Mathematics and Computers in Simulation 34 (1992) 261–282
North-Holland

MATCOM 908

Integrated knowledge representation and management in simulation-based design generation

Jerzy W. Rozenblit *
Department of Electrical and Computer Engineering, The University of Arizona, Tucson, AZ 85721, United States

Jhyfang Hu
Department of Electrical Engineering, Tulane University, New Orleans, LA 70118, United States

Abstract

Rozenblit, J.W. and J. Hu, Integrated knowledge representation and management in simulation-based design generation, Mathematics and Computers in Simulation 34 (1992) 261–282.

The rising complexity of systems has increased the difficulty in managing knowledge in computer-aided design tools. Conventional representation schemes such as production rules, frames, AND/OR trees or semantic networks do not provide sufficient power for managing complex design knowledge. An integrated knowledge representation and management scheme, termed Frames and Rules Associated System Entity Structure (FRASES), is presented for model-based system design applications. A design methodology supported by FRASES is discussed. The methodology combines simulation and artificial intelligence to aid in the design model development and performance evaluation processes. The proposed representation scheme is fundamental to the methodology in that (a) it captures the structure of the system being designed, (b) by organizing complex design knowledge into a hierarchical and entity-based structure, it increases the efficiency of design inference, and (c) it reduces the cost and increases the reliability of a knowledge base. A comprehensive design example illustrating the proposed scheme is also presented.

1. Knowledge-based system design: an overview

Knowledge-based frameworks consider design as a technological activity in which knowledge about a specific domain is used to represent design artifacts, constraints and requirements. It is a process that seeks all relevant knowledge and combines it to produce a design solution. Design is often considered as a search in which a satisfactory design solution is produced from a number of alternatives [5,35]. The search proceeds in a design space whose elements are design objects (components) and attributes (parameters).

Correspondence to: Prof. J.W. Rozenblit, Department of Electrical and Computer Engineering, The University of Arizona, Tucson, AZ 85721, United States.

* The work of this author has been supported by External Relations Grant "Knowledge-Based Support of Modular, Hierarchical Simulation Model Management" from McDonnell Douglas Corporation.

Design frameworks differ mainly in the underlying knowledge representation scheme and the search methods employed for solution generation. However, all the methodologies attempt to capture and enumerate alternative solutions in a design domain. Design knowledge should be organized in such ways that it can be manipulated effectively and efficiently. The system design approach proposed in [20,28,29] termed knowledge-based simulation design, focuses on the use of modeling and simulation techniques to build and evaluate models of the system being designed. It treats design as a series of activities that include the following phases: 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.

In the ensuing sections, phases of the methodology are briefly summarized. We then focus on the knowledge representation and management aspect of the discussed framework and show how the proposed representation scheme can effectively support the generation of design solutions.

1.1. Design problem formulation

As pointed out earlier, design is often considered as a search problem. In our approach, a target design model (a goal state) should be generated which best satisfies design constraints and requirements. Thus, the problem can be formulated as follows. Given a set of design objectives, constraints and requirements OCR, find a design model DM* such that: $DM^* = best\{DM_i \mid i = 1, \ldots, n\}$, where each DM_i is a design model that satisfies the set of constraints OCR (a set of objectives, constraints and requirements), and "*best*" is a function ranking and selecting the design alternatives DM.

The knowledge-based simulation design methodology provides a set of methods for generating a solution to the above problem. The solution process consists in constructing a set of alternative design models DM_is, simulating their behavior, and selecting the model DM* [21,24].

The design model construction process begins with developing a representation of design components and their variants. Thus a knowledge representation scheme is needed to capture 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 is a representation for the kinds of variants that are possible for an object, i.e., how it can be categorized and subclassified. The synthesis (coupling) constraints impose a manner in which components identified in decompositions can be connected together. The selection constraints limit choices of variants of objects determined by the taxonomic relations.

Beyond this, procedural knowledge should be available to select and synthesize the system's components identified in the chosen representation scheme. This selection and synthesis process is called *pruning* [25,26]. Pruning results in a recommendation for a *model composition tree*, i.e., the set of hierarchically arranged entities corresponding to model components.

Performance of design models is evaluated through computer simulation. Alternative design models are evaluated with respect to experimental conditions (experimental frames [36]) that reflect 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.

The first subproblem in generating a design solution is to find a set of design model structures (composition trees) that conform to static design constraints and requirements. (By *static*, we mean the constraints and requirements whose satisfaction can be accomplished prior to simulation of a design model's behavior.)

1.2. Rule-based model structure generation

The rule-based generation of admissible model structures requires that a knowledge base that contains rules for selection and configuration of the systems' components be specified. Production rules [33] are used to represent design objectives, constraints, user's requirements and performance expectations. The generation of structure, called pruning, can be interpreted as a search directed by constraints through the search space consisting of design objects, their variants identified in taxonomic relationships and their decompositions.

The following steps are required to provide the rules that guide pruning the space of design components given by a structured representation scheme: (1) for each taxonomic relationship specify a set of rules for selecting a design component (object); (2) for a design object with several decompositions specify rules for generating a unique decomposition; (3) for each decomposition specify synthesis rules that ensure that the objects selected from specializations and aspects are configurable, i.e., the components they represent can be validly coupled. Each rule can be assigned a certainty factor indicating the rule's degree of applicability. Rule-based pruning is an effective means of generating recommendations for composition trees that satisfy static design constraints [26].

1.3. Simulation and evaluation

We separate the model description from a simulation experiment under which the model is observed. This facilitates much greater flexibility in design model ranking and relieves the models from the burden of collecting data about themselves. A set of circumstances under which a model is to be observed and experimented with is called an *experimental frame*. Zeigler [36] has shown that an experimental frame can be realized as a coupling of three components: a generator (supplying a model with an input segment reflecting the effects of the external environment upon a model), an acceptor (a device monitoring a simulation run) and a transducer (collecting and processing model output data).

The data collected from alternative design model runs within respective experimental frames are compared in order to select the best design solution. Clearly, design performance measures may conflict with one another. Therefore, Multiple Criteria Decision Making (MCMD) methods are used for ranking candidate designs. Some MCMD criteria and simple examples of design trade-offs are presented in [8].

To gather up the strands, we now summarize the design solution generation phases.

1.4. Design generation phases revisited

(1) We conceptualize decompositions and specializations of components of the system being designed using a structured knowledge representation scheme.

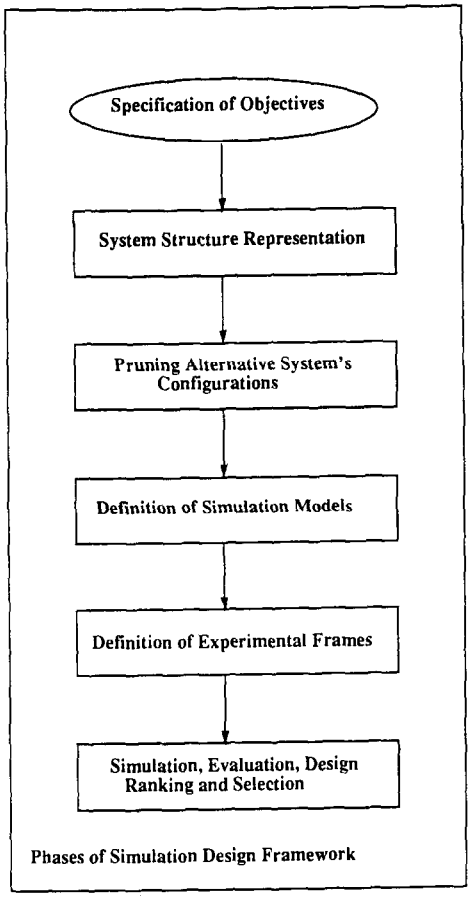


Fig. 1. Model-based system design.

(2) We develop a rule base to be used in the pruning process.

(3) We invoke the pruning engine to generate recommendations for candidate solutions to the design problem in the form of model composition trees.

(4) We synthesize models from the composition trees obtained in phase (3).

(5) To carry out a simulation experiment, we specify an experimental frame.

(6) We evaluate simulation results and rank models with respect to the performance measures that express design objectives and requirements. MCMD criteria are used to define the "*best*" function which generates $\mathrm{DM}^* = best\{\mathrm{DM}_i \mid i = 1, \ldots, n\}$.

Figure 1 illustrates these phases.

2. Design knowledge representation

In the last decade the technology of knowledge-based systems has been widely used in solving various engineering problems. The expected contributions of knowledge engineering to CAD/CAM are the integration of complex system's components and the construction of an

efficient model for the design process [17]. To be successful in these efforts, design knowledge has to be organized properly so that it can be manipulated effectively and efficiently. Generally, performance of a knowledge-based system is determined by its knowledge management scheme. Unfortunately, the optimal knowledge management strategy is usually application-dependent. Up to now, there is no universal scheme that covers all the diverse design applications. To assure the high quality of a knowledge-based system design process, more powerful management schemes are needed.

When reviewing the common traits of design practice, one finds that structured schemes (i.e., hierarchy, modularity and regularity) are used to reduce the complexity of the design process [32]. The use of *hierarchy* involves dividing a system into subsystems and then repeating this operation on subsystems until the complexity of the subsystems is at a desired abstraction level. A *modular* design approach facilitates flexibility and future modifications. Modularity helps designers reduce the complexity of system models and clarify an approach to a problem. *Regularity* denotes employing a regular structure at all design levels to simplify the design process. In order to increase the efficiency of design processing and knowledge management, design knowledge must be organized in a way that reflects such common traits.

Thus, to describe the structure of the system being designed and its topology, a structure is needed that embodies knowledge about *decomposition, taxonomy, coupling* and *design attributes*. As indicated in Section 1, decomposition knowledge means that the structure has schemes for representing the manner in which an object is decomposed into components. Taxonomy captures variants that are possible for an object, i.e., how it can be categorized and subclassified. Coupling information provides communication links and indicates how component models are synthesized to form the overall system's model. Attributes characterize static properties and the dynamic behavior of a system. In addition to the declarative knowledge listed above, we need procedural knowledge that can be used to support the designer in carrying out the design process. Such knowledge includes rules for selecting alternative system components, procedures for evaluating performance of a design prototype, and methods for modifying and varying design parameters.

Although different schemes such as *production rules* [15], *frames* [14], *semantic networks* [19], *AND / OR trees* [16], *predicate logic* [4] and the *system entity structure* [36] have been introduced for knowledge representation and management, none of them provides enough expressive power when applied to system design individually. In the ensuing sections, we refine the system entity structure concept by integrating it with the frame and production rule specifications. This integration results in a hierarchical, entity-based knowledge management scheme called Frames and Rules Associated System Entity Structure (FRASES) [8,9]. We demonstrate how FRASES supports various design phases of our methodology. First, however, a summary of different representations is given.

2.1. Frames

Frames are primarily intended to handle declarative knowledge [33]. A frame is a generalized property list which can be divided into discrete elements called "slots". Each slot describes an attribute which may contain one or more facets such as "value", "default" and/or "if-needed". Facets can have many values. A value can be either a number, a symbol, a string or a procedure. Procedures that are activated automatically when a value is needed (i.e.,

if-needed), when a value is placed (i.e., if-added) or when a value is removed (i.e., if-removed) are called *demons*. Although a frame is a convenient means for representing declarative knowledge, it does not provide efficient schemes for managing procedural knowledge.

2.2. Production rules

Production rules represent knowledge in two parts: *condition and action*. Each rule is written in an "IF-THEN" clause. The "IF" part states a situation or a premise and the "THEN" part states an action or a conclusion. A rule is triggered if current facts match the antecedent (condition) part of the rule. Conflict resolution strategies [33] are applied to select the rule to be fired when multiple rules are triggered in forward chaining reasoning. Once a rule is fired, its action part is carried out [16,33].

Although production rule-based systems support flexible inferencing on procedural knowledge, they do not provide explicit schemes for managing declarative knowledge related to system design architectures. During knowledge refinement, whenever a rule is modified, the entire rule base must be searched in order to find all related rules that need to be updated. This increases the cost of maintaining a knowledge base.

2.3. The system entity structure

As a step toward a complete knowledge representation scheme for design support we have combined the decomposition, taxonomic and coupling relationships in a knowledge representation scheme called the *System Entity Structure* (SES) [36]. Knowledge representation is now generally accepted to be a key ingredient in designing artificial intelligence software. Previous work [20,23,27] identified the need for representing the structure and behavior of systems in a declarative scheme related to frame-theoretic and object-based formalisms [37,38]. The elements represented are motivated, on the one hand, by systems theory [13,34] concepts of decomposition (i.e., how a system is hierarchically broken down into components) and coupling (i.e., how these components may be interconnected to reconstitute the original system). On the other hand, systems theory has not focused on taxonomic relations, as represented for example in frame-hierarchy knowledge representation schemes. In the SES scheme, such representation concerns the admissible variants of components in decompositions and the further specializations of such variants.

The interaction of decomposition, coupling and taxonomic relations in an SES affords a compact specification of a family of models for a given domain. In a system entity structure, *entities* refer to conceptual components of reality for which models may reside in a model base. Also associated with entities are slots for attribute knowledge representation. An entity may have several *aspects*, each denoting a decomposition, and therefore having several entities. An entity may also have several *specializations*, each representing a classification of possible variants of the entity.

The construction of, and operations on a system entity structure are governed by the following axioms.

(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

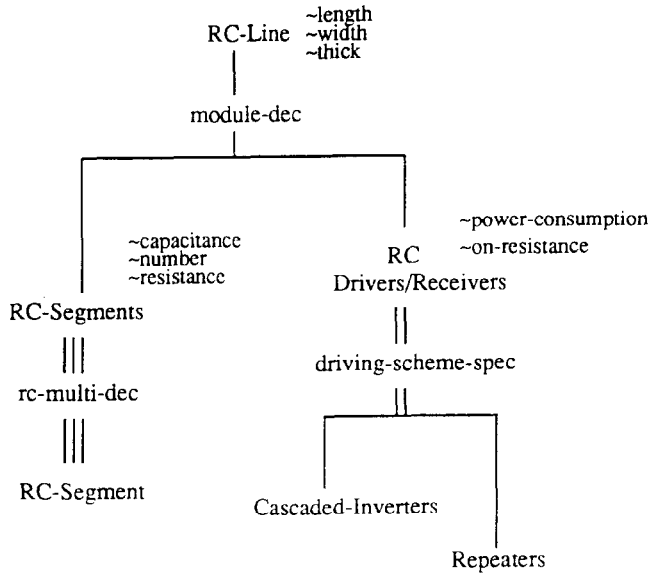


Fig. 2. System entity structure representation of *RC-line*.

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.

(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.

(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.

(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.

(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.

For illustration, consider Fig. 2 that shows a system entity structure representation of a lumped *RC-line* model for a VLSI interconnection design. As shown in the figure, an RC-Line can be decomposed (denoted graphically as |) into functional modules: *Drivers / Receivers* and *RC-Segments*. Two design alternatives (denoted by ||), *Repeaters* and *Cascaded-Inverters*, are used for the realization of *Drivers / Receivers*. Since the number of *RC-Segments* may vary with the selection of *Drivers / Receivers*, a multiple decomposition (denoted |||) is used for representing entities whose number may vary in the system.

The system entity structure organizes possibilities for a variety of system decompositions and, consequently, a variety of model constructions. Its generative capability facilitates convenient

definition and representation of models and their attributes at multiple levels of aggregation and abstraction. More complete discussions of the system entity structure and its associated structure transformations are presented in [20,28,36].

The SES represents the structural information about a system in a hierarchical and modular manner. However, it does not support the management of procedural knowledge. Thus, we propose to combine the system entity, frames and production rule representations into a single comprehensive scheme that can be effectively exploited in the design process.

3. Frames and rules associated system entity structure (FRASES)

FRASES combines the three aforementioned schemes. An underlying data structure of FRASES is the system entity tree. Each node of the SES tree has a frame attached to it that encompasses declarative and procedural knowledge in a design problem. Such a frame is called *Entity Information Frame* (EIF). An Entity Information Frame (EIF) integrates design knowledge by providing slots for representing design procedural knowledge. Thus, the SES layer assures that the decomposition, taxonomic and coupling relationships are properly specified while each EIF captures the following information about the node to which it is attached:

$$\text{EIF} = (\text{MD, AT, DS, SR, PR, LK}),$$

where MD is the name of the simulation model specified for the entity node; AT are attributes of the entity; DS is the design specification for the component represented by the entity; SR are the simulation requirements for the model of the entity; PR are production rules for design structure pruning and synthesis; LK are links (pointers) to other FRASES nodes.

In our methodology, the behavior of a system component (represented by an entity and its EIF in the FRASES tree) is determined by a simulation model defined in the model base. To extract the model specification for simulation, a key (i.e., model name MD) is provided. As shown in Fig. 3, the *Minimum-Size-Repeater* (a leaf entity) has a corresponding model called MSR defined in the model base. Therefore, the MD slot of the EIF associated with *Minimum-Size-Repeater* is: `(MD (name (value MSR)))`.

The AT-slot contains attributes that characterize the static and dynamic properties of an associated object [22]. In general, attributes of an object can be categorized into three types: static variables, design parameters and performance indices. Static variables are constant attributes that should not be changed during the design process. Each static variable is usually initialized with a default value by a database query or by a user-provided function. Design parameters are variables related to behavioral characteristics of the associated object. Performance of the target system is determined by the combination of design parameters. To assure a valid design that meets all system requirements, each design parameter is associated with a quantitative function for boundary checking and primary estimation. Performance indices are variables used to evaluate the system performance.

For illustration we describe the *Minimum-Size-Repeater*. We may provide information such as "node-type" (static variable), "input-capacitance" (design parameter), "output-resistance" (design parameter), "power-dissipation" (performance index) and "50%-delay" (performance index). To calculate the "power-dissipation" and "50%-delay", quantitative functions called "MSR-power-fun" and "MSR-delay-fun" are used. The corresponding EIF representation of a *Minimum-Size-Repeater* is:

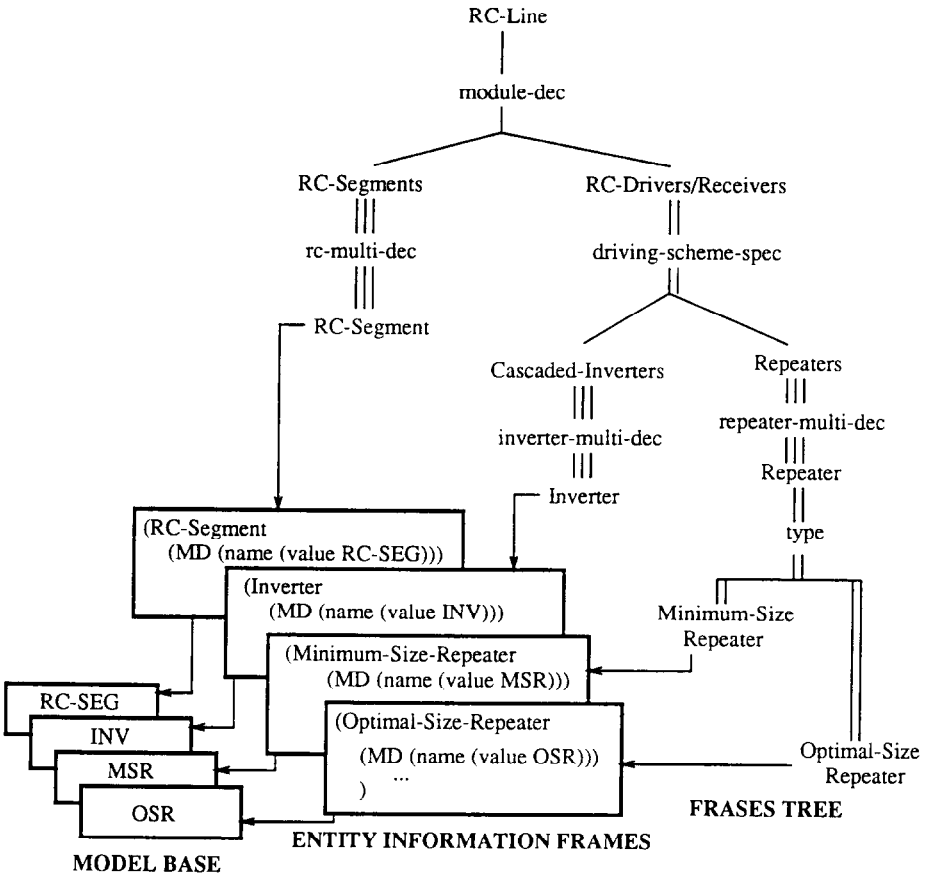


Fig. 3. FRASES representation of *RC-line*.

```
(Minimum-Size-Repeater
  (MD (name (value MSR)))
  (AT (node-type entity)
    (input-capacitance
      (unit (default micro-farad))
      (if-needed (data-query MSR input-capacitance)))
    (output-resistance
      (unit (default ohm))
      (if-needed (data-query MSR output-resistance)))
    (power-dissipation
      (unit (default watt))
      (if-needed MSR-power-fun))
    (50%-delay
      (unit (default micro-second))
      (if-needed MSR-delay-fun)))
```

The Design Specification (DS) slot accepts design specifications such as design objectives, system constraints and criteria preference that must be satisfied by the target system. Consider again the *Minimum-Size-Repeater* as an example. We may want to impose design constraints on "50%-delay" and "power-dissipation". Then, a typical design specification for the *Minimum-Size-Repeater* is given in the following frame:

```
(DS (constraints (<50%-delay 0.1)(< power-dissipation 1))
  (objectives (minimize power-dissipation 50%-delay))
  (preference (rank 50%-delay power-dissipation))
)
```

Design constraints imply requirements that must be satisfied by the resulting system. Each design constraint is expressed by the "relation", the "index" and the "value". Design objectives imply the design goal. Typical specification of design objectives are stated to maximize and/or minimize one or more performance indices. The specification of design objectives guides appropriate application of Multi-Criteria Decision Making (MCDM) [8,24].

Design preference conveys the designer's preference over a set of performance indices. The preference scheme guides the system in selecting the best design model based on the desired MCDM method. Four types of preference schemes (i.e., unknown, complete, ranking and fuzzy) [8,24] are allowed to express the preference over criteria. Under the unknown preference, all design criteria are regarded as having equal importance. The complete preference scheme is used when the user is able to specify exact preference for each criterion. The ranking preference is used whenever partial preference information is available but is not comprehensive enough to give the exact preference values of the criteria. The fuzzy preference allows the user to define the preference range of criteria with the lower and upper bound.

The Simulation Requirements (SR) slot defines experimental circumstances under which a design model is to be simulated such as input segments and control schemes. For example, if a unit step function is selected as input and the simulation is run for 5 time units, then the FRASES representation for the associated SR-slot takes the following form:

```
(SR (arrival (value unit-step))
  (control (run-time (value 5))))
```

The PR-slot contains production rules for pruning design alternatives and configuring design model structures at different levels of abstraction. Constraints derived from the architecture, technology and available resources are translated into production rules to assist in design reasoning. In FRASES, selection rules for pruning alternative components are defined in the PR slot of the Entity Information Frames of specialization nodes; synthesis rules for configuring design objects are defined in the EIFs of decomposition nodes. For instance, to determine the driving scheme for the *RC-Line*, the selection rule is defined for the specialization node *rc-driving-scheme* as follows:

```
(rc-driving-scheme
  (AT (node-type (value specialization)))
  (PR (rcd-r1: if (> line-resistance (* 7 MSR.on-resistance))
                then (prune Cascaded-Inverters)))
);; `prune ' means to remove
```

Synthesis rules are defined to indicate how component models are coupled together to form a larger model. For example, if a coupling function named "repeater-coupling-fun" is provided, we define a synthesis rule for the *repeater-multi-dec* as:

```
(repeater-multi-dec
  (AT (node-type (value multi-decomposition)))
  (PR (rcm-r1: if (=operation-phase coupling)
                then (repeater-coupling-fun Repeaters number)))
)
```

The LK slot defines pointers in the FRASES tree. For example, the *rc-driving-scheme* has parent nodes *RC-Drivers / Receivers* and children nodes *Repeaters* and *Cascaded-Inverters*. This information is represented in the EIF as shown below:

```
(rc-driving-scheme
  (AT (node-type (value specialization)))
    ...
  (LK (children Repeaters Cascaded-Inverters)
    (parent RC-Drivers/Receivers))
)
```

In general, the structure for each EIF slot is the following template:

```
⟨MD⟩=(MD ⟨name⟩)
⟨AT⟩=( (⟨name⟩ (value ⟨atom⟩)) (tuning ⟨list⟩)
       (if-needed ⟨list⟩) (boundary ⟨list⟩))
⟨DS⟩=( (constraints ⟨list⟩) (preference ⟨list⟩)
       (objectives (max [⟨name⟩]) (min [⟨name⟩])))
⟨SR⟩=( (arrival ⟨list⟩) (control ⟨list⟩))
⟨PR⟩=( (⟨name⟩ if ⟨list⟩ then ⟨list⟩))
⟨LK⟩=( (children ⟨name⟩) (parents ⟨name⟩))
⟨name⟩=string
⟨atom⟩=symbol|string | number
⟨list⟩=function|logic-expression
```

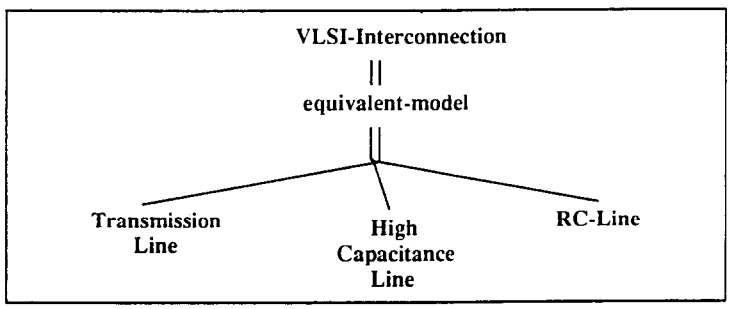
By exploiting the reasoning flexibility provided by production rules, the efficiency in representing declarative knowledge offered by frames, and the visibility and hierarchy supported by system entity structures, FRASES is a powerful and efficient scheme that supports modern system design. In the next section, we illustrate its application to a high-level design of VLSI interconnects.

4. Example — VLSI interconnect design

VLSI package modeling denotes the development of equivalent electrical circuits for describing the physical structure that makes up a given package. The overall modeling procedure for microelectronics package design begins with specifying the physical parameters of the mechanical structure. For example, a single interconnection has known line lengths (l),

widths (w) and thickness (t). Resistivity (ρ) of conductor materials, dielectric constant (μ) of insulator materials and mobility of carriers (ϵ) are normally limited by the capability of the existing manufacturing process and materials. Once the parameters for the mechanical structure are known, the package's electrical design commences with a modeling procedure that converts the structure into an equivalent electrical circuit. The result is a linear network made up of resistances (R), inductances (L), and capacitance (C) [18,30,31].

Although transmission line analysis gives the correct answer irrespective of the rise time, the same result can be obtained with similar accuracy using lumped approximations when the rise time is five times larger than the time of flight delay. The lumped model calculations are much simpler than the transmission line analysis. Based on this observation, we may classify a VLSI interconnection into three lumped equivalent models: *High-Capacitance-Line, RC-Line* and *Transmission-Line*. Rules are associated with the specialization node (*equivalent-model*) to




```
(VSLI-Interconnection
    (AT (node-type (value "ent"))
        (line-length (if-needed ask) (unit cm))
        (design-rule (if-needed ask) (unit micron))
        (IC-technology (if-needed ask) (boundary "cmos nmos bipolar GaAs"))
        (package-technology (if-needed ask)
              (boundary "WSI Ceramic-Hybrid Thin-Film Printed-Wiring-Board))
        (conductor (if-needed ask) (boundary "Au Al Cu Ag"))
        (line-capacitance (if-needed
            (/ (* (query dielectric dielectric-constant)
                  line-width line-length) dielectric-thick)) (unit farad))
        (time-of-flight-delay (if-needed
            (/ (* sqrt (query package-technology relative-dielectric-constant))
                  line-length) 30)) (unit nsec))
        (transistor-on-resistance (value (query IC-technology on-resistance)) (unit ohm) )
         ... ...
    (LK (parent "") (children equivalent-model)) ... )

(equivalent-model
    (AT (node-type (value "spec")))
    (PR (em1: if (< signal-rise-time (* 3 time-of-flight-delay))
              then (prune RC-Line High-Capacitance-Line))
        (em2: if (> signal-rise-time (* 3 time-of-flight-delay))
              then (prune Transmission-Line))
         ... ...
    (LK (children High-Capacitance-Line RC-Line Transmission-Line)
        (parent VLSI-Interconnection)) ... )

(RC-Line (AT (node-type (value "ent")) ....
```

Fig. 4. FRASES representation for equivalent models of VLSI interconnect.

guide a proper selection of equivalent models. To depict the geometry of an interconnection, parameters such as *line-width, line-length, line-thickness* are associated with a query function for acquiring physical information about the conductors. Similarly, a query function is associated with *conductor-material, package-technology* and *IC-technology* for acquiring processing information. All these user-provided parameters are contained in the Entity Information Frame (EIF) of *VLSI-Interconnection*. Note that the knowledge defined in the EIF of *VLSI-Interconnection* will be inherited by *High-Capacitance-Line, RC-Line* and *Transmission-Line*.

To provide information for design constants (i.e., resistivity, magnetic permeability, relative permittivity and dielectric constant), we use associated database queries for data acquisition. After all related parameters are acquired, quantitative functions are used to estimate propagation-speed, line-resistance, time-of-flight-delay and signal-rise-time. This results in the FRASES and its associated EIFs as shown in Fig. 4. Figure 5 shows more design detail, as described below.

To further the design detail, the *RC-Line* is decomposed into *RC-Segments* and *RC-Drivers / Receivers* circuits. Since the number of *RC-Segments* depends on the realization of *RC-Drivers / Receivers*, synthesis constraints and coupling information about *RC-Segments* and *RC-Drivers / Receivers* must be defined in the EIF of *rc-module-dec*. There are two alternatives

```
(rc-module
   (AT (node-type (value "dec")))
   (PR (rc1 if (select? Repeaters)
            then (set Repeaters.number RC-Segments.number))
       (rc2 if (select? Cascaded-Inverters)
            then (set RC-Segments.number 1))
       (rc3 if (and (select? Repeaters) (equal? phase "coupling"))
            then (link RC-Line.in RC-Segments[1].in)
                 (for (i=1; i <= Repeaters.number; i++)
                   (link Repeaters.out[i] RC-Segments.in[i])
                 (if (< i Repeaters.number)
                  (link RC-Segments.out[i] Repeaters.in[i+1])))
                 (link RC-Segments[RC-Segments.number].out
                      RC-Line.out)))
       ... ...
   (LK (parent RC-Line)
       (children RC-Segments RC-Driver/Receiver)))

(Repeaters
   (AT (node-type (value "ent"))
       (number (if-needed (sqrt
         (/ (* 0.4 RC-Line.line-resistance
                   RC-Line.line-capacitance)
           (* 0.7 (query minimum-size-buffer input-capacitance)
             (query minimum-size-buffer output-resistance)))))
        (boundary (> number 2))) )
       ... ...
   (LK (parent rc-driving-scheme)
       (children repeater-multi-dec)))

(rc-multi-dec
   (AT node-type (value "m-dec")))
   (PR ...
```

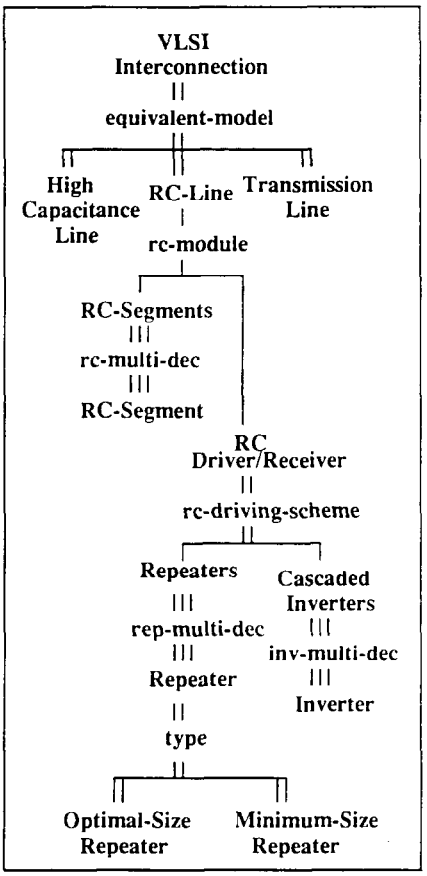


Fig. 5. FRASES representation for equivalent *RC-line* model of VLSI interconnect.

(*Repeaters* and *Cascaded-Inverters*) for the realization of *RC-Drivers-Receivers*. If the *Repeaters* is selected, the number of *RC-Segments* is equal to the number of *Repeaters*. On the other hand, if the *Cascaded-Inverters* is selected, the number of *RC-Segments* is equal to one. To assure the strict hierarchy of FRASES knowledge, this synthesis constraint is specified in the EIF of *rc-module-dec*. Rules for selecting the proper driving scheme(s) for drivers/receivers circuits are defined in the EIF of *rc-driving-scheme*. For example, the *Repeaters* is selected when the RC constant of the wire is seven times the minimum-size buffer.

Both *Minimum-Size-Repeater* and *Optimal-Repeater* can be used to drive signals for an RC line. In order for the repeaters to reduce the overall delay, their number must be at least two. The optimal number of repeaters depends on the performance requirements. To properly describe the object whose number may vary with system requirements, a multiple decomposition (triple bars) is used in FRASES. To determine the optimal number of repeaters, a quantitative routine is associated with the *number* attribute in the EIF of *Repeaters*. This number function is inherited by both specialized variants: *Minimum-Size-Repeaters* and *Optimal-Repeaters*. For optimal repeaters, an additional procedure for estimating the size increase factor is required. Coupling among individual repeaters is defined in the EIF of *repeater-mult-dec*. The above expansion of an *RC-Line* is illustrated in Fig. 5.

A similar approach for decomposing the *RC-Line* is applied to *High-Capacitance-Line* and *Transmission-Line*. For example, driving methods such as *Cascaded-Inverters, Precharge-to-VDD, Static-Sense-Amplifier, Clocked-Sense-Amplifier* and *Precharge-to-VDD / 2* can be used to re-

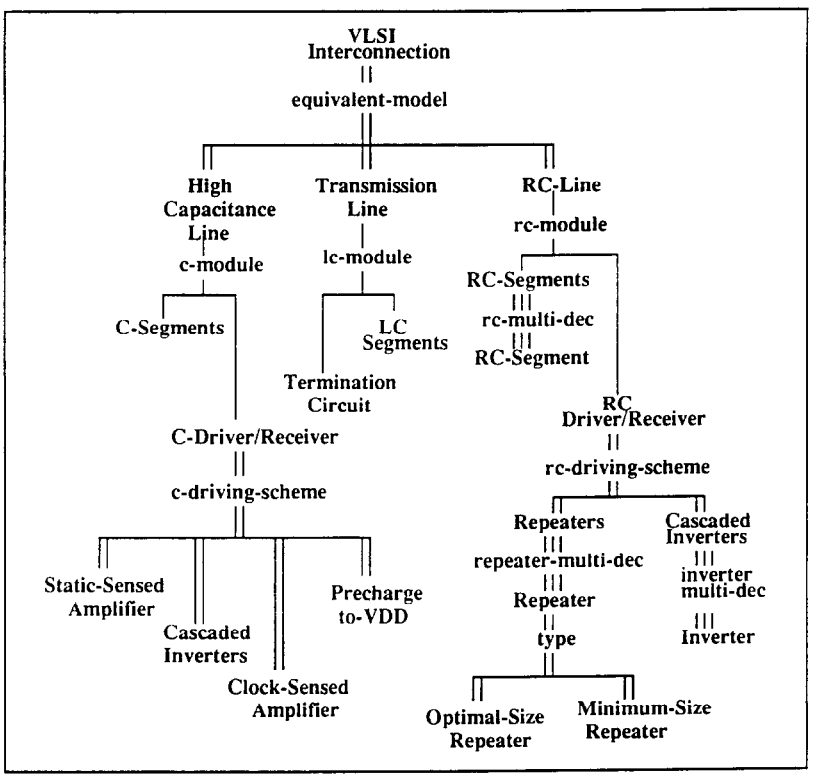


Fig. 6. FRASES tree of VLSI interconnect.

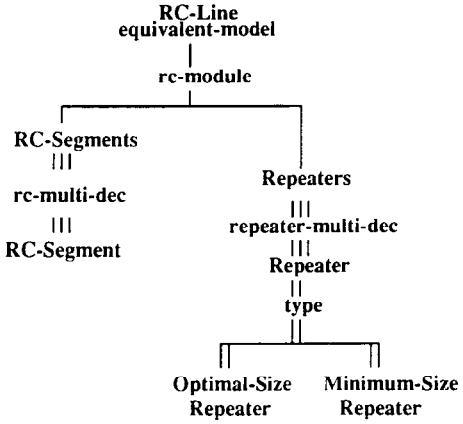


Fig. 7. FRASES tree generated after pruning.

duce the propagation delay in a high-capacitance line. Rules for selecting appropriate driving circuits are derived and are attached to the specialization node called *c-driving-scheme*. For transmission lines, rules must determine the termination scheme using resistors, capacitors, diodes and transistors. The resulting FRASES tree is given in Fig. 6. The Appendix contains a detailed description of Entity Information Frames for the interconnect design.

The application domain of the current FRASES structure can be expanded to assist in design modeling of multiple conductors by adding design knowledge for coupled inductance, coupled capacitance and topologies of interconnection networks. More design details about electrical VLSI package modeling can be found in [1,3,6,7,12,18,30,31].

After the design knowledge is built into FRASES, design specifications are defined to drive design knowledge processing. Each entity node of the FRASES tree has its own design specification. This results in a hierarchical system design specification. For example, the

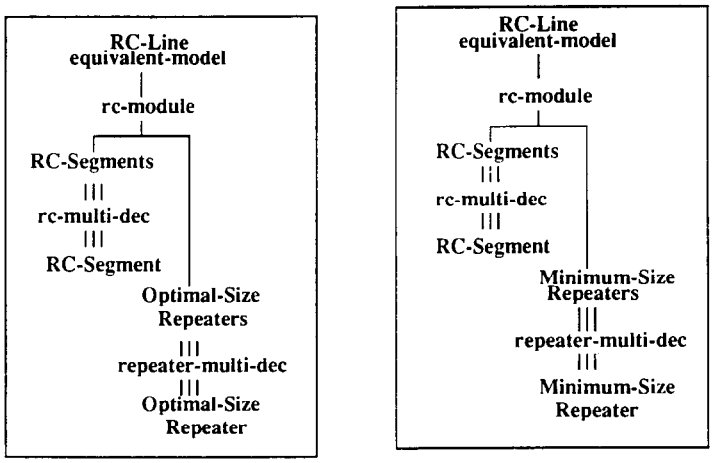


Fig. 8. Alternatives for *RC-line* design model structures.

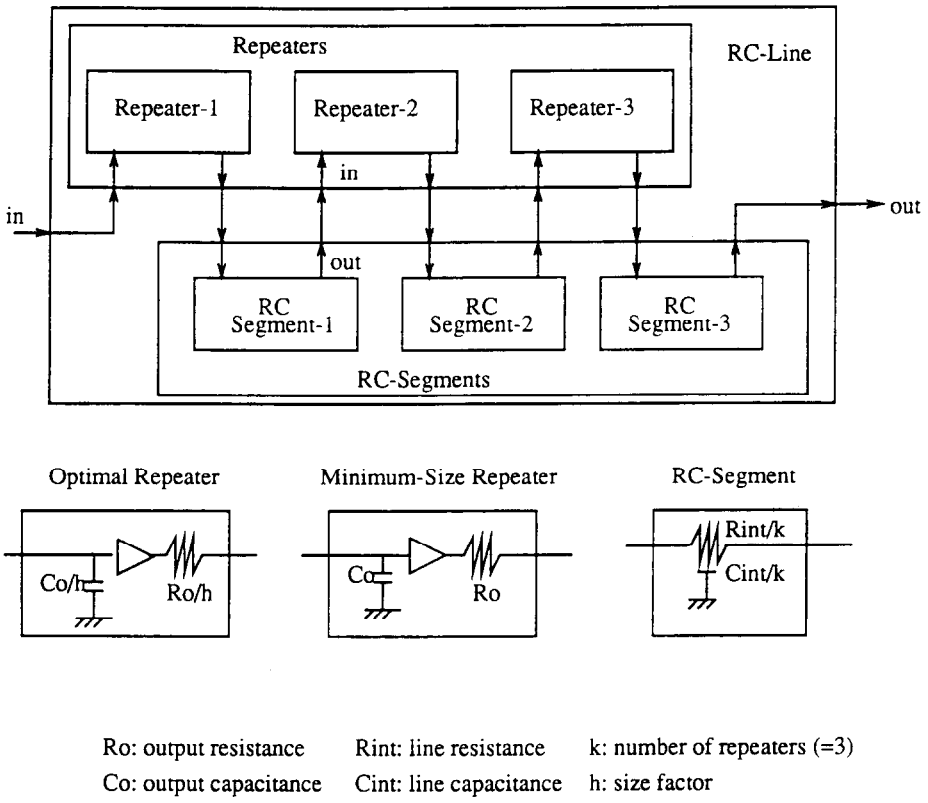


Fig. 9. Synthesis of *RC-line* design models.

designer may set performance constraints (e.g., 50% delay) for both *VLSI-Interconnection* and *RC-Drivers / Receivers*, which appear at different abstraction levels of the hierarchy.

To derive admissible design models, pruning is conducted on the FRASES structure by interpreting the selection rules associated with all specialization nodes. The inferencing order among specialization nodes is determined by the selected search strategy (e.g., breadth-first or depth-first). If breadth-first search is selected, inferencing will be performed on the rules associated with the *equivalent-model* first. Suppose the rule inferencing on *equivalent-model* results in, both the *Transmission-Line* and *High-Capacitance-Line* being pruned out of the FRASES structure. Then, the inference engine proceeds and interprets the rules associated with the rc-driving-scheme in order to determine the appropriate drivers/receivers circuits.

The FRASES tree shown in Fig. 7 has been generated by the inference process where *Repeaters* are selected for the realization of driver/receiver circuits. To derive the optimal *number* of *Repeaters*, a quantitative function associated with the *number* slot of *Repeaters* is invoked. The pruned FRASES can be transformed into two alternative design structures as shown in Fig. 8. By interpreting synthesis and coupling rules associated with decomposition nodes, two design model structures are synthesized (Fig. 9) in a hierarchical manner. Then, they be used in a circuit simulation program for performance evaluation.

5. Synopsis of FRASES' characteristics

We emphasize the unique characteristics of the proposed knowledge representation and management scheme. These features indicate that employing the scheme in engineering design affords several advantages.

- *Flexibility*: The hierarchical and modular nature of FRASES enables the designer to overcome the complexity of representation in a divide-and-conquer manner. With FRASES, design knowledge is described from an abstract level to a more specific level that can be expressed by a mathematical model. At each design level, details of decomposition can be modified as the technology evolves and changes. Unlike other schemes that appear efficient in a specific problem domain, FRASES can be used in design of any system exhibiting a hierarchical and modular structure.
- *Efficiency*: The inheritance and uniformity axioms highly reduce the size of a design knowledge base required for the same design application. At each inference cycle, only the rules associated with the focus node need be examined. Subtrees that do not meet system requirements are cut at an early pruning stage to reduce the search space. Thus, the inferencing time is reduced significantly. Via verifying axioms of FRASES, query/validation rules for acquiring essential design knowledge are generated automatically to facilitate the development of a knowledge base [10].
- *Manageability*: FRASES reduces the complexity of a knowledge base. The uniformity axiom prevents multiple definitions of an object and assures the consistency of design knowledge. Due to the strict hierarchy of FRASES, related knowledge of an entity can be easily allocated by examining (a) the tree path from the entity to the root or (b) the subtree of the entity. This eliminates the need for searching the entire knowledge base, and thus facilitates the refinement of design knowledge. From the software point of view, the scheme encompasses all elements of an object-oriented model (i.e., *encapsulation, abstraction, hierarchy and modularity* [2] that assure the minimum software complexity and cost for future maintenance and expansion.

6. Summary

An efficient scheme called Frames and Rules Associated System Entity Structure (FRASES) that organizes complex design knowledge into a hierarchical, entity-, frames- and rule-based structure was presented. The scheme integrates knowledge and its management for model-based system design applications. FRASES facilitates complex design knowledge acquisition, representation, inference, and refinement. With it, the efficiency of design knowledge representation and management is highly improved.

Our current efforts focus on the realization of the theory-based concepts in the form of an integrated knowledge-based system design and simulation environment. A prototype implementation of FRASES and associated pruning procedures has been recently completed in Common Lisp [11].

Appendix — entity information frames for VLSI interconnect

```
(VLSI-Interconnection
   (AT (node-type (value entity))
       (line-width (if-needed ask) (unit (value micron)))
       (line-length (if-needed ask) (unit (value cm)))
       (line-thick (if-needed ask) (unit (value micron)))
       (design-rule (if-needed ask) (unit (value micron)))
       (dielectric-thick (if-needed ask) (value (unit micron)))
       (dielectric (if-needed ask)
            (boundary polymide silicon-dioxide epoxy-glass alumina))
       (IC-technology (if-needed ask)
                     (boundary cmos nmos bipolar GaAs) )
       (package-technology  (if-needed ask)
            (boundary Wafer-Scale-Integration Ceramic-Hybrid
                      Thin-Film-Hybrid Printed-Wiring-Board))
       (conductor-material (if-needed ask) (boundary Au Al Cu Ag))
       (line-capacitance (unit farad) (if-needed (/ (*
            (query dielectric dielectric-constant)
            line-width line-length) dielectric-thick)) )
       (signal-rise-time  (unit nsec)
          (if-needed (data_query IC-technology rise-time)) )
       (time-of-flight-delay (if-needed (/ (* (sqrt
            (data_query package-technology
                        relative-dielectric-constant))
            line-length) 30)) (unit nsec))
       (transistor-on-resistance
         (data_query IC-technology on-resistance) (unit ohm))
   (LK (parent nil) (children equivalent-model)) )

(equivalent-model
    (AT (node-type (value specialization)) )
    (PR (em-r1: if (< signal-rise-time (* 5 time-of-flight-delay))
                then (prune RC-Line) and
                     (prune High-Capacitance-Line) )
        (em-r2: if (> signal-rise-time (* 5 time-of-flight-delay))
                then (prune Transmission-Line)
        (em-r3: if (> line-resistance Transistor-On-Resistance)
                then (prune High-Capacitance-Line))  )
    (LK (parent VLSI-Interconnection)
          (children High-Capacitance-Line
                    RC-Line Transmission-Line)))
```

```
(RC-Line
    (AT (node-type (value entity))
        (line-resistance (unit ohm) (if-needed
             (/ (* (data_query conductor-material resistivity)
                   line-length) (* line-width line-thick))) )
    (LK (parent equivalent-model) (children rc-module-dec)))

(rc-module
   (AT (node-type (value decomposition)))
   (PR (r1: if (equal? RC-Driver-Receiver Repeaters)
            then (set! RC-Segments.number Repeaters.number))
       (r2: if (equal? RC-Driver-Receiver Cascaded-Inverters)
               then (set! RC-Segments.number 1))
       (r3: if (equal? operation-phase coupling) and
               (equal? RC-Driver-Receiver Repeaters)
            then (RC-Line.in -> Repeaters.in[1])
                 (for (i=1; i<=Repeaters.number; i++)
                    (Repeaters.out[i] -> RC-Segments.in[i])
                    if (i==Repeaters.number)
                       (RC-Segments.out[i] -> RC-Line.out)
                    else (RC-Segments.out[i] -> Repeaters.in[i+1]) ))
       (r2: if (equal? operation-phase coupling) and
               (equal? RC-Driver-Receiver Cascaded-Inverters)
            then (RC-Line.in -> RC-Segments.in)
                 (RC-Segments.out -> Cascaded-Inverters.in)
                 (Cascaded-Inverters.out -> RC-Line.out) )))
   (LK (parent RC-Line) (children RC-Segments RC-Driver-Receiver)) )

(rc-multi-dec
   (AT (node-type (value multiple-decomposition)))
   (PR (rcm-r1:
        if (equal? operation-phase coupling)
        then (for (i=1; i<= RC-Segments.number; i++)
                (RC-Segments.in[i] -> RC-Segment[i].in)
                (RC-Segments.out[i] -> RC-Segment[i].out) )))
   (LK (parent RC-Segments) (children RC-Segment)))

(rc-driving-scheme
   (AT (node-type (value "spec")))
   (PR (rcd-r1:
         if (> line-resistance
            (* 7 (data_query minimum-size-buffer on-resistance)))
         then (prune Cascaded-Inverters)))
   (LK (parent RC-Driver-Receiver)
         (children Repeaters Cascaded-Inverters)))
```

```
(Cascaded-Inverters
   (AT (node-type (value entity))
       (number (if-needed
          (ln (/ (+ RC-Line.line-capacitance RC-Line.load-capacitance)
               (data_query minimum-size-buffer input-capacitance)))))
   (LK (parent c-driving-scheme) (children nil)))

(Repeaters
   (AT (node-type (value entity))
       (number
        (if-needed (sqrt
          (/ (* 0.4 RC-Line.line-resistance RC-Line.line-capacitance)
             (* 0.7 (data_query minimum-size-buffer input-capacitance)
               (data_query minimum-size-buffer output-resistance)))))
        (boundary (> 2)) ))
   (LK (parent rc-driving-scheme) (children repeater-multi-dec)))

(repeater-multi-dec
   (AT (node-type (value multiple-decomposition)))
   (PR (rpm-r1:
         if (equal? operation-phase coupling)
         then (for (i=1; i<=Repeaters.number; i++)
                 (Repeaters.in[i] -> Repeater[i].in)
                 (Repeaters.out[i] -> Repeater[i].out) )))
   (LK (parent Repeaters) (children Repeater)))

(inv-multi-dec
   (AT (node-type (value multiple-decomposition)))
   (PR (ivm-r1:
         if (equal? operation-phase coupling)
         then (Cascaded-Inverters.in -> Inverter[1].in)
              (for (i=1; i<number-1; i++)
                   (Inverter[i].out -> Inverter[i+1].in))
              (Inverter[i].out -> Cascaded-Inverters.out) ))
   (LK (parent Cascaded-Inverters) (children inverters)))

(Repeater
   (AT (node-type (value entity))
       (50%-delay (if-needed (* 2.5 (sqrt
          (* (data_query minimum-size-buffer output-resistance)
             (data_query minimum-size-buffer input-capacitance)
             RC-Line.line-resistance
             RC-Line.line-capacitance)))) )
   (LK (parent repeater-multi-dec) (children repeater-type)))

(Inverter
```

```
(MD (value inverter))
(AT (node-type (value entity))
    (50%-delay (if-needed
     (+ (* 0.4 VLSI-Interconnection.line-resistance
              VLSI-Interconnection.line-capacitance)
        (* 0.7 VLSI-Interconnection.line-resistance
              VLSI-Interconnection.load-capacitance)
        (* 1.9 (data_query minimum-size-buffer output-resistance)
              (data_query minimum-size-buffer input-capacitance)
              (ln (/ (+ VLSI-Interconnection.line-capacitance
                        VLSI-Interconnection.load-capacitance)
              (data_query minimum-size-buffer input-capacitance))))))
      (unit micro-sec)))
(LK (parent inv-multi-dec) (children nil)))
```

References

[1] H. Ameniya, Time-domain analysis of multiple parallel transmission lines, *RCA Rev.* 27 (1967) 241–246.

[2] G. Booch, *Object-Oriented Design with Applications* (Benjamin/Cummings, Menlo Park, CA, 1991).

[3] P.A. Brennan, N. Raver and A.E. Ruehli, Three dimensional inductance computations with partial element equivalent circuits, *IBM J. Res. Develop.* 23 (6) (1979) 661–668.

[4] C.L. Chang and R.C. Lee, *Symbolic Logic and Mechanical Theorem Proving* (Academic Press, New York, 1973).

[5] J.S. Gero et al., An approach to knowledge-based creative design, in: *Proc. NSF Engineering Design Research Conference*, Amherst (Univ. of Massachusetts Press, Amherst, MA, 1989) 333–346.

[6] A.J. Gruodis and C.S. Chang, Coupled lossy transmission line characterization and simulation, *IBM J. Res. Develop.* 25 (1) (1981) 25–41.

[7] S.T. Ho and S.K. Mullick, Analysis of transmission lines on integrated circuit chips, *IEEE J. Solid State Circuits* SC-Z (4) (1967) 201–208.

[8] J. Hu, Towards a knowledge-based design support environment for design automation and performance evaluation, Ph.D. Dissertation, Univ. Arizona, Tucson, 1989.

[9] J. Hu, Y.M. Huang and J.W. Rozenblit, FRASES — a knowledge representation scheme for engineering design, *Advances in AI and Simulation*, Soc. Comput. Simulation Ser. 20 (4) (1989) 141–146.

[10] J. Hu and J.W. Rozenblit, Knowledge acquisition based on representation (KAR) for design model development, in: P. Fishwick and R. Modjeski, eds., *Knowledge-based Simulation: Methodology and Application* (Springer, New York, 1991) 77–94.

[11] Y.M. Huang, Knowledge-based generation of design model structures: towards on object-oriented multiprocessing architecture, Ph.D. Dissertation, Univ. Arizona, Tucson, 1991.

[12] G.A. Katopis, Delta-I noise specification for a high performance computing machine, *Proc. IEEE* 73 (9) (1985) 1405–1415.

[13] M.D. Mesarovic, and Y. Takahara, *General Systems Theory: Mathematical Foundations* (Academic Press, New York, 1975).

[14] M. Minsky, A framework for representing knowledge, in: P.H. Winston, ed., *The Psychology of Computer Vision* (McGraw-Hill, New York, 1975) 211–277.

[15] A. Newell and H.A. Simon, *Human Problem Solving* (Prentice-Hall, Englewood Cliffs, NJ, 1972).

[16] N.J. Nilsson, *Problem-Solving Methods in Artificial Intelligence* (Mc-Graw-Hill, New York, 1971).

[17] S. Ohsuga, Conceptual design of CAD systems involving knowledge bases, in: J.S. Gero, ed., *Knowledge Engineering in Computer-Aided Design* (North-Holland, Amsterdam, 1985).

[18] O.A. Palunsinski, J.C. Liao, P.E. Teschan, J.L. Prince and F. Quintero, Electrical modeling of interconnections in multilayer packaging structures, *IEEE Trans. Components, Hybrids, and Manufacturing Tech.* CHMT-10 (2) (1987) 217–223.

[19] M.R. Quillian, Semantic memory, Scientific Report, Air Force Cambridge Research Laboratories (1968) 216–270.
[20] J.W. Rozenblit, A conceptual basis for integrated, model-based system design, Technical Report, Dept. Electrical Comput. Engrg., Univ. Arizona, Tucson, 1986.
[21] J.W. Rozenblit, Simulation modeling in design generation and solution, in: A. Jávor, ed., *Proc. 1990 IMACS European Simulation Meeting*, Esztergom, Hungary (1990) 135–141.
[22] J.W. Rozenblit and J. Hu, Experimental frame generation in a knowledge-based system design and simulation environment, in: M.S. Elzas, T.I. Ören and B.P. Zeigler, eds., *Modeling and Simulation Methodology. Knowledge Systems' Paradigms* (North-Holland, Amsterdam, 1989) 451–466.
[23] J.W. Rozenblit, J. Hu and Y.M. Huang, An integrated, entity-based knowledge representation scheme for system design, in: *Proc. NSF Engineering Design Research Conference*, Amherst (Univ. of Massachusetts Press, Amherst, 1989) 393–408.
[24] J.W. Rozenblit, J. Hu, Tag Gon Kim and B.P. Zeigler, Knowledge-based design and simulation environment (KBDSE): foundational concepts and implementation, *J. Oper. Res. Soc.* 41 (2) (1990) 475–489.
[25] J.W. Rozenblit and Y.M. Huang, Constraint-driven generation of model structures, in: A. Thesen, H. Grant and W.D. Kelton, eds., *Proc. 1987 Winter Simulation Conference*, Atlanta, GA (Soc. for Computer Simulation, 1987).
[26] J.W. Rozenblit and Y.M. Huang, Rule-based generation of model structures in multifacetted modelling and system design, *Oper. Res. Soc. Amer. J. Comput.* 3 (4) (1991) 330–344.
[27] J.W. Rozenblit and B.P. Zeigler, Entity-based structures for model and experimental frame construction, in: M.S. Elzas and B.P. Zeigler, eds., *Modelling and Simulation Methodology in the Artificial Intelligence Era* (North-Holland, Amsterdam, 1986) 78–100.
[28] J.W. Rozenblit and B.P. Zeigler, Design and modelling concepts, in: R. Dorf, ed., *International Encyclopedia of Robotics, Applications and Automation* (Wiley, New York, 1988) 308–322.
[29] J.W. Rozenblit and B.P. Zeigler, Knowledge-based simulation design methodology: a flexible test architecture application, *Trans. Soc. Comput. Simulation* 7 (3) (1990) 195–228.
[30] A.E. Ruehli, Survey of computer-aided electrical analysis of integrated circuit interconnections, *IBM J. Res. Develop.* 23 (6) (1979) 626–639.
[31] A.E. Ruehli and P.A. Brennan, Capacitance models for integrated circuit metallization wires, *IEEE J. Solid State Circuits* 10 (6) (1975) 640–651.
[32] N. Weste and K. Eshraghian, *Principles of CMOS VLSI Design — A System Perspective* (Addison-Wesley, Reading, MA, 1985).
[33] P.H. Winston, *Artificial Intelligence* (Addison-Wesley, Reading, MA, 2nd ed., 1984).
[34] A.W. Wymore, *A Mathematical Theory of Systems Engineering: The Elements* (Wiley, New York, 1967).
[35] J. Yang and J.W. Rozenblit, Case studies of design methodologies: a survey, in: B.P. Zeigler and J.W. Rozenblit, eds., *Proc. Internat. Conf. on AI, Simulation and Planning in High Autonomy Systems* (IEEE Computer Soc. Press, Silver Spring, MD, 1990) 136–141.
[36] B.P. Zeigler, *Multifacetted Modelling and Discrete Event Simulation* (Academic Press, London, 1984).
[37] B.P. Zeigler, Hierarchical, modular discrete-event models in an object-oriented environment, *Simulation* 50 (5) (1987) 219–230.
[38] B.P. Zeigler, *Object-Oriented Simulation with Hierarchical Modular Models: Intelligent Agents and Endomorphic Systems* (Academic Press, New York, 1990).