Proceedings of the 1985 Winter Simulation Conference
D. Gantz, G. Blais, S. Solomon (eds.)

CONCEPTS FOR KNOWLEDGE-BASED SYSTEM DESIGN ENVIRONMENTS

Jerzy W. Rozenblit
Dept. of Computer Science and Engineering
Oakland University
Rochester, Michigan 48063

Bernard P. Zeigler
Dept. of Electrical and Computer Engineering
University of Arizona
Tucson, Arizona 85721

ABSTRACT

The paper sets up a conceptual framework for constructing knowledge-based, computer-aided environments for system design. The framework is based on the formal structures underlying the expert system design methodology being developed by Zeigler [18], namely that of the system entity structure and experimental frame. The system entity formalism is employed to structure the family of design configurations. The rules for design model synthesis are generated by pruning the design entity structure with respect to generic experimental frames [13] that represent the design objectives. This leads to a methodology for design of system design environments which recognizes three primary relationships of the application domain that must be modelled: the decomposition hierarchy (of the system being designed), the taxonomic structure (determining the design alternatives), and the coupling constraints (restricting the combinations in which components can be synthesized into the target system).

1. SYSTEM DESIGN AND MODELLING ENTERPRISE - SYNERGIES

The process of design is a transformation of a designer's ideas and expertise into a concrete implementation. This process is driven by the design requirements provided by the client and the available technology. The growing complexity of systems being designed has strongly influenced research efforts in constructing computerized support environments for assistance in the design process [4,5,11,12,15].

Our primary goal in this paper is to embed system design within the multifacetted modelling framework [1,8,16,17] and thus provide a systematic design methodology supported by adequate formal structures. We shall argue that such an approach is amenable to computerization and direct application of expert systems and AI techniques. As illustrated in Figure 1, system design is brought into the multifacetted framework, with the design process being supported by the modelling and simulation techniques, in the following contexts:

a.) Modelling is a creative act of individuals using basic problem-solving techniques, building conceptual models based on knowledge and perception of reality, requirements and objectives of the modelling project. Thus, considering models as design "blueprints" we establish a direct relationship with the modelling enterprise.

b.) By providing mechanisms for model decomposition, hierarchical specification and aggregation of partial models, the multifacetted modelling fully responds to the needs of the design of large scale systems.

c.) By providing a spectrum of performance evaluation methods including the trade-off measurements and evaluation of multi-level, multi-component, hierarchically specified models, our framework allows the designer to describe the attributes of designs in comparative measures. This leads eventually to the choice of the best design with respect to performance measures under consideration.

d.) The representation schemes offered by the multifacetted methodology are well structured and have formalized operations that can exploit such structures. This significantly reduces the effort of designing expert environments for a given problem domain.

With the above issues in mind, we shall present concepts for constructing knowledge-based design environments. The two key formal objects in our approach are the system entity structure and the generic experimental frame. The entity structure is based on a tree-like graph encompassing the system boundaries and decompositions that have been conceived for the system. As we shall describe it in detail in Section 3 the entity structure formalism is a knowledge representation scheme that facilitates expressing the decomposition hierarchy, the taxonomy of the objects it represents, and the coupling constraints on the ways in which system components identified in the decomposition hierarchy can be coupled together.

The generic experimental frame is a structure that represents a set of design objectives in the form of standard variable types. Such standard variable types express measures of input/output performance, utilization of resources, reliability assessments etc. In outline, these two structures play the following role in our design framework:

* the system entity structure is a basic means of organizing a family of possible configurations of the system being designed.

* the objectives and requirements of the design project induce appropriate generic experimental frames.

* the design entity structure is pruned with respect to the generic frames. This results in a family of design configurations that conform to the design objectives.

* the pruned substructures serve as skeletons for generating rules for synthesis of design models.

* resulting models are evaluated in respective experimental frames and the best design models are chosen on the basis of such evaluations.

We shall provide a detailed exposition of this approach in the ensuing sections. However, let us

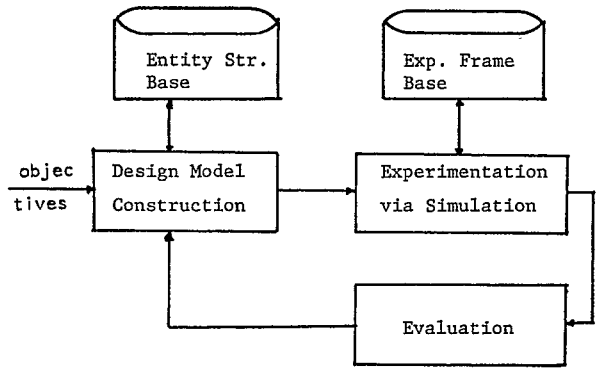

Fig. 1 System Design in the Multifacetted Modelling Framework

first briefly characterize the major steps of the design process as we perceive it in the context of this paper.

2. CHARACTERIZATION OF THE DESIGN PROCESS

The term system design will denote in our framework the use of modelling and simulation techniques to evaluate the proposed operation of the system that is being designed. As opposed to system analysis where the model is derived from an existing real object or phenomenon, in system design the model comes first as a set of "blueprints" from which the system will be build, implemented or deployed [2,19,20]. The blueprints might take several forms. They could be simple descriptions, a set of equations or a complex computer program. The task of system design viewed in this perspective is to create and study models of designs before they are physically implemented.

To characterize the design process we adopt the results of our previous studies [12] which we summarize as follows: the design procedure is a series of successive refinements comprising two types of design activities. The first type concerns the transitions between the so-called design levels. The second type defines a set design actions associated with a given design level. The design levels are successive refinements of the decomposition of the system under consideration. The first, and thus the most abstract level, is defined by the behavioral description of the system. Subsequently, the next levels are defined by decomposing the system into modules, and applying the decompositions to such modules until the subsystems are not further decomposable. Thus, the atomic system components are represented at the lowest level of the design hierarchy.

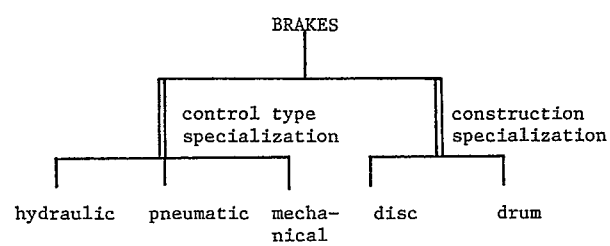

Fig. 2 Representation of Taxonomic Relations in the System Entity Structure

With each design level we associate a set of horizontal activities such as: requirements specification, system functional specifications, modelling, evaluation and choice of design alternatives via simulation studies. The design should proceed along both axes of the above characterization. The designer should be able to derive complete specifications at and design models at each level, he should be able to validate and verify the resulting system and its alternatives with the help of analytical and/or simulation tools. Transitions between the design levels must be possible and easy to perform.

A detailed look into all the facets of the design process verifies the need for high-level tools to support the design activities at all levels and phases. The architectures for such tools have been studied are presented in the literature [3,6,7]. However, the proposed solutions lack an underlying theoretical framework that permits a uniform treatment of design at different levels by providing concepts like structure and behavior, and allows for individuality of detail at each level.

The above orthogonal characterization of the design process has been successfully applied to define hardware design support systems [12,15]. While we do not attempt to further refine the definition of design, nor describe all its phases in detail, we shall show how the modelling techniques and its formal objects can support the expert design environments.

3. ENTITY STRUCTURING FOR REPRESENTATION OF DESIGN HIERARCHIES

In this section we present the formal concepts for representing and integrating the possible design alternatives that may be conceived for a given project. We argue that the system entity structure should be an underlying object used in the construction of the expert system design environments.

To appropriately represent the family of design structures we need a structure that embodies knowledge about the following three relationships: decomposition, taxonomy, and coupling. By knowing about decomposition we mean that the structure has schemes for representing the manner in which an object is decomposed into components, and can operate on, and can communicate about such schemes.

By taxonomic knowledge, we mean a representation for the kinds of variants that are possible for an object, i.e., how they can categorized and sub-classified. For example the structure could know that transmissions are automatic or manual, and that the latter can be of the four-speed or five-speed variety.

Our primary objective is to construct models of the system being designed in order to evaluate them with the help of simulation studies and select the best design alternative. To construct a model, the components of a decomposition must be coupled together. Thus, the third kind of knowledge that our structure for representing the design architectures should have is that of coupling relationships.

The methodology for constructing an expert system design environments will base itself on codifying appropriate decompositions, taxonomic and coupling relations. In other words we seek to model the expert's knowledge about the design domain by finding pertinent decompositions of the domain, the possible

variants that can fit within these decompositions, and the constraints on how the components of the decompositions can be coupled together. This will constitute the so-called declarative knowledge base. Beyond this, we should provide the procedural knowledge base in the form of production rules which can be used to manipulate the elements in the design domain.

The rationale behind such an approach is two fold. In the first place, we identify the decomposition, taxonomic, and coupling representation as knowledge which enables the structure to communicate about its objects. But more that that, we propose that the rules themselves can be much better designed once good representations for the above relations have been identified. There are several reasons for advancing this proposition. We shall show that large segments of rules can be generated almost automatically from the knowledge structures: in the absence of such structures these rules would have to be generated one-by-one in an often ad hoc manner. To the extent that most of the rules can be generated automatically, we can then focus our attention on the exceptions. Ultimately, rule development should then reduce to the creative effort required to deal with the irreducible idiosyncracies of the problem domain.

The formal object that meets the requirements stipulated above is the system entity structure whose definition follows.

Definition (Belogus [1], Zeigler [18]).

A system entity structure is a labeled tree with attached variable types. When a variable type V is attached to an item occurrence I, this signifies that a variable I.V may be used to describe the item occurrence I. The structure satisfies the axioms of:

* alternation entity/aspect
* entity/specialization
* strict hierarchy
* inheritance: birth, life
* multiple entity

and allows for the following operations:

* naming scheme
* generation of distribution /aggregation relations
* transformations to taxonomy free form
* pruning
* attachment of constraints to aspects

For a more detailed formal treatment of the system entity structure we refer the reader to [17]. Here we shall indicate how the discussed knowledge representation scheme is realized by the structure.

We begin by characterizing the taxonomic representation scheme. An entity may have several specializations; each specialization may have several entities. The original entity is called a general type relative to the entities belonging to a specialization, which are called special types. Since each such entity may have several specializations, a hierarchical structure results, which is called a taxonomy.

Figure 2. depicts the entity structure in which the entity BRAKES has been given two specializations, control_type and construction_type. A salient feature is the alternation property which requires that entities and specializations alternate along any path from root to leaves. Specializations have independent existence just as entities do. A specialization may occur in more than one location; whenever it occurs it carries with it all its attributes and substructures. Of course it may not be meaningful to attach a particular specialization to a particular entity.

Hierarchical decomposition is in many ways analogous to the specialization hierarchy just discussed. The alternation property now requires alternation of entities and aspects. An aspect is a mode of decomposition for an entity just as a specialization is a mode of classification for it. There may be several ways of decomposing an object just as there may be several ways of classifying it. Formally, aspects and specializations are quite alike in their behavior (but not in their interpretation); they each alternate with entities, but cannot be hung from each other. A special type of decomposition called a multiple decomposition facilitates flexible representation of multiple entities whose number in a system may vary. Specializations of an entity can be mapped into corresponding aspects of its multiple entity. Such transformations are discussed extensively in [17,18].

Our approach to expressing the coupling constraints is as follows: we apply the mapping to remove the specializations to obtain an entity structure containing only entities and aspects. Now we imagine that we are synthesizing models by working our way down the entity structure selecting a single aspect for each entity and zero or more entities for each aspect. Such a process is called pruning of the entity structure. We shall describe it in detail in the next section. The coupling constraints we wish to express must then be associated with aspects since they represent the decompositions from which we shall choose when pruning. Moreover, we must associate a constraint with an aspect which scopes all the entities that are involved in that constraint. What is more, this aspect should be minimal in the sense that there is no other aspect that lies below it in the entity structure which also scopes all the entities involved in the constraint.

4. ENTITY STRUCTURE PRUNING FOR GENERATION OF DESIGN MODEL STRUCTURES

We are now ready to discuss the two most essential concepts in our proposed design framework. The first concept concerns the generic frame-based pruning of the system entity structure.

As we have already pointed out the first and crucial step in the design process is to determine the set of all possible configurations of the system being designed. The system entity structure is the basic means of organizing such a family. The entities represent system components while aspects allow the designer to form various alternatives for decompositions of components. Thus, the system entity structure is a set of substructures from which design models can be constructed. To select such substructures we must meaningfully prune the entity structure.

Recall that our design framework requires that the design models (or more precisely, the structures that are used to construct them) accommodate the design objectives and requirements. The generic experimental frame which we shall define formally shortly, serves as a means of expressing such objectives. Thus, by pruning the system entity structure with respect to generic frames we derive the following benefits:

1.) In terms of the contribution to the design process

a.) a generic frame extracts only those substructures which conform to the design objectives. Thus, a number of design alternatives may be disregarded as not applicable or not realizable for a given problem.

b.) partial models of the design can be formulated and evaluated. This may significantly reduce the complexity which would arise if we had to deal with the overall design model. The generic frame concept may thus be viewed as an object that partitions the system entity structure into design-objective related categories.

c.) the evaluation of design models constructed from the pruned substructures is performed in corresponding experimental frames. Such frames are generated by instantiating the generic frames used to prune the system entity structure. Hence, an automatic evaluation procedures could be employed in the design process. (For details see [11,13])

2.) In terms of facilitating the pruning process itself, generic frames automatically determine:

a.) the aspects that are selected for each entity

b.) the depth of the pruning process

c.) the descriptive variables of components

Having presented the benefits afforded by the generic frame concept let us now give its formal definition and define the procedure to prune the design entity structure.

4.1 Generic observation frame

The concept of the generic experimental frame has been originally developed for the purpose of generating the experimental frames in simulation [13]. The generic frame is defined by means of unqualified generic variable types that correspond to the objectives of a simulation study.

In system design context it is enough to restrict the generic frame to the so-called generic observation frame which we define as follows:

$$GOF=\{IG, OG\}$$

where IG denotes the set of generic input variable types, and OG is the set of generic output variable types.

By defining the generic observation frame in the above manner we limit its role to representing the behavioral aspects of design objectives and requirements. If we were to construct models of designs based on the structures pruned in the generic observation frames we could only implement the behavioral specifications of designs. There are also objectives that concern the structural aspects of the project under consideration. Therefore, as we shall see in the next section, it will be necessary to augment the design model construction with a process that we term synthesis rule generation in order to realize the structural constraints.

Let us however return to the pruning procedure and define how a generic observation frame generates all the system entity substructures that accommodate behavioral design objectives.

4.2 Observation frame-based pruning of the design entity structure

Given the generic observation frame we seek to extract all the substructures that accommodate the input and output variable types present in that frame. Let V_{GOF} denote the set of input and output variable types that belong to the generic frame GOF. A copy of this set called CV_{GOF} is created to control the pruning. The frame based pruning can be defined by the following algorithm:

```
Procedure Prune(E_j, CV_GOF, V_GOF);
 { This procedure prunes the system entity structure
 and returns
   the model structures that accommodate the generic
   observation
   frame GOF. Multiple occurrences of a frame variable
   type are
   permitted in the model structure }

begin
  for each aspect A_i ε E_j do
    for each entity E_k ε A_i do
      begin
       CV_GOF := CV_GOF - {v_k } where v_k ε V_GOF (is a
       frame variable type), and v_k is present in the
       entity E_k {update the current set of frame
       variables by subtracting the types already
       present in the entity substructure}
       Attach E_k with all its variables as a child of

       TE_j and attach the coupling constraint of the
       aspect A_i to TE_j { TE_j denotes the root of the
       current substructure of the model structure }
       If at least one entity at the current level
       contains a variable type which is present in
       V_GOF then mark this level in the model
       structure as the last level at which frame
       variables are present
      end;
     for each E_k ε A_i such that E_k has aspects do
       Prune(E_k, CV_GOF, V_GOF);
   if CV_GOF is empty then
      begin
        create a copy of the current model structure
        without the last level entities {this copy
        will serve as a basis for pruning in the next
        aspect A_i+1};
        store the current model structure TE_j, however
        eliminate all the entities that appear below
        the level marked as the last level with frame
        variable type occurrence;
      end;
   Update the current model structure TE_j by cutting
   off the last level entities { the entities in the
   next aspect may now be attached to this current
   tree};
  end;
end; {of Prune}
```

We have not specified at which level of the system entity structure the pruning should begin. This allows the modeller for a flexible choice of the model boundaries. We should indicate however that due to

recursion based on the entities the procedure must be initially called as follows:

a.) in the system entity structure choose the entity E_i that represents the model you intend to evaluate (this entity will label the root of the model structure TE_i).

b.) create a dummy entity DE (with no variables) with a dummy aspect DA in which E_i is a subentity of DE.

c.) call Prune(DE, CV_{GOF}, V_{GOF});

After the procedure has been executed we have to eliminate DE from all the model structures. Notice that the purpose of CV_{GOF} is merely to check whether the currently traced substructure has already accommodated the generic frame or not.

We have already indicated that the procedure Prune generates a set of design model structures in the form of decomposition trees [17]. Each such structure accommodates the generic observation frame GOF and constitutes a skeleton for a hierarchical model construction.

5. ENTITY STRUCTURE-BASED SYNTHESIS RULE WRITING

The pruning process described in the foregoing section restricts the space of possibilities for selection of components and couplings that can be used to realize the system being designed. Thus we can assume that design may now be reduced to the synthesis problem. Synthesis involves putting together a system from a known and fixed set of components in a fairly well-prescribed manner. In the synthesis problem, we are modelling a rather restricted design process, one amenable to automation by extracting concepts and procedures from experts' knowledge and experience, augmenting them and molding them into a coherent set of rules. The rule development methodology that we propose for such a modelling enterprise is as follows:

* Restrict the design domain by pruning the design entity structure in respective generic observation frames.

* Examine the resulting substructures and their constraints. Try to convert as many constraint relations as possible into the active form, i.e. into rules that can satisfy them. For those that cannot be converted into such rules write rules that will test them for satisfaction.

* Write additional rules, modify existing ones, to coordinate the actions of the rules (done in conjunction with the selected conflict resolution strategy).

We shall proceed to discuss this methodology in greater detail.

5.1 Types of Constraints and Their Conversion to Active Form

In a synthesis problem, several kinds of constraints may come into play. Objectives-derived constraints formulate the objectives that we have in mind for a specific system being synthesized. For example it must be able to achieve certain levels of performance, exceed certain levels of accuracy, etc. In addition to such specific design objectives, industry wide, or governmentally imposed, standards place performance constraints that all products of the kind being built must satisfy. Standards may have been put into place to assure a minimum level of safety or to facilitate interchangability of parts constructed by different manufacturers. Resource constraints arise from the fact that resources available to construct the system may be limited and costly. Resources may be replenishable, such as electric power or non-replenishable such as construction material. Generally, we want to minimize the use of the resources, and most definitely, we cannot allow the synthesis process to use more of a non-replenishable resource that is available. Natural constraints arise from the limitations imposed by the laws of nature. Syntactic constraints relate to the order in which components may be coupled together; they may be imposed arbitrarily to reduce the space of possible configurations or may be formulated to ensure satisfaction of more fundamental performance, natural or other constraints.

Assuming that the synthesis problem is appropriate for expert system design, there are known actions that can be taken to try to satisfy the performance constraints derived from the objectives and imposed standards. Indeed, an expert's procedural knowledge represents efficient procedures that are likely to achieve the goals and subgoals that arise in attempting to meet the performance requirements. The pruning process described in the previous section is an example of such an action. However, meeting the requirements is subject to the given resource, natural and syntactic constraints. Thus, we see a second kind of constraint classification emerging: some constraints are convertable to active form, i.e., they can be converted into actions intended to satisfy them. Other constraints are inherently passive, they do not motivate or guide action, they sit there demanding satisfaction. The question that now begs to be addressed is: assuming that it is possible, how can we convert a constraint to active form? We conceive of the synthesis problem as a search through the search space, the set of all pruned design structures. These are candidates for solution to the problem. Our set of rules will take us from an initial state in this space to a goal state. The search should proceed by generating successive candidate structures in an efficient manner.

We can assume that for each active constraint we have a means of generating such candidates to test against the constraint. Call such an operator NEXT_IN_Ci.

The passive constraints have no corresponding operators and thus we can only test for their satisfaction. Failure causes backtracking if a state has been reached for which none of the operators can be applied. Instead of applying an operator and then testing if it has consumed more than what remains of an available resource, we can try to inhibit the application of operators that would bring about the resource depletion.

Let Con be a constraint that we wish to pretest. An operator, NEXT_IN_Ci will map a state s into the region satisfying Con if, and only if, Con(NEXT_IN_Ci(s)). To allow the operator to be applied safely we need to define applicability predicate, Ai such that:

Ai(s) if, and only if, Con(NEXT_IN_Ci(s))

Thus the canonical rule scheme for the synthesis problem takes the form of Figure 3.

```
RC  If C satisfied on (state)
       then Output (state) as the solution

R1  If C1 is not satisfied
       A1 is satisfied
       then state:=NEXT_IN_C1(state)

 .......

Ri  If Ci is not satisfied
       Ai is satisfied
       then state:=NEXT_IN_Ci(state)

 .......

Rn  If Cn is not satisfied
       An is satisfied
       then state:=NEXT_IN_Ci(state)
```

Fig. 3 The canonical rule scheme for the synthesis problem

Having presented the formal structures for constructing the expert design environment let us gather the strands up and propose an architecture for such a system.

6. EXPERT ENVIRONMENT FOR DESIGN MODEL DEVELOPMENT

Given the aforementioned structures and procedures for pruning and rule generation we propose an expert system architecture for support of automatic development of design models.

As depicted in Figure 4. the data base of design objectives specifications is one of the major components of the system. The objectives drive three processes in our framework: retrieval and/or construction of the design entity structure, definition of generic observation frames, and generation of rules for model synthesis. The purpose of the system illustrated in Figure 4. is to analyze and integrate the relationships concerning the objectives specification base, the generic observation frame base, and the design entity structure. Such an integration should result in design models and ultimately in the formulation of an appropriate simulation experiment for a problem at hand [13]. This represents a great potential for the application of expert systems technology [9,14].

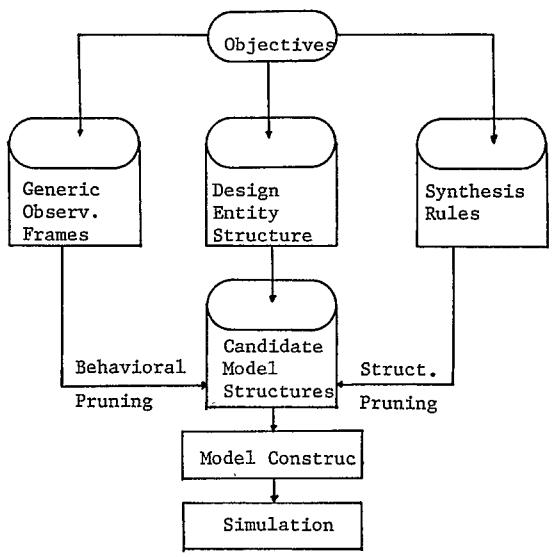

Fig. 4 Architecture for Expert System Design Environment

Let us summarize how such a system should operate. The behavioral aspects of the design objectives are expressed in terms of generic observation frames. Pruning the design entity structure in corresponding observation frames results in substructures conforming to the behavioral objectives.

The substructures are then tested for satisfaction of synthesis rules that are derived from the design structural constraints as presented in Section 5. Both, behavioral and structural pruning applied to the design entity structure should result in design structures that we term candidates for hierarchical model construction. The term candidates implies that some checks for consistency and admissibility (in the sense of conformance to the objectives) should be performed at this stage. If the candidate is inadmissible or no candidates can be obtained by pruning, the process should be reiterated with possible user intervention. The kinds of interventions we suggest are modifications or retrieval of the new system entity structure, enhancement of the generic experimental frame or modification of synthesis rules. The system should construct design models for the structures generated as a result of behavioral and structural pruning employing the multifacetted model construction methodology [17].

Let us now illustrate the concepts under discussion by presenting a simple example.

7. EXAMPLE - AUTOMOTIVE DESIGN

Assume that an automotive company is designing a new model of a passenger car whose fuel efficiency meets the standards imposed by the Department of Transportation. In the first stage of development, a design entity structure representing possible configurations for a car is proposed. Such an entity structure can take the form depicted in Figure 5. For the sake of brevity we present a rather simplified version of a car design structure with only three aspects that is: Physical Decomposition, Service Aspect, and Utility Specialization. Notice, that the Physical Decomposition Aspect of CAR can be hung from the entities Passenger Car and Truck (in Utility Specialization) in place of the *** symbol (Figure 5).

Given the entity structure we are now ready to derive generic observation frames. Notice that such a frame is explicitly stipulated in the given behavioral requirement i.e., the gasoline consumption aspect of the design. An appropriate generic set of variables that defines the frame is given below:

Generic Observation Frame: Gasoline Consumption

Input variables:
- fuel level
- speed

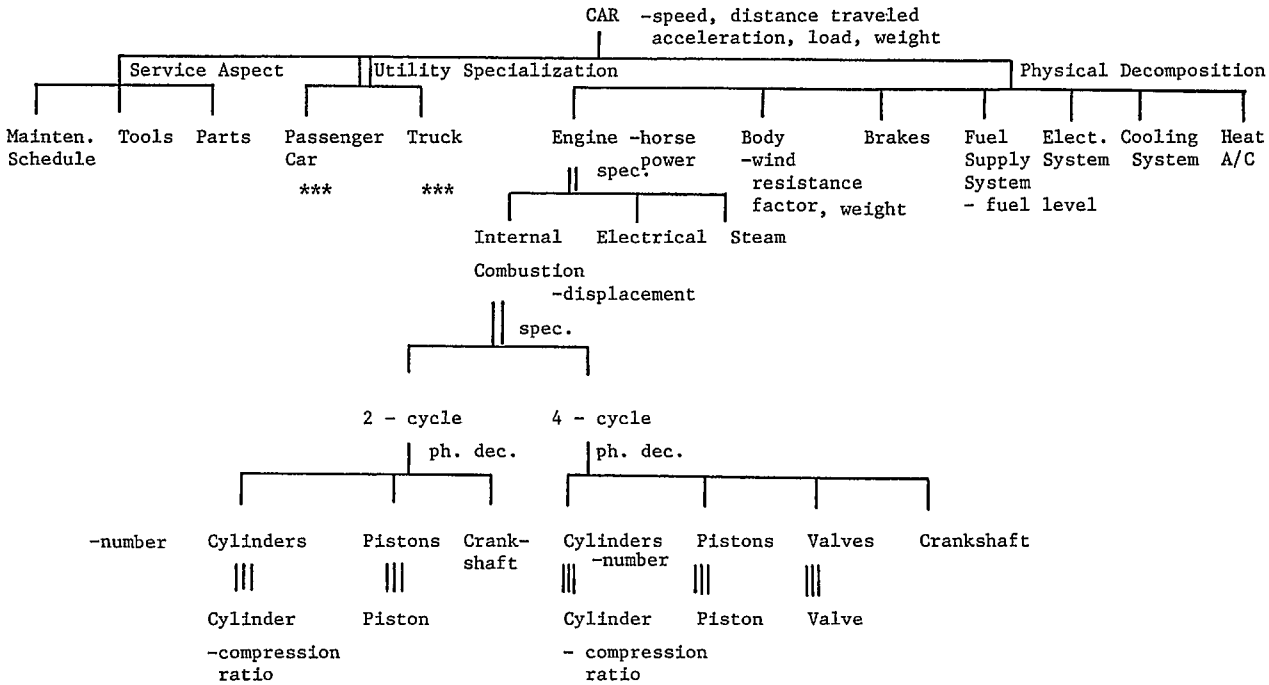

Fig. 5 System Entity Structure for the Car Design Problem

acceleration
load
horse power
displacement
compression ratio
wind resistance factor
weight

Output variables:
fuel level
distance traveled

Pruning the design entity structure of Figure 5. with respect to the frame "Gasoline Consumption" will result in the substructures of the Utility Specialization and Physical Decomposition aspect. Service aspect will be disregarded as irrelevant. At the lower level of the hierarchy both Electrical and Steam engines will be pruned out as they have no variable types present in our observation frame. To limit the design to a passenger car we restrict the design class by selecting the Passenger Car specialization. Another constraint that further limits the design space is a standard constraint imposed by Dept. of Transportation that prohibits the use of 2 - cycle engines in passenger cars. Thus, the pruned entity structure takes the form of Figure 6.

The general car design problem is now reduced to the synthesis of a passenger car with a 4 - cycle internal combustion engine. Let us formulate structural constraints and convert them into a production rule scheme.

In our formulation we shall synthesize a very coarse model of a car. We shall simply assume that a car results from a coupling of an engine and a body. The following factors play a major role in the synthesis process: first, we are restricting the number of passengers to 6. The measures of load and weight are then given by the relations: below:

LOAD = LOAD.FACTOR * PASSENGERS.VOLUME
BODY.WEIGHT= WEIGHT.FACTOR * BODY.VOLUME

Secondly, it will be necessary to synthesize an engine with enough power to set the car in motion. We assume that in order to increase the engine's power we can add cylinders in pairs. However, the number of cylinders cannot exceed 8. Adding a pair of cylinders also increases the volume of the engine i.e.:

ENGINE.VOLUME = CYLINDER.VOLUME * CYLINDERS.NUMBER

The constraints associated with the physical decomposition of the entity Passenger Car can be formulated as follows:

1.) BODY.VOLUME >= ENGINE.VOLUME + PASSENGERS.VOLUME
2.) ENGINE.POWER >= BODY.WEIGHT + MAXIMUM.LOAD
3.) BODY.VOLUME <= MAXIMUM.VOLUME (for a 6 passenger car)

The constraints associated with the Engine synthesis have the form:

4.) CYLINDERS must be coupled in pairs either in line or across from each other

5.) CYLINDERS.NUMBER $\in$ [2,8]

To convert the constraints to production rules we implement the canonical scheme given by Figure 3. As

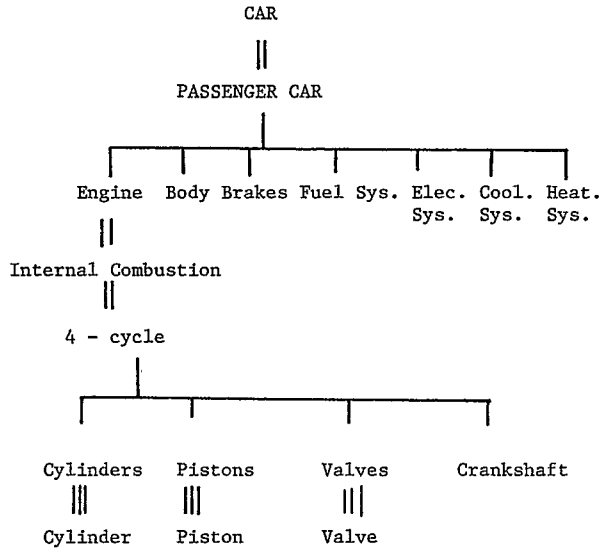

Fig. 6 Pruned Entity Structure for the Car Design Problem

in the general approach rule RC is the global constraint checker. Rules RC1 and RC2 are implemented as local constraint satisfiers for constraints 1 and 2. Note, that the resource constraints 3 and 5 have been formulated as pretests for applicability of the rules. The production rule scheme is presented below:

```
RC  if  ENGINE.POWER >= BODY.WEIGHT + MAXIMUM.LOAD
        BODY.VOLUME      >=      ENGINE.VOLUME +
        PASSENGERS.VOLUME

    then Print "Car Completed"

RC1 if BODY.VOLUME ≤ MAXIMUM.VOLUME  - 1 UNIT
       BODY.VOLUME < ENGINE.VOLUME + PASSENGERS.VOLUME

    then expand BODY.VOLUME by 1 UNIT
         update BODY.WEIGHT

RC2 if a pair of CYLINDERS is available
       ENGINE.POWER < BODY.WEIGHT + MAXIMUM.LOAD

    then add this pair of CYLINDERS to the ENGINE
         update ENGINE.VOLUME
```

Fig. 7 Production Rule Model for the Car Synthesis Problem

After a candidate structure that satisfies all the constraints has been found a design model of the car should be constructed and the observation frame "Gasoline Consumption" should be refined to an experimental frame [13]. Then, the model can be evaluated via simulation experiments as shown in Figure 8.

8. SUMMARY

We have attempted to outline a foundation on which the organization of the design process can be based. We envision a computer-aided expert design environment which internally represents the entity structures and generic observation frames, and has a means for dynamically manipulating these structures. The means are based on the procedures discussed above. Implementation of such a package, in all its generality, may be a long way off. However, specific parts of it, have already been implemented [10,21] and efforts are under way to further advance the theory of knowledge-based system design [12,18].

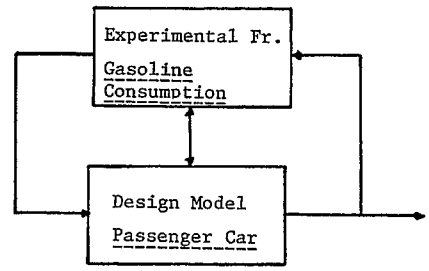

Fig. 8 Simulation Study for Design Evaluation

REFERENCES

1. Belogus D., "Multifacetted Modelling and Simulation: A Software Engineering Implementation", Doct. Diss., Weizmann Institute of Science, Rehovot, Israel, 1985

2. Elzas M.S., "The Use of Structured Design Methodology to Improve Realism in National Economic Planning", in Model Adequacy (ed. H. Wedde), Pergamon Press, London, 1982.

3. Fasang P.P., Ulrich P., Whelan M., "A Perspective on the Levels of Methodologies in Digital System Design, Proc. of the 1983 IEEE Conference on Computer Design

4. Gonauser, M., Sauer, A., "Needs for High-Level Design Tools", Proc. of the 1983 IEEE Conference on Computer Design

5. Gonauser, M, Kober R., Wenderoth W., "A Methodology for Design of Digital Systems and Requirements for a Computer Aided System Design Environment", IFIP WG 10.0, Sept. 1983

6. Javor, A. , "Proposals on the Structure of Simulation Systems", in:Discrete Simulation and Related Fields,(ed. Javor, A.), North-Holland, Amsterdam 1982.

7. Kober R., Wenderoth W., "Problems and practical experience in High-Level Design", Proc. of the 1983 IEEE Conference on Computer Design

8. Oren, T.I., Zeigler B.P. "Concepts for Advanced Simulation Systems", Simulation, 32,3 pp. 69-82, 1979

9. Reddy Y.V., Fox M.S., Husain N., "Automating the Analysis of Simulations in KBS", in Proc. of the SCS Multiconference, San Diego, January 1985.

10. Rozenblit J.W., "EXP - A Software Tool for Experimental Frame Specification in Discrete Event Modelling and Simulation", in Proc. of the 1984 Summer Computer Simulation Conference, pp. 967-971, Boston 1984.

11. Rozenblit J.W., "A Conceptual Basis for an Integrated, Simulation Based System Design", Doct. Diss. Dept. of Computer Sci., Wayne State University, Detroit, Michigan, 1985

12. Rozenblit J.W., "Structures for a Model-Based System Design Environment", Technical Report, Siemens AG, West Germany Munich, 1984 (internal distribution)

13. Rozenblit J.W., "Generating Meaningful Experimental Modules for Simulation Design", to appear in Knowledge-Based Modelling and Simulation Methodologies, ed. M.S. Elzas et. al., North Holland, Amsterdam, 1986

14. Shannon R.E., Mayer R., Adelsberger H.H., "Expert Systems and Simulation", Simulation vol. 44, number 6, June 1985.

15. Siewiorek D.P., Giuse D., Birmingham W.P., "Proposal for Research on DEMETER: A Design Methodology and Environment", Carnegie-Mellon University, Pittsburgh, PA, Jan. 1983

16. Zeigler B.P. "Structures for Model Based Simulation Systems", in: Simulation and Model-Based Methodology: An Integrative View, Springer-Verlag, New York, 1984

17. Zeigler B.P. "Multifacetted Modelling and Discrete Event Simulation", Academic Press, London, 1984

18. Zeigler B.P., "Expert Systems: A Modelling Framework", (in preparation)

19. Wymore, W.A., "A Mathematical Theory of Systems Design", Technical Report, University of Arizona, Tucson, 1980

20. Wymore W., "Systems Engineering Methodology for Interdisciplinary Teams", John Wiley, New York, 1976

21. Zeigler B.P., Belogus D., Bolshoi A., "ESP - An interactive Tool for System Structuring", Proc. of the 1980 European Meeting on Cybernetics and Systems Research, Hemisphere Press 1980

JERZY W. ROZENBLIT is currently a visiting assistant professor of computer science at Oakland University, Rochester, Michigan. He received his MSc degree in computer science from The Technical University of Wroclaw, Poland. In 1981 he joined Wayne State University in Detroit to pursue the PhD degree under the direction of Professor Bernard Zeigler. His research interests are in the area of modelling and simulation, system design and software engineering.

Jerzy W. Rozenblit
Dept. of Computer Science and Engineering
Oakland University
Rochester, Michigan 48063
(313) 370-2200

BERNARD P. ZEIGLER is a professor of computer science and engineering at University of Arizona. He is author of "Multifacetted Modelling and Discrete Event Simulation", Academic Press, 1984, and "Theory of Modelling and Simulation", Wiley, 1976. His interests include distributed simulation and expert systems for simulation methodology.

Bernard P. Zeigler
Dept. of Electrical and Computer Engineering
University of Arizona
Tucson, Arizona 85721
(602) 621-2434