ORSA Journal on Computing
Vol. 3, No. 4, Fall 1991

0899-1499/91/0304-0330 $01.25
© 1991 Operations Research Society of America

Rule-Based Generation of Model Structures in Multifaceted Modeling and System Design

JERZY W. ROZENBLIT *Department of Electrical and Computer Engineering, University of Arizona, Tucson, AZ 85721, INTERNET: jr@helios.ece.arizona.edu*

YUEH M. HUANG *Department of Electrical and Computer Engineering, University of Arizona, Tucson, AZ 85721*

(Received: March 1989; final revision received: November 1990; accepted: March 1991)

A framework for aiding the construction of simulation models in system design problems is presented. The framework employs concepts of artificial intelligence and simulation modeling. A knowledge representation scheme called *system entity structure* expresses information about the structure of the system to be designed and its corresponding models. More specifically, the entity structure represents objects and their attributes, decompositions, and taxonomies. A knowledge base of production rules is defined for a given design domain and is incorporated in an expert system shell which recommends a feasible configuration of design objects from the system entity structure, based on specific design objectives, constraints, and requirements. This configuration is a basis for constructing a model of the system. The methodology for constructing the system entity structure and its corresponding rule base is presented. A case study based on a high level robot design problem is discussed to illustrate the conceptual framework.

Recent trends in simulation modeling research emphasize the development of integrated software support environments.[3-6,12,22,24] Such environments are envisioned as conglomerates of tools that aid in the construction of models and simulation programs. Several notable features of the existing software prototypes distinguish them from conventional simulation tools.

First, the new simulation environments are methodology-based, i.e., their design is strongly influenced by a methodology that underlies the model development process in a given environment. Secondly, state-of-the-art software technology is employed to implement theoretical concepts. Common software techniques used in designing the new simulation systems include object-oriented programming, graphics interfaces with animation, and automatic programming. Recently, Artificial Intelligence (AI) methods have been used in model construction and validation, as well as simulation management and analysis.[5,15,22,24]

This paper briefly discusses efforts in developing integrated, advanced modeling environments. It presents methods and tools to support the construction of system models. Applying the AI-based production rule framework to simulation model structuring, i.e, defining the structure of a simulation model, may reduce the complexity of the model development process in system design problems. These methods are expected to contribute to both general system design and simulation modeling research. In the system design context, a representation scheme is presented that can capture and organize sets of components (and relations among them) of the system being designed. A method to select design components is given to ensure that their arrangement (coupling) satisfies design constraints and criteria. This framework allows designers to generate structures from which alternative models of a design can be constructed and evaluated by a simulation study.

Our results extend directly to general modeling research. More specifically, they are applicable to large-scale modeling problems that involve a multitude of system components and various ways of decomposing and classifying the components. The approach provides model management facilities for dealing with the complexity of such problems. It complements some of the research efforts in the model development methodology, briefly described in the ensuing section.

1. Current Efforts in Design of Advanced Modeling Environments

Balmer[5,6] characterized modeling styles and their impact on the design of the integrated software support environment (ISSE) in the CASM (Computer Aided Simulation Modeling) context. The CASM environment is centered around a simulation system consisting of modules of declarations, functions, and procedures, and a program template implemented in Pascal. This basic organization provides a structured framework for the development of simulation models. Besides providing basic support in the

Subject classifications: Computers/Computer Science. artificial intelligence, system design Simulation· construction of simulation models.

form of file handling facilities, editors, and debuggers, the system has a prototype program generator, graphical modeling support, and an expert system interface.

The modeling process is based on the three phase simulation approach.[5] The system supports hierarchical modeling by providing methods for description, editing and retrieval of model specifications in a form reflecting their hierarchical structure. Facilities are available for comparison, analysis, and exploratory simulation of partial models in large scale simulations of multi-component systems. AI support is provided through a natural language interface and expert simulation control.

Guided by the paradigms of the Conical Methodology,[17] Nance and Balci[4] have been developing a system termed Simulation Model Development Environment (SMDE). The goal of this effort is to provide a collection of computer-based tools for automated support of model development, to improve the model quality, and to increase the effectiveness and productivity of modeling teams. The architecture of SMDE consists of four layers: hardware and operating system, kernel SMDE, minimal SMDE, and SMDEs supporting specific applications. Detailed descriptions of these components are given in [4]. Model development proceeds in a particular conceptual framework (e.g., next event, activity scanning). Balci's recent work is concerned with applying the automation-based paradigm[4] to routinely translate the model's formal specification into executable code.

Another notable development in designing automatic model synthesis systems is the work of Murray and Sheppard[15] who designed a Knowledge-Based Model Construction (KBMC) system. KBMC's definition combines simulation modeling, automatic programming, and expert systems. The system provides facilities for complete model specification through an interactive knowledge dialog (modeling knowledge acquisition). An executable simulation model is derived based on modeling knowledge, target language knowledge, and a set of construction rules incorporated in KBMC's knowledge base. The system has been implemented in OPS83 and SIMAN.

The ensuing section demonstrates an approach to building models derived from our work in system design methodology termed Knowledge-Based Simulation Design (KBSD).[21,23,25]

2. Model-Based System Design

A design methodology is an important basis for supporting automation of the design process. Although design concepts are pervasive in modern engineering, no single framework is accepted as fundamental. The methodologies offered by various design disciplines lack a uniform treatment of the design process at various levels of abstraction. Often, there is no underlying formal basis for design representation and evaluation. Consequently, efforts to develop environments for support of design activities have no theoretical backing, and the resulting systems are usually conglomerates of different, incompatible tools whose coordination creates a substantial overhead in the design process.[9,11,19,20,26,27]

A number of methodologies and design systems have been developed to aid the engineering design process in different domains.[1,2,7,10,24] Knowledge-Based frameworks consider design as an activity in which knowledge about a specific domain is used to represent design artifacts, constraints, and requirements. Design is often considered as a search process in which a satisfactory design solution is produced from a number of alternatives.[2,10,11,25] The search proceeds in a design space whose elements are design objects (components) and attributes (parameters).

Knowledge-Based Simulation Design uses modeling and simulation techniques to build and evaluate models of the system being designed. It treats the design process as a series of activities that include specification of design levels in a hierarchical manner (decomposition), classification of system components into different variants (specialization), selection of components from specializations and decompositions, development of design models, experimentation and evaluation by simulation, and choice of design solutions.

The design model construction process begins with developing a representation of design components and their variants. To appropriately represent the family of design configurations, we have proposed a representation scheme called the *system entity structure* (SES).[31] The scheme captures the following three relationships: decomposition, taxonomy, and coupling. Decomposition knowledge means that the structure has schemes for representing the manner in which an object is decomposed into components. Taxonomic knowledge represents the possible variants for an object, i.e., how it can be categorized and subclassified. Synthesis (coupling) constraints dictate the manner in which components identified in decompositions can be connected. Selection constraints limit choices of variants of objects determined by the taxonomic relations.

Beyond this, procedural knowledge is available in the form of production rules which manipulate the elements in the design domain by appropriately selecting and synthesizing the domain's components. This selection and synthesis process is called *pruning*. Pruning results in a recommendation for a *model composition tree*, i.e., the set of hierarchically arranged entities corresponding to model components. A composition tree is generated from the system entity structure by selecting a unique entity for specializations and a unique aspect for an entity with several decompositions.

The final step in the framework is the evaluation of alternative designs. This is accomplished by simulation of models derived from the composition trees. Discrete Event System Specification (DEVS)[31,34] is used as a modeling formalism for system specification in the methodology. DEVS provides a formal representation of discrete event systems. It is closed under coupling. This property facilitates the construction of hierarchical DEVS network specifications.

Performance of design models is evaluated through computer simulation in the DEVS-Scheme Environment.[32,34] DEVS-Scheme is an object-oriented simulation shell for modeling and design that facilitates construction of families of models specified in the DEVS formalism. Alternative design models are evaluated with respect to design performance questions. Results are compared and traded off in the presence of conflicting criteria. This results in a ranking of models and supports choices of alternatives best satisfying the set of design objectives.

This article focuses on one aspect of the knowledge-based simulation design (KBSD) framework, namely, the process that supports the development of design model structures.

2.1. Formal Concepts for Model-Based Design

Our methodology for supporting the design process is based on codifying appropriate decompositions, taxonomic, and coupling relationships. In other words, knowledge about the design domain is modeled by finding pertinent decompositions of the domain, the possible variants that fit within these decompositions, and the constraints that restrict the ways in which components identified in decompositions can be coupled together. This constitutes the declarative knowledge base. Beyond this, production rules can be used to manipulate the elements in the design domain by appropriately selecting and synthesizing the domain's components.

A formal object that meets the requirements stipulated above is the *system entity structure*. A system entity structure is a labeled tree. Nodes of the tree are classified as *entities*, *aspects*, *specializations*, and *multiple decompositions*. Variables can be attached to nodes. They are called *attached variables*. A high level view of system entity structure for a robot design is shown in Figure 1. Definitions and explanation of the system entity structure components follow: (In describing the terms, we refer to Figure 1 which depicts a high level view of the entity structure representing possible, high level decompositions and specializations of a robot.)

Entity: a real world object which either can be independently identified or postulated as a component of some decomposition of a real world object. In the SES of Figure 1, Robot is a root entity signifying the system to be designed. Communication Subsystem is an entity identified as a component of a decomposition of Robot.

Aspect: one possible decomposition of an entity. The

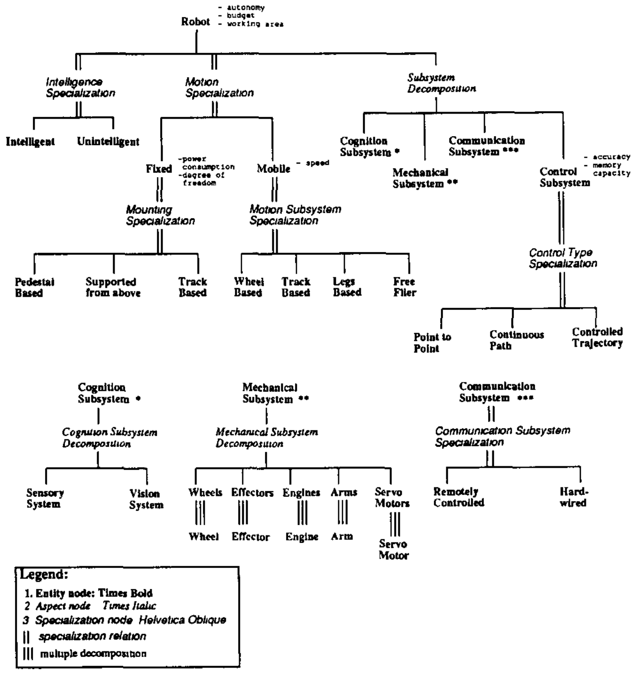


Figure 1. Robot system entity structure.

children of an aspect are again entities representing distinct components of the decomposition. A model can be constructed by connecting together some or all of these components. (The various aspects of an entity do not necessarily represent disjoint decompositions. A new aspect can be constructed by selecting entities from existing aspects as desired.) *Subsystem Decomposition* in Figure 1 is an aspect that defines the four entities Cognition, Mechanical, Control, and Communication Subsystems.

Specialization: a mode (that is, a particular way) of classifying an entity. It depicts the taxonomy of the system being represented by SES. The use of specializations is a powerful way of representing many different variations of the same object. For example, the entity Robot is classified (double vertical line in the figure) into Intelligent and Unintelligent Robot through the *Intelligence Specialization* and through the *Motion Specialization* categorizes robots into Mobile and Fixed Robots.

Multiple decomposition: a special type of decomposition (aspect) used to represent entities whose number in a system may vary (denoted by triple vertical lines as in Figure 1). For example, Mechanical Subsystem may have 0, 1, 2, or more Wheels.

Attached variable: an attribute of an object represented by the entity with which the variable is associated. Thus, Control Subsystem may be characterized by its *accuracy*, *memory capacity*, etc. (attached variables are preceded with by a hyphen (-) in the figure).

The system entity structure satisfies a set of axioms that guide its development. The axioms also govern how knowledge expressed by an SES is processed. The axioms are:

1. *Uniformity*: any two nodes with the same labels have identical attached variables and isomorphic subtrees. The *uniformity* axiom ensures compactness of representation; once a node and its substructure and attributes have been specified, it need not be done again if a new node with the same label in a different path of the tree is created. For example, in Figure 1 if Analog/Digital Converter were added as an entity of Vision System and then as an entity of Sensory System, those two entities would have identical attached variables and substructures.
2. *Strict hierarchy*: no label appears more than once down any path of the tree. The *strict hierarchy* axiom ensures that no object can be decomposed into itself. For example, decomposing Mechanical System into Robot would violate this axiom.
3. *Alternating mode*: each node has a mode which is either "entity," "aspect," or "specialization"; if the mode is entity, then the modes of its successors are aspect or specialization; if the mode is aspect or specialization, then its children are entities. The mode of the root is entity. The *alternating mode* axiom ensures consistency in successive decompositions and specialization of entities. The root of an SES tree is always an entity. Consider again Figure 1 with Robot as the root entity. Next level nodes in this tree are specializations, namely, *Intelligence* and *Motion Specialization* and a decomposition node called *Subsystem Decomposition*. The children of these nodes are entities.
4. *Inheritance*: every entity in a specialization inherits all the variables, aspects and specializations from the parent of the specialization. The *inheritance* axiom facilitates derivation of alternate arrangements of entities in an SES tree. The root entity Robot has a specialization called *Motion Specialization* with entities Fixed and Mobile. In such a specialization relation Robot is referred to as a general type relative to the entities Fixed and Mobile, which are called specialized types. Besides having their own structure and attributes, Fixed and Mobile inherit all of the variables and substructures of Robot. To make this true, we must be assign to Robot only those attributes that are common to all its variants.
5. *Valid siblings*: no two sibling nodes have the same label.
6. *Attached variables*: no two variables attached to the same item have the same name. The last two axioms prevent us from specifying duplicate names for entities in the same decomposition (specialization) and from duplicating the names of entities' attached variables.

The axioms furnish a unifying set of rules for developing and manipulating entity structures. A more detailed, formal treatment of the system entity structure concepts can be found in [23, 31].

The system entity structure specifies a family of possible arrangements of components for the system being designed (hereafter called design configurations). The entities represent system components whose models will be built. Aspects and specializations allow us to specify a family of design alternatives by selecting different components and decompositions. Thus, the system entity structure is a generative scheme from which a set of substructures underlying the construction of alternative design models can be derived. The multiplicity of taxonomic and decomposition relationships in a large design entity structure leads to a combinatorial explosion of possible model alternatives. Therefore, it is necessary to provide procedures that effectively reduce both the complexity of the search process for admissible model structures and the size of the search space itself. This paper emphasizes the importance of this problem and describes a rule-based procedure to solve it, called *pruning*.

Pruning the system entity structure results in a set of alternative composition trees.[21,23] A model composition tree is a structure from which a simulation model is constructed hierarchically by coupling model specifications associated with the nodes of the tree. To obtain a model composition tree from the system entity structure, we need to select a unique aspect for each entity and eliminate specializations. An example of a model composition tree that results from the system entity structure shown in Figure 1 is given in Figure 8 and discussed in Section 5. Rule-based pruning is discussed first.

3. Rule-Based Model Structure Synthesis

The multiplicity of aspects and specializations in large design entity structures offers a spectrum of choices for alternative model structures. Therefore, an effective procedure capable of handling the combinatorially unfolding set of design model choices is needed. Our previous work focused on a process termed *generic experimental frame driven pruning*.[21,23] This process is summarized next and its deficiencies are pointed out. These deficiencies motivated us to extend the previously developed algorithms and develop the rule-based approach presented in this paper.

The generic experimental frame driven pruning is based on the following scheme. Assume that a set of generic variables that reflect design dynamic performance attributes (e.g., throughput, queue length, component utilization) is defined. This set is called a *generic experimental frame*.[21] Also, assume that an entity structure has been transformed into a structure that contains no specializations. Such an entity structure is called a *pure* *entity structure*. For entity structures in which specializations occur, procedures for mapping such structures into a set of pure entity structures are provided in the next paragraph. In specifying pruning algorithms, the concept of a nondeterministic algorithm is employed, i.e., the algorithms are allowed to contain operations whose outcome is not defined uniquely but limited to a specified set of possibilities. In the case of pruning pure entity structures, the purpose of the nondeterministic version of the algorithm is to provide a definition of the set of all correct model structures that the deterministic version should produce.

The case of pruning entity structures with specializations is more complex. The choice of a nondeterministic algorithm is justified in that there is no deterministic procedure that generates the solution in polynomial time.[21,23]

The goal of the generic frame based pruning process is defined as follows: extract the substructure(s) of the system entity structure whose entities have the attached variables present in the generic experimental frame. Thus, obtain model substructure(s) whose variables correspond to those expressed by design dynamic performance attributes. These substructures form the basis for building design models which can be evaluated by simulation with respect to performance measures that the attributes characterize.

Consider first the pruning of pure entity structures. A nondeterministic version of the algorithm employs a function *choose* that selects an aspect for an entity. The algorithm returns a nondeterministically selected composition tree that accommodates the generic experimental frame. The deterministic version of the algorithm is based on depth first tree traversal. In this procedure, every entity in each aspect is searched for occurrences of variables present in the generic frame. The entities are attached to the model composition tree being built as the search is progressing. The algorithm calls itself recursively for each entity being searched.

The limitation of the procedure is that it operates on pure entity structures. Therefore, the procedure must be augmented with modules that prune entity structures with specializations. The extended procedure includes an algorithm which generates a set of all possible pure structures given any entity structure. This process is called *Split*. The set of pure entity structures is generated by substituting the specialized entities (entities that occur in a specialization relation of their parent entity) in place of their parent entity. (Thus the number of pure structures is equal to the number of specialized entities.) The pure structures are then pruned as described in the preceding paragraph.

A deterministic algorithm that performs the split generates all the combinations of pure entity structures resulting from substituting specialized entities into the parent entities. Basically, it unfolds the entity structure in a combinatorial manner. Thus, the problem grows exponentially with respect to the size of the system entity structure. Beyond the computational complexity, there is no way to capture and propagate constraints imposed on the structure of the system being designed (recall that the generic frame represents only behavioral aspects of a design problem).[21]

To alleviate these problems, the production rule formalism is used as a framework for design entity structure pruning. The rule-based driven pruning is characterized as follows: A knowledge base for a given design application domain is defined. The knowledge base contains rules for selection and synthesis of entities defined in specializations and aspects of the design entity structure. The rules are derived from both the performance and the structural constraints of the design application. Then, the modeler invokes an inference engine which, through a series of queries, produces a recommendation for the structure of the design model. An overview of the model generation process is shown in Figure 2.

3.1. Rule Base Construction

Knowledge base construction begins with knowledge acquisition. A software tool aiding knowledge acquisition for the system entity structure specification was first developed by Zeigler, Belogus and Bolshoi[30] and subsequently incorporated in the DEVS-Scheme system—a hierarchical, object-oriented, discrete event simulation environment—under the name ESP-Scheme.[32]

The program helps the modeler to conceptualize and record the decompositions underlying a model, or family of models, before, during, and after development. To the extent that ESP-Scheme is used before beginning the model development, it assists in the top-down model design. However, when additions and changes are made as the development proceeds, ESP serves as a recorder of progress. At the end of the development phase, the record constitutes *de facto* documentation of the system structure arrived at. Pruned entity structures serve as a basis for retrieval from a model base of model components specified in DEVS-Scheme.

Recall that an entity structure represents possible configurations of components of the system being designed. From the problem solving standpoint, the nodes of the entity structure represent states which make up the search space. The process of design model structure generation can be interpreted as a search directed by constraints through the search space consisting of the entities, their aspects and specializations. An inferencing scheme drives this search. The resultant model structure is given by the composition tree generated from the states selected through the inferencing process.

The form of the constraints depends on the problem domain. AI techniques use production rules as a constraint representation framework. This rule-based approach is the

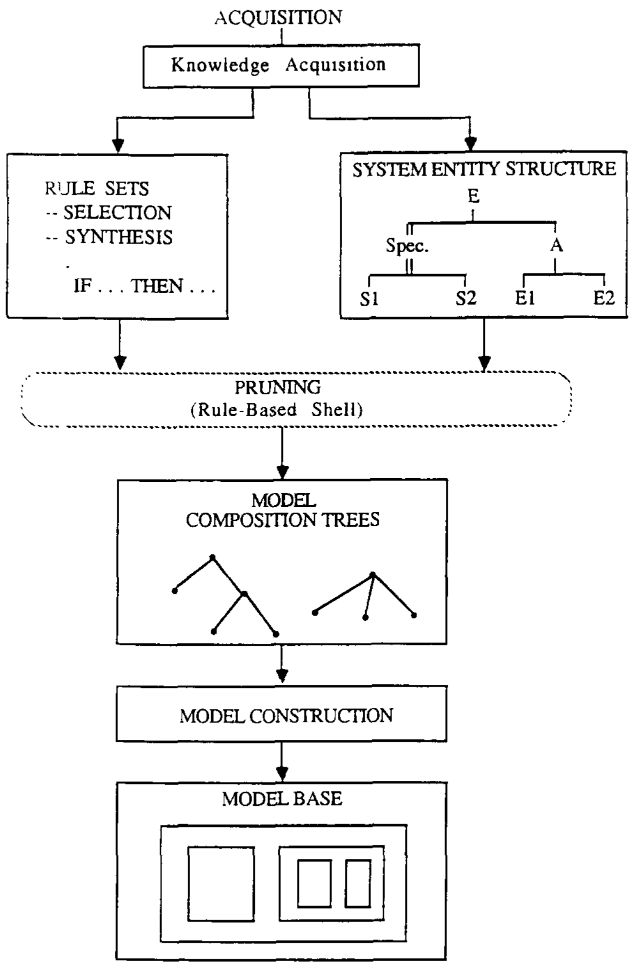


Figure 2. Model generation process.

basis of many successful expert systems, e.g., MYCIN, DENDRAL, PROSPECTOR.[28] Some of its advantages are: (a) the conversion of knowledge into a rigid formalism results in easy checking of uniformity, (b) each production rule represents a small, independent piece of knowledge, thus facilitating modularity (c) rigid syntax affords the convenience of checking consistency, and (d) it is relatively easy to furnish simple explanation facilities[29] that annotate why or how a rule was invoked in the inferencing process. However, design of complex explanation facilities requires substantial effort and is the subject of separate research.[14]

In our approach, the rules are used to express modeling objectives, design constraints, users' requirements and performance expectations. The rule sets define a space of constraints within which design solutions can be found. The aim of the inferencing process is to generate solutions from this space.

The following steps provide the rules that guide pruning of the system entity structure: (1) for each specialization, specify a set of rules for selecting an entity from this specialization, (2) for each entity with an aspect, specify synthesis rules that ensure that the entities selected from specializations below this aspect are configurable (i.e., the components they represent can be validly coupled) and that such a configuration satisfies the design problem at hand. Each rule can be assigned a certainty factor indicating the rule's degree of applicability.

The modules of selection and synthesis rules are defined as follows:

Selection rule set: this rule set is associated with a specialization of an entity (let us denote it *ES*). The rules determine which specialized entity can be selected from this specialization to serve as an instance of the class represented by the entity *ES*.

Synthesis rule set: this rule set is associated with an entity that has an aspect (let us denote it *EA*). The rules check whether the aspect's entities can be combined according to the coupling constraints to form the entity *EA*. In case an entity has multiple aspects, the synthesis rules define alternative configurations of components for this entity. The rules may specify a separate configuration for each aspect or they may combine several aspects to form one configuration. (Note that the aspects of an entity do not necessarily represent disjoint decompositions. A new aspect can be constructed by selecting from these aspects as desired.)

Recall Figure 1 with the Robot system entity structure. Below, we give an illustrative selection and a synthesis rule for this entity structure. A complete knowledge base for the Robot design example is given in Section 4, Table 1.

Example of a selection rule for the entity Control Subsystem:

Rsel

If position accuracy is high ($<$ 0.01 inch) or
  memory capacity in robot is high and
  budget is high and
  moving of end effector is interruptible
Then type of control is controlled-trajectory (1.0)

Example of a synthesis rule for the entity Robot:

Rsyn

If recommended-robot is determined-by-type-of-motion and
  type of motion is mobile-free-flier and
  type of control is controlled-trajectory
Then robot system is configurable (1.0) and
  motion subsystem is free flier (1.0) and
  remote communication is required (0.9) and
  engine motion mechanism is strongly recommended (1.0)

The selection and synthesis rule sets define a knowledge base utilized by the inference engine for pruning a design entity structure for a particular application domain.

Writing the rules is a nontrivial task. It requires that the knowledge engineer closely cooperate with the design experts and follow a knowledge acquisition process driven by the system entity structure representation. The entity structure serves as a framework to explore design knowledge. The relevant knowledge is "coded" in a cluster of rules called a rule module. The chaining path of a rule module is connected only to its parent and children modules. This eliminates "bugs" in rules such as circulated chaining or unchained rules.[18]

Following the guidelines below, a complete rule base for a design application domain is built.

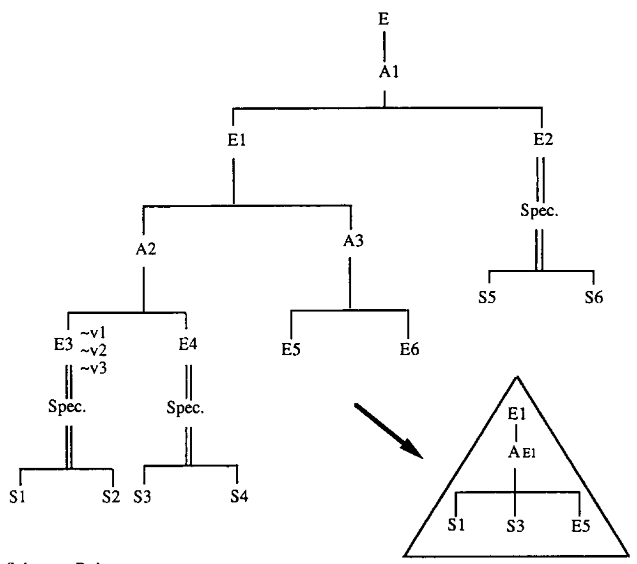
Phase I: Selection Rules

1. Each rule corresponds to an entity node in a specialization. The conclusion of a rule is an action that substitutes this entity node for its parent entity. This is called instantiation of a general entity type. The premise of the rule is made up of attached variables and their legal values which are derived from the designer's expertise. The legal values can be selected through a consultation process or may be assigned by default. Consider the rule Rsel given above. Its premise part is defined by the following compound condition: (position accuracy is *high* (< 0.01 inch) or memory capacity in robot is *high* and budget is *high* and end effector is *interruptible*). The conclusion part states that if the above conditions are satisfied, then the type of control to be used in the system being designed is *controlled-trajectory*.
2. Repeat Step 1 with another entity until a rule set has been created for all the entities in this specialization.
3. In the sequence of a tree postorder, find another entity and repeat the above steps until all the entities in the specializations of the system entity structure have been assigned selection rules.

Phase II: Synthesis Rules (attached to entities with aspects)

1. The premise of a synthesis has one or two parts. The first part is formed from the conclusions of the next level selection rules. (The action of the rule is to combine the specialized entities chosen by the selection rule module.) For example, in the rule Rsyn above, the conclusions of selection rules are mobile-free-flier, controlled-trajectory, and determined-by-the-type-of-motion. The second part is optional, but may include conditions that employ variables and their value assignments that describe the entity to which the rule is attached. Such conditions may impose a manner in which components can be connected or arranged in the final configuration of the system being designed.
2. The conclusion of a rule is a set of assertions specifying components that make up the entity to which the synthesis rules are attached. Again consider the rule Rsyn. The conclusion part states that a robot can be synthesized given that the conditions of the premise part are satisfied, and that the following subsystem types are required: a free flier motion subsystem, a remote communication subsystem, and an engine motion mechanism. The assertions of the conclusion part may also include a specific way of coupling the constituent components, e.g., a cascade or a parallel composition. For entities with multiple aspects, the synthesis rules define alternative configurations of components for this entity. (Recall that the aspects of an entity do not necessarily represent disjoint decompositions and a new aspect can be constructed by selecting entities from existing aspects as desired.) These assertions define an aspect called a *generated aspect* (please see Figure 3). A generated aspect specifies a final, unique arrangement of subentities for the entity being synthesized.
3. In postorder, repeat Steps 1 and 2.

Figures 3 and 4 illustrate examples of the syntax of the rules and the order of their specification.



Selection Rule:
If v1 = Lv-11 and v2 = Lv-22 or
v3 = Lv-32
Then E3 = S1
where
Lv stand for legal values of attributes

Synthesis Rule.
If E3 = S1 and E4 = S3 and
include (E5) = true
Then AE1 (generated aspect) = (S1, S3, E5)
where
include (E5) is a set of conditions for composing a new aspect
AE1 (generated aspect) that includes S1, S3, and E5

Figure 3. Examples of creating selection and synthesis rules. E, entity; A, aspect; Spec., specialization; S, specialized entity; V, attribute (attached variable).

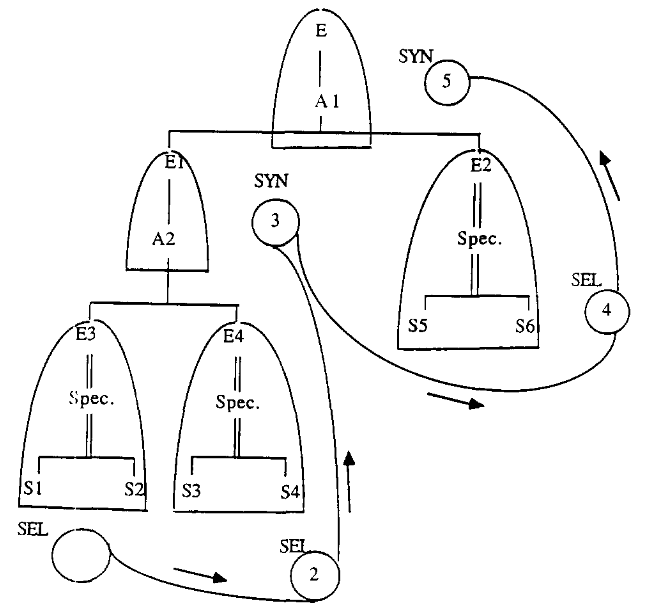


Figure 4. Postorder of writing selection and synthesis rules. E, entity; A, aspect; Spec., specialization; S, specialized entity; SEL, selection module; SYN, synthesis module.

Based on the above development method, the rule base consists of two types of rule sets organized in a hierarchical manner and mapped onto the entity structure. This characteristic may facilitate the construction of autonomous knowledge base units applied in a distributed problem-solving network. In other words, this rule base organization may not only benefit the future knowledge base refinements and verifications but also the construction of large scale knowledge-based systems (LSKBs).[8]

3.2. Organization of the System Shell

After the knowledge base has been constructed, it is subjected to an inferencing process by a shell called MODSYN (MODel SYNthesis).[13] MODSYN consists of four components as depicted in Figure 5: the inference engine, the user interface, the explanation facility, and the working memory. The shell was implemented in Turbo Prolog and runs on IBM PC compatible machines. Brief descriptions of each component follow.

The Inference Engine

The inferencing mechanism has two parts: the reasoning and the control module. Since the shell was implemented in Prolog, the reasoning technique inherits such features as pattern matching, backtracking, and nondeterminism. When a goal is given (objective of the model synthesis), the inference engine tries to match this goal with the conclusion of the first synthesis rule. If the goal does not match, another synthesis rule is selected. (If no rule matches the goal, pruning fails.) If matched, the inference

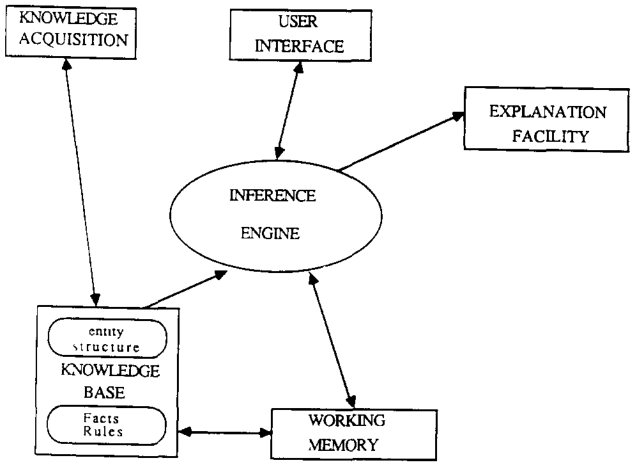


Figure 5. MODSYN architecture.

engine explores all possible chaining paths to satisfy the premise part of the rule. All explored paths form a tree structure with the goal as its root. The nodes of the tree represent rules unless its leaves are facts from the user's specifications or the working memory. The facts (data) from the user determine whether the reasoning tree can be constructed or not. In other words, the inference engine takes the goal and data from the user, and uses the hypotheses generated by the knowledge base to prune the search space tree. If all the facts provided are consistent with the knowledge base, a pruned reasoning tree is generated.

The control technique is a depth-first search which traverses the entity structure as depicted in Figure 6. The search begins from the main synthesis module down to the leaf selection modules. The module instantiation processes proceed in the opposite direction. If a module is instantiated, the engine climbs up to its parent and then down to the sibling modules. When the sibling modules are instantiated, parents are as well. The search process continues until the main synthesis module is instantiated expanding only one aspect of an entity at a time and eliminating the aspects that cannot be synthesized. This enables the construction of a composition tree for a design model. Unlike the generic frame based pruning, rule-based pruning operates on system entity structures in their most compact form, i.e., the structures are not split or unfolded with respect to specialization and aspect relations. This significantly speeds up the entity structure traversal.

User Interface and Explanation Facilities

MODSYN uses two basic windows in the user interface: the entity structure display and the consultation display. The former displays an entity structure with a dynamic menu that shows associated rules or attributes at each node. The latter is menu-driven. The legal values of objects' attributes are retrieved from the rule base auto-

Table I
Selection and Synthesis Rules for Robot Design

	if	then
Intelligence Specialization Selection Rules for the Entity Robot		
R1	desired autonomy is high or medium	recommended-robot is intelligent (0.9)
R2	desired autonomy is low	recommended-robot is unintelligent (0.9)
R3	desired autonomy is "don't care"	recommended-robot is determined-by-type-of-motion (1.0)
Motion Specialization Selection Rules for the Entity Robot		
R4	budget is relatively low and working area is not greater than 25 square feet or requirement of arm carrying capacity is heavy (> 1000 lbs)	recommended-motion-type is fixed (1.0)
R5	budget is relatively high and working area is usually greater than 25 square feet or requirement of arm carrying capacity is not heavy (<= 1000 lbs)	recommended-motion-type is mobile (1.0)
Mounting Specialization Selection Rules for the Entity Fixed Robot		
R6	recommended-motion-type is fixed and power consumption is low and degree of freedom is low and work area has a solid ground	type of mounting is fixed-pedestal (0.9)
R7	recommended-motion-type is fixed and power consumption is medium and degree of freedom is low and work area has a soft ground	type of mounting is fixed-supported-from-above (0.7)
R8	recommended-motion-type is fixed and power consumption is relatively high and degree of freedom is medium	type of mounting is fixed-track-based (0.8)
Motion Subsystem Specialization Selection Rules for the Entity Mobile Robot		
R9	recommended-motion-type is mobile and required moving speed is medium and ground adaptivity is medium	type of motion is mobile-wheel-based (0.9)
R10	recommended-motion-type is mobile and required moving speed is high and ground adaptivity is low and power consumption is low	type of motion is mobile-track-based (0.9)
R11	recommended-motion-type is mobile and required moving speed is low or ground adaptivity is high	type of motion is mobile-leg-based (0.7)

Table I
Continued

R12 if recommended-motion-type is mobile and required moving speed is high and ground adaptivity is high and power consumption is high	then type of motion is mobile-free-flier
Control Subsystem Specialization Selection Rules for the Entity Control Subsystem	
R13 if position accuracy is low (> 0.01 inch) or budget is relatively low and memory capacity in robot is low and moving of end effector is not interruptible	then type of control is point-to-point
R14 if position accuracy is medium (between 0.01 to to 0.005 inch) or budget is medium and moving of end effector is interruptible	then type control is continuous-path
R15 if position accuracy is high (< 0.005 inch) or memory capacity in robot is high and budget is high and moving of end effector is interruptible	then type of control is controlled-trajectory
Synthesis Rules for the Main Synthesis Module	
R16 if recommended-robot is intelligent and type of control is controlled-trajectory	then robot system is configurable (0.9) and robot is an intelligent robot (1.0) and cognition system is strongly recommended (1.0) and communication subsystem is either remote or hard-wired (1.0)
R17 if recommended-robot is determined-by-type-of-motion and type of motion is mobile-free-flier and type of control is controlled-trajectory	then robot system is configurable (1.0) and motion subsystem is free flier (1.0) and remote communication is required (0.9) and engine motion mechanism is strongly recommended (1.0)

matically. Besides the values, other terms such as UNKNOWN, WHAT, and WHY are included in the menu. The last two terms provide simple explanation facilities as to what facts have been determined and the triggering status of the system.

Working Memory

The shell employs a default relational database of the Prolog programming environment as the working memory. The working memory is used to store generated facts during the inferencing process.

4. Example: Generating a Robot Model Structure

In related work,[33] we have been developing concepts for establishing a simulation environment capable of supporting the design of robot organizations for managing chemical, or similar laboratories aboard the Space Station. Model specifications in such a simulation environment are facilitated by applying the system entity structure concepts. In the following, we illustrate the model structure generation concepts by first developing a high level robot system entity structure and then applying the rule-based pruning procedure.

The System Entity Structure

Recall Figure 1 discussed in Section 2. *Intelligence Specialization* classifies the entity Robot as Intelligent or Unintelligent. *Motion Specialization* generates Fixed Robot and Mobile Robot. *Mounting Specialization* further classifies Fixed Robot as Pedestal Based Robot, Robot

Supported from Above, and Track Based Robot while *Motion Type Specialization* defines Mobile Robot as Wheel Based, Track, Based, Legs Based, or Free Flier Robot.

The entity Robot has an aspect called *Subsystem Decomposition* whose major components are Cognition, Mechanical, Control, and Communication Subsystems each of which is further classified and decomposed. Entities in the Robot design entity structure have attached variables which represent attributes of system components represented by the entities. In Figure 1, the variables are prefixed a hyphen (-).

To select a specific robot design model structure, we construct a knowledge base that consists of selection and synthesis rules that reflect robot design expertise.

The Knowledge Base

The selection and synthesis rule modules given in Table I have been defined for the Robot entity structure (numbers in parenthesis are certainty factors with (0.0–1.0) range).

The group of rules for Intelligence Specialization define criteria for selecting an intelligent or unintelligent robot design (Figure 1). In this rather simple example, the criterion used is the degree of desired autonomy the robot should have. The next three selection modules correspond to Motion Specialization, Mounting Specialization of Fixed Robots, and Motion Subsystem Specialization. Notice the two tiered hierarchy of selection rule blocks. This hierarchy corresponds to the hierarchy of specializations in the SES of Figure 1. General criteria for selecting fixed versus mobile robots are the scope of the work area, arm carrying capacities, and available budget. More specific criteria such as power consumption, ground adaptivity, moving speed, and degree of freedom enable further selection of robot components. The rules for choosing the type of control use attributes such as memory capacity, interruptibility of the end effector, and accuracy. The synthesis rules define the overall configuration for the entity Robot based on the choices resulting from lower level selection modules.

We now illustrate pruning of the system entity structure of Figure 1 with respect to the rules specified above.

4.1. Consultation Process and Pruned Model

The following is a consultation session that generates a recommendation for a robot model structure. The questions are activated by MODSYN. The user's responses are preceded by the symbol >.

1. Is the autonomy of robot high, low, or don't care?
   > don't care
2. What is the budget?
   > high
3. What is the scope of work area?
   > greater than 30 square feet
4. What is the moving speed? (relative value)
   > high
5. Is the ground adaptivity high, medium, or low?
   > high
6. What is the value of power consumption?
   > high
7. What is the required position accuracy?
   > smaller than 0.01 inch
8. Is the end effector interruptible?
   > yes

Conclusion: Robot system is configurable (0.9) and
type of control is controlled-trajectory (1.0) and
motion subsystem and is free flier (0.9) and
remote communication is required (0.81) and
engine motion mechanism is strongly recommended (0.9)

MODSYN uses backward chaining as its reasoning mechanism. Table II describes the chaining process for the consultation session listed above. The resulting pruned entity structure is shown in Figure 7. This structure defines the composition tree depicted in Figure 8.

Although the sizes of the system entity structure and the rule base in this example are relatively small, manual tracing of the pruning process proves rather cumbersome. In larger design problems, such an inspection would be ineffective. Hence, it is clear that MODSYN provides a means for assisting the user in design decisions concerning choices of design model components and their arrangement.

As we have indicated in Section 2, the next phase in this design framework is to develop a simulation model for the design structure recommended by MODSYN. This is beyond the scope of this paper. However, ample examples of discrete event model specification in Model Based System Design are given in the literature.[24,32,34] A schematic of a model that would be developed and simulated to obtain performance design measures is shown in Figure 9.

5. How the Presented Framework Supports System Design and Modeling

Basic elements in the dynamics of the design process[9,11,16] are summarized below and related to our methodology.

1. Designs are created by individuals who use basic problem-solving techniques, namely, defining the problem, proposing a solution and testing how well the solution works with respect to the problem definition. Modeling, as a creative act, very much follows the above steps. To build a model, the designer defines the requirements and objectives of the project in the light

Table II
Example of Inferencing Process

Rule Triggered	Rule Fired	Facts Given	Explanations
R16		Desired-autonomy = ''*don't care*''	The first synthesis rule R16 is triggered. The system attempts to verify if the type of robot is Intelligent Rule 16 cannot fire since the robot type is determined by the type of motion as given by rule R3.
R17			Rule 17 is triggered and the system attempts to verify if its premises are satisfied. Since the answer to the question ''Is the autonomy of robot high, low, or don't care?'' is ''*don't care*,'' the robot type is determined by the type of
	R3		motion as given by rule R3.
R12			The first condition of rule 17 holds true. The system chains backward to rule 12.
R5			It checks if the type of motion is *mobile-free-flier*. To confirm this, MODSYN checks rule 5.
		Budget = *high* scope of work area = *greater than 30 sq. ft.*	Rule 5 determines if the motion type is *mobile*. To verify this, the system asks for values of *budget* and *the scope of the work area*.
	R5		The answers entered for questions 2 and 3 cause rule 5 to fire. The control returns to rule 12.
		Moving speed = *high* adaptivity = *high* power consumption = *high*	The values for *moving speed*, *ground adaptivity*, and *power consumption* are obtained through the next three queries (4, 5, and 6).
	R12		Rule 12 fires and the control returns to rule 17. The system
R15			must now verify if the type of control is *controlled trajectory*.
		Position accuracy = *smaller than 0.005 in.* interruptible end effector = *yes*	This is done by checking rule 15 and asking for values of *position accuracy* and *the interruptibility of the end effector*. The answers to questions 7 and 8 cause rules 15 and 17 to fire. The system reaches the conclusion given by rule 17.
	R15		
	R17		

of his perception of reality. Models in our approach are interpreted as ''design blueprints.'' Simulation is a means of executing models and testing how well they meet the project's requirements.

2. The problems being addressed are often large-scale. Thus, methods for decomposing the problems into subproblems should be easily comprehensible by the designer. Solutions of the subproblems could then be generated and integrated using proper aggregation mechanisms. By providing mechanisms for model decomposition, hierarchical specification and aggregation of model components, our approach responds to the needs stipulated above. The system entity structure facilitates generation of families of models (of designs) in various decompositions and specializations.
3. The attributes of design should be described in comparative measures. Designs should be ranked by using trade-off techniques. Pruning allows designers to generate several alternative design structures. Corresponding models can be evaluated by simulation. They can be ranked based on the results obtained from simulation studies.
4. The tools, techniques and methods are currently mostly manual methodologies and automated tools for system design are only now evolving.[7] The underlying purpose of multifaceted modeling is to provide structures implementable in computerized support environments. This is where we envision the possibility for a response to the growing needs for computer-aided design tools. Current tools lack an underlying theoretical framework

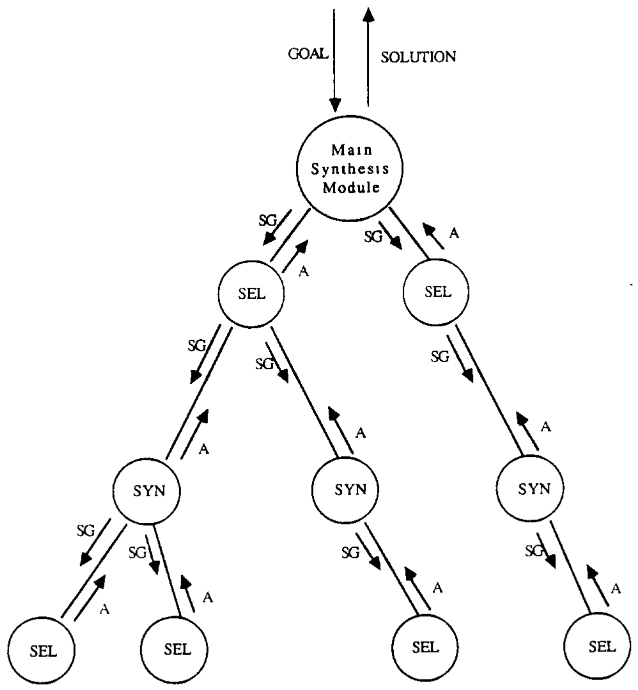


Figure 6. Depth-first search in knowledge base. SEL, selection module; SYN, synthesis module; SG, subgoal; A, answer (fired rules).

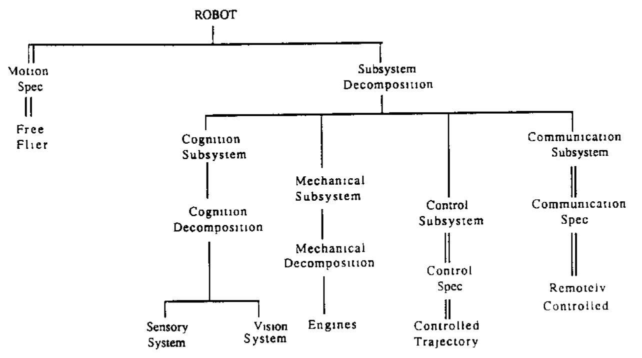


Figure 7. Pruned entity structure of robot design.

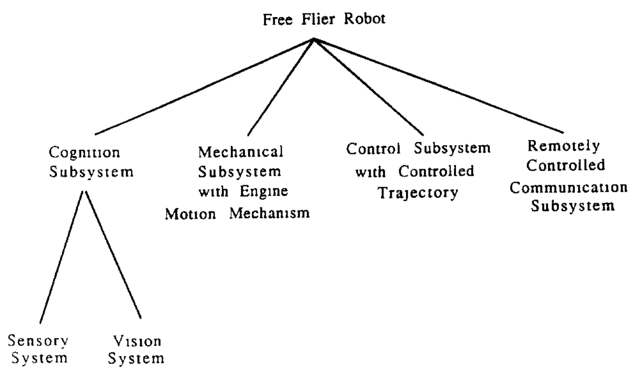

Free Flier Robot
Cognition Subsystem
Mechanical Subsystem with Engine Motion Mechanism
Control Subsystem with Controlled Trajectory
Remotely Controlled Communication Subsystem
Sensory System
Vision System

Figure 8. Robot composition tree.

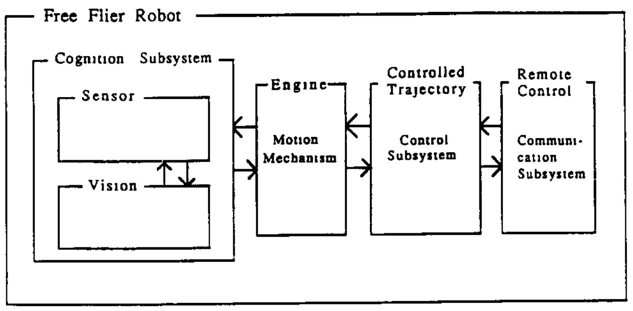


Figure 9. Structure of robot model.

that permits a uniform treatment of system design by providing concepts such as structure and behavior, decomposition and hierarchy of specification. The multifaceted framework offers well structured representation schemes and formalized operations that can exploit such schemes. This significantly reduces the effort in designing expert computer-based environments.[24]

The findings presented in this paper contribute to both simulation modeling and system design:

1. New procedures for the objectives-driven model development process are formulated based on the system entity structure and production rule formalism.
2. The formal concepts of the methodology integrate design steps and facilitate a uniform treatment of design at different levels of abstraction. The representation schemes are well structured and have well formalized operations that can exploit such structures. This can support design of CAD environments.
3. The methodology is tailored to a very wide area of problems.
4. By being well structured, the approach can be easily explained and followed. Thus, communication, documentation, explanation, and education in design can be improved.

6. Summary and Future Research

This paper presented our work in supporting the model development process. A rule-based process for selecting and synthesizing design model structures has been proposed. This process is driven by the design project's requirements and constraints. Two types of rules, i.e., selection rule and synthesis rule are sufficient to generate an acceptable structure. Thus, the modeler is assisted in choosing and properly configuring design models.

Our current efforts are focused on reducing the complexity of the knowledge base caused by the number of rules that have to be specified for a given entity structure. This complexity can be diminished by employing the entity structure driven rule base development strategy and improving the efficiency of the pruning procedure. We are investigating the feasibility of both forward and backward pruning with different search direction control strategies.

We intend to extend the pruning concepts to set up a framework for an efficient generation of design model structures on a multi-processor hardware platform. More specifically, the objectives of extending the pruning concepts are to: (1) develop methods for assigning knowledge base modules, characterized by the design entity structure, to processors, (2) develop a pruning inferencing scheme using a distributed, multiprocessor architecture, with the aim of reducing the time it takes to do a design, and (3) devise methods for concurrently generating a family of pruned entity structures.

ACKNOWLEDGMENT

This research was supported by NASA-Ames Cooperative Agreement No. 2-525. The authors wish to thank Prof. Richard Nance and two anonymous reviewers for their helpful comments which significantly improved the presentation of this paper.

REFERENCES

1. M. Agogino and A.S. Almgren, 1987. Techniques for Integrating Qualitative Reasoning and Symbolic Computation in Engineering Optimization, *Engineering Optimization 12:2*, 117-135.
2. M. Agogino et al., 1989. AI/OR Computational Model for Integrating Qualitative and Quantitative Design Methods, *Proceedings of NSF Engineering Design Research Conference*, Amherst, MA, June, pp. 97-112.
3. O. Balci, 1986. Requirements for Model Development Environments, *Computers and Operations Research 13:1*, 53-67.
4. O. Balci and R.E. Nance, 1987. Simulation Support: Prototyping The Automation-Based Paradigm, *Proceedings of the 1987 Winter Simulation Conference*, Atlanta, GA, December, pp. 495-502.
5. D.W. Balmer, 1987. Modelling Styles and Their Support in the CASM Environment, *Proceedings of the 1987 Winter Simulation Conference*, Atlanta, GA, December, pp. 478-485.
6. D.W. Balmer and R.J. Paul, 1986. CASM—The Right Environment for Simulation, *Journal of the Operational Research Society 37*: 443-452.
7. J. Bigelow, 1988. Hypertext and CASE, *IEEE Software*, March, pp. 23-27.
8. M.L. Brodie, 1986. Large-Scale Knowledge-Based Systems: Concluding Remarks and Technological Challenges, Chapter 40 in M.L. Brodie and J. Mylopoulos (eds.), *On Knowledge Base Management Systems*, Springer-Verlag, New York, NY, pp. 579-586.
9. A.N. Christakis, D.B. Keever and J.N. Warfield, 1987. Development of Generalized Design Theory and Methodology, *Proceedings of the NSF Workshop: The Study of the Design Process*, Oakland, CA, February.
10. J.R. Dixon et al., 1989. Computer-Based Models of Design Processes: The Evaluation of Designs for Redesign, *Proceedings of NSF Engineering Design Research Conference*, Amherst, MA, June, pp. 491-506.
11. J.S. Gero et al., 1989. An Approach to Knowledge-Based Creative Design, *Proceedings of NSF Engineering Design Research Conference*, Amherst, MA, June, pp. 333-346.
12. J.O. Henriksen, 1983. The Integrated Simulation Environment (Simulation Software of the 1990s), *Operations Research 31*: 1053-1073.
13. Y.M. Huang, 1987. Building An Expert System Shell for Design Model Synthesis in Logic Programming, Master Thesis, University of Arizona, Tucson, AZ.
14. J.D. Moore and W.R. Swartout, 1989. A Reactive Approach to Explanation, *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, Detroit, MI, August, pp. 1504-1510.
15. K.J. Murray and S.V. Sheppard, 1987. Automatic Model Synthesis Using Automatic Programming and Expert Systems Techniques Toward Simulation Modeling, *Proceedings of The 1987 Winter Simulation Conference*, Atlanta, GA, December, pp. 534-543.
16. G. Nadler, 1981. *Planning and Design Approaches*, John Wiley & Sons, New York, NY.
17. R.E. Nance, 1987. The Conical Methodology: A Framework for Simulation Model Development, *Proceedings of the 1987 Eastern Simulation Conference*, Orlando, FL, April, pp. 38-43.
18. T.A. Nguyen and W.A. Perkins, 1987. Knowledge Base Verification, *AI Magazine 8:2*, 42-50.
19. S. Oshuga, 1984. Conceptual Design of CAD Systems Involving Knowledge Bases, in *Knowledge Engineering in Computer-Aided Design*, North-Holland, New York, NY.
20. D.R. Rehak, H.C. Howard and D. Sriram, 1984. Architecture of an Integrated Knowledge Based Environment for Structure Engineering Applications, in *Knowledge Engineering in Computer-Aided Design*, North-Holland, New York, NY.
21. J.W. Rozenblit, 1986. A Conceptual Basis for Integrated, Model-Based System Design, Technical Report, Department of Electrical and Computer Engineering, University of Arizona, Tucson, AZ, January, 1986.
22. J.W. Rozenblit, T.G. Kim and B.P. Zeigler, 1988. Towards an Implementation of a Knowledge-Based System Design and Simulation Environment, in *Proceedings of the 1988 Winter Simulation Conference*, San Diego, CA, December, pp. 226-230.
23. J.W. Rozenblit and B.P. Zeigler, 1988. Design and Modelling Concepts, Chapter in R. Dorf (ed.), *International Encyclopedia of Robotics, Applications and Automation*, John Wiley & Sons, New York, NY, pp. 308-322.
24. J.W. Rozenblit, J.F. Hu, T.G. Kim and B.P. Zeigler, 1990. Knowledge-Based Design and Simulation Environment (KBDSE): Foundational Concepts and Implementation, *Journal of the Operational Research Society 41:2*, 101-114.
25. J.W. Rozenblit and B.P. Zeigler, 1990. Knowledge-Based Simulation Design Methodology: A Flexible Test Architecture Application, *Transactions of the Society for Computer Simulation 7:3*, 195-228.
26. A.P. Sage, 1977. *Methodology for Large Scale Systems*, McGraw Hill, New York, NY.
27. D.P. Siewiorek, D. Giuse and W.P. Birmingham, 1983. Proposal for Research on DEMETER: A Design Methodology and Environment, Technical Report, Carnegie-Mellon University, Pittsburgh, PA, January.
28. D.A. Waterman, 1986. *A Guide to Expert Systems*, Addison-Wesley, Reading, MA.
29. P.H. Winston, 1984. *Artificial Intelligence*, Ed. 2, Addison-Wesley, Reading, MA.

30. B.P. Zeigler, D. Belogus and A. Boshoi, 1980. ESP—An Interactive Tool for System Structuring, *Proceedings of the 1980 European Meeting on Cybernetics and Research*, Hemisphere Press.
31. B.P. Zeigler, 1984. *Multifacetted Modelling and Discrete Event Simulation*, Academic Press, London.
32. B.P. Zeigler, 1987. Hierarchical, Modular Discrete Event Modelling on an Object Oriented Environment, *Simulation 49:5*, 219-230.
33. B.P. Zeigler, F.E. Cellier and J.W. Rozenblit, 1989. Design of a Simulation Environment for Laboratory Management by Robot Organizations, *Journal of Intelligent and Robotic Systems 1:1*, 299-309.
34. B.P. Zeigler, 1990. *Object-Oriented Simulation with Hierarchical, Modular Models: Intelligent Agents and Endomorphic Systems*, Academic Press, Orlando, FL.

SUPPLEMENTAL BIBLIOGRAPHY

D. Belogus, 1985. Multifaceted Modelling and Simulation: A Software Engineering Implementation, Doctoral Dissertation, Weizmann Institute of Science, Rehovot, Israel.

N.J. Nilsson, 1980. *Principles of Artificial Intelligence*, Tioga, Palo Alto, CA.

J.W. Rozenblit and Y.M. Huang, 1987. Constraint-Driven Generation of Model Structures, *Proceedings of the 1987 Winter Simulation Conference*, Atlanta, GA, December, pp. 604-611.

J.W. Rozenblit and B.P. Zeigler, 1986. Entity Based Structures for Experimental Frame and Model Construction, Chapter II.2 in M.S. Elzas et al. (eds.), *Modelling and Simulation in the Artificial Intelligence Era*, North Holland, Amsterdam, pp. 79-100.