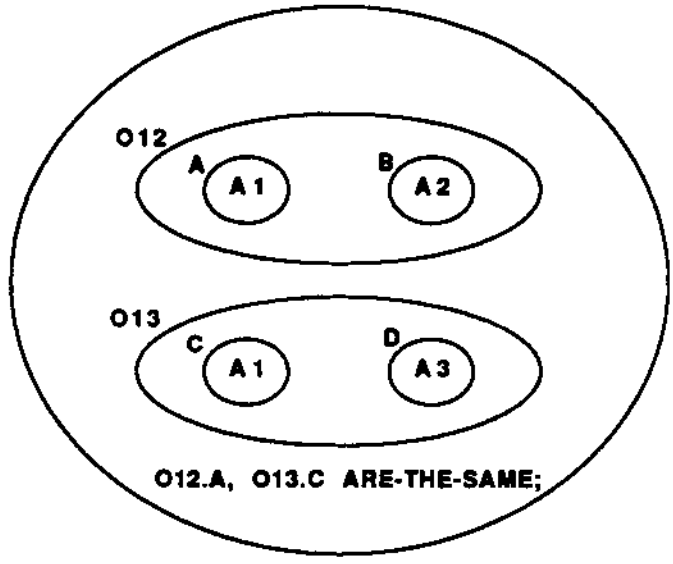
FIGURE 11 ASCEND MODEL FOR STRUCTURE IN FIGURE 10

```
MODEL  A1;

END;
   ***
MODEL  A2;                                    (a)

END;(a)
   ***
MODEL OPER12;
  A IS-A A1;
  B IS-A A2;
    ** CODE REFERENCING VARIABLES IN A AND B
  A.VAR1 ...
  B.VAR3 ...
END;
```



(b)

FIGURE 12 ASCEND-LIKE MODELING OF TAO'S. (a) is the code, (b) shows how OPER12 "owns" two aspects A and B of type A1 and A2 respectively.



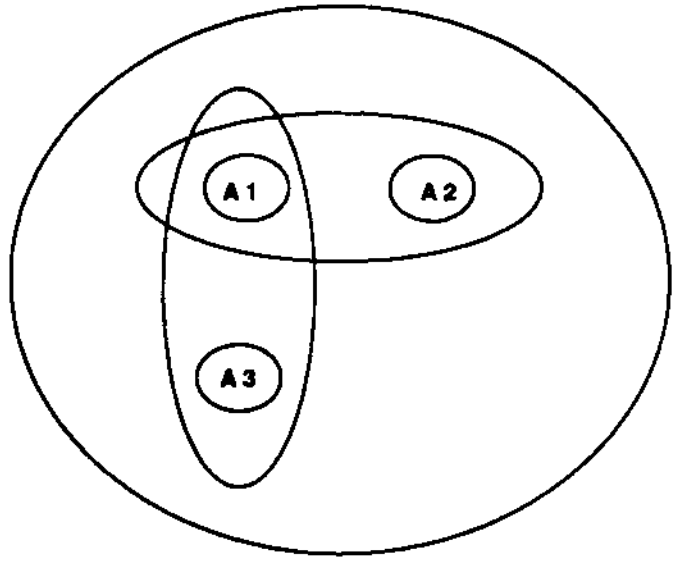


FIGURE 13 WIRING A TAO NETWORK TOGETHER WITH EQUIVALENCING