ILL: 79539355

ILLiad TN: 589898

Borrower: AZU

Ship via: *Odyssey*

Service Level: GWLA

Ariel: 150.135.238.50

Email: askddt@u.library.arizona.edu

Fax: (520) 621-4619

Odyssey: 150.135.238.6

Lending String: *AFU,IXA,TXI,UIU,PAU

ISSN: 9780333551172

OCLC: 26673474

ILL - AFU
UNIVERSITY OF ARKANSAS
365 N MCILROY AVE
FAYETTEVILLE AR 72701-4002

RETURN POSTAGE GUARANTEED

LIBRARY MAIL

UNIVERSITY OF ARIZONA
LIBRARY - ILL
1510 EAST UNIVERSITY DRIVE
TUCSON AZ 85721-0055

Date: **6/29/2011 8:32:57 AM**

Call #: **Q335 .A7877 1992**

Location:

Volume:

Issue:

Year:

1992

Pages:

Journal Title: Artificial intelligence in operational research /

Article Author:

Article Title: ; Knowledge-Based Design and Simulation Environment (KBDSE); Foundational Concepts and Implementation

Notice: This material may be protected by Copyright Law (Title 17 U.S. Code)

Charge
Maxcost: **$50IFM**

Patron: **Hwang, George**

Initials: _____

Shelf: _____ Per: _____

Sort: _____ ILL: _____

Bad Cite: _____

Years checked _____

Table of Contents / Index _____

Copyright © 1990 Operational Research Society Ltd

Knowledge-based Design and Simulation Environment (KBDSE): Foundational Concepts and Implementation

JERZY W. ROZENBLIT,[1] JHYFANG HU,[2] TAG GON KIM[3] and BERNARD P. ZEIGLER[1]

[1]Department of Electrical and Computer Engineering, The University of Arizona, Tucson, Arizona,
[2]Department of Electrical Engineering, Tulane University, New Orleans, Louisiana, and
[3]Department of Electrical and Computer Engineering, University of Kansas, Lawrence, Kansas, USA

Research developments leading to implementation of an intelligent software environment supporting system design and simulation are presented. Knowledge-based system design and multifaceted simulation methodologies are a foundation for the system realization. The paper describes the major theoretical concepts and processes employed to develop and simulate design models. The environment implementing these concepts and methods consists of two basic components: one serves as a front end supporting the model construction processes; the other is an object-oriented, discrete-event simulator supporting evaluation of hierarchical, multi-component models. Current state of the system implementation and future work are discussed.

Key words: modelling, simulation, system design

INTRODUCTION

Despite great strides in development of computational tools such as high performance workstations intended to help to cope with the rising complexity of designs, the design process remains error prone. Given the often severe constraints imposed by cost, environmental impacts, safety regulations, etc., it is a fact of life that designers are forced to make compromises that would not be necessary in an ideal world. Simulation is increasingly recognized as a useful tool in assessing the quality of sub-optimal design choices and arriving at acceptable trade-offs.

We have focused on developing and implementing a methodology of design in which design models can be synthesized and tested using computer simulation. This framework, termed knowledge-based system design and simulation,[1–3] lends itself to realization in the form of an integrated, intelligent design support environment.

Our work complements recent trends in simulation modelling research which emphasize the development of integrated software modelling support environments.[4–7] Such environments are envisioned as conglomerates of tools that will aid modellers in the model construction process and simulation program generation. There are several notable features of the existing software prototypes that 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. Second, 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. We also observe emergence of artificial intelligence (AI) applications that assist the modeller in model construction and validation, simulation management and analysis.[5,8]

In the ensuing sections, we characterize the basic tenets of our design modelling approach. We then describe the architecture of the software system and explain how simulation model development is supported by the environment. We conclude with a brief description of current applications and work in progress on extending the system.

MULTIFACETED MODELLING AND SYSTEM DESIGN

Multifaceted methodology denotes a modelling approach which recognizes the existence of multiplicities of objectives and models in any simulation project. It provides formal representation

schemes that support the modeller in organizing the model construction process, aggregating partial models and specifying simulation experiments.[9] Modelling objectives drive three fundamental processes in the methodology: they facilitate the construction, retrieval and manipulation of design entity structures,[10] selection of model structures, and specification of experimental conditions under which design models are evaluated by a simulation study.

The design entity structure is a knowledge representation scheme based on a tree-like graph that encompasses the boundaries, decompositions and taxonomic relationships that have been perceived for the system being modelled. An entity signifies a conceptual part of the system which has been identified as a component in one or more decompositions. Each such decomposition is called an aspect. Thus entities and aspects are thought of as components and decompositions, respectively. In addition to decompositions, there are relations termed specializations. A specialization relation facilitates representation of variants for an entity. These are called specialized entities and inherit properties of an entity to which they are related by the specialization relation.

Aspects can have coupling constraints attached to them. Coupling constraints restrict the way in which components (represented by entities) identified in decompositions (represented by aspects) can be joined together.

In addition to coupling constraints, there are selection constraints in the system entity structure. Selection constraints are associated with specializations of an entity. They restrict the way in which its subentities may replace it in the model construction process. Synthesis constraints restrict ways in which entities selected from specializations may be configured to represent the structure of the system being designed.[11,12] Later, we describe the process that employs the production rule formalism to support automatic selection of entities and synthesis of a design model structure. We call this process rule-based driven design model structure generation.

Models can be expressed in special formalisms depending on the problem at hand. Typical specifications include differential equations, finite state machine or discrete event. Each formal model description specifies a system and selects a class of subsystems by placing constraints on the possible static and dynamic structures it encompasses. A characterization of such constraints is given by Murray and Sheppard.[13] The model construction process involves the specification of the static and dynamic structure. In our system, models are developed using discrete event system specification (DEVS) formalism.[9] This formalism underlies the construction of models in our simulation environment—DEVS-SCHEME.

The DEVS formalism

The DEVS hierarchical, modular formalism, as implemented in DEVS-SCHEME, closely parallels the abstract set theoretic formulation developed by Zeigler (see Kim and Zeigler[9]). In such a formalism, one must specify basic models from which larger ones are built, and how these models are connected together in a hierarchical fashion. A basic model, called an atomic DEVS, is defined by the following structure:[9]

$$M = < X, S, Y, \delta_{\text{int}}, \delta_{\text{ext}}, \lambda, t_a >$$

where X is a set (external input event types),
S is a set (sequential states),
Y is a set (external output event types),
δ_{int} is a function (internal transition specification),
δ_{ext} is a function (external transition specification),
λ is a function (output function) and
t_a is a function (time advance function)

with the following constraints:

(i) the total state set of the system specified by M is

$$Q = \{(s, e) \mid s \in S, 0 \leqslant e \leqslant t_a(s)\},$$

(ii) δ_{int} is a mapping from S to S:

$$\delta_{\text{int}}: S \rightarrow S,$$

(iii) δ_{ext} is a function:

$$\delta_{ext}: Q \times X \rightarrow S,$$

(iv) t_a is a mapping from S to the non-negative reals with infinity,

$$t_a: S \rightarrow R \text{ and}$$

(v) λ is a mapping from S to Y:

$$\lambda: S \rightarrow Y.$$

An interpretation of the DEVS and a full explication of the semantics of the DEVS are found in Kim and Zeigler.[9]

The second form of models, called a coupled model, tells how to couple several component models together to form a new model. This latter model can itself be employed as a component in a larger coupled model, thus giving rise to the hierarchical construction. A coupled DEVS is defined as a structure:[9]

$$DN = \langle D, M_i, I_i, Z_{ij}, SELECT \rangle$$

where D is a set (component names), and for each i in D:

M_i is a component and

I_i is a set (influences of i),

and for each j in I_i:

Z_{ij} is a function, (i-to-j output translation) and

$SELECT$ is a function (tie-breaking selector)

with the following constraints:

$$M_i = \langle X_i, S_i, Y_i, \delta_i, \lambda_i, t_{ai} \rangle$$

I_i is a subset of D, i is not in I_i

Z_{ij}: $Y_i \rightarrow X_j$

$SELECT$: subsets of $D \rightarrow D$

such that for any non-empty subset E, $SELECT(E)$ is in E.

The formal model specification in multifaceted methodology consists in specifying the system entity structure and attached variable types (called descriptive variables), pruning and then specifying a discrete-event model for the components identified by the pruned entity structure. Selection of input, output and state variables results in the model's static structure. Definition of transition and output functions adds the dynamic components to the DEVS specification.

Clearly, a formal set theoretical description of a large-scale system would be a tedious and impractical process. In fact, this may well have been a reason why theory-based approaches have been shunned by simulation practitioners, and a primary motivation for the development of software implementing the above formal modelling concepts.

SIMULATION MODEL DEVELOPMENT IN KBDSE

The basic organization of the software under development is given in Figure 1. There are two fundamental modules in the system:

(i) the module supporting entity structure programming and pruning (ESPP) and

(ii) the module supporting simulation and performance analysis (DEVS-SCHEME).

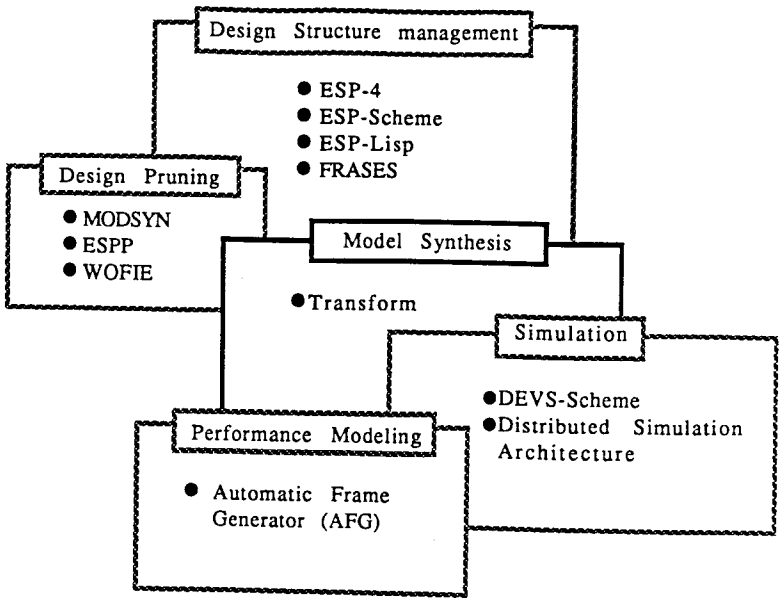


FIG. 1. *Organization of software in KBDSE.*

The modules are interfaced through the TRANSFORM procedures that automatically generate simulation code. The code is generated by retrieving from the model base simulation modules associated with the composition tree generated by the pruner. We now proceed to describe these modules in more detail.

The entity structuring program and pruner (ESPP)

This program helps the modeller conceptualize and record the decompositions underlying a model (or family of models) before, during and after development. To the extent that ESPP is used before beginning model development, it is a tool for assisting in top-down model design. However, when additions and changes are made as the development proceeds, ESPP 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.

We have augmented the system entity structure into an integrated, entity-oriented knowledge representation scheme, termed the frame and rule-associated system entity structure (FRASES). FRASES is a scheme that combines concepts of the system entity structure, frame,[14] and production rules.[15,16] 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 the system entity structure, FRASES is a powerful and efficient scheme for managing domain knowledge supporting design model development.

Structure of FRASES

FRASES is a superclass of the system entity structures which encompasses the boundaries, decompositions and taxonomic relationships of the system components being modelled. All axioms and operations defined originally for managing system entity structures are also present in FRASES representation.

A typical example of FRASES for representing a *LAN*-based distributed system is shown in Figure 2. As shown in the figure, each entity of FRASES is associated with an entity information frame (EIF). Every occurrence of an entity has the same EIF and isomorphic substructure. During application, knowledge contained in the EIF is extracted and interpreted by the inference engine for design reasoning.[17,18] EIF is a structure:

$$\langle M, ATTs, DSF, ESF, CRS, CH \rangle$$

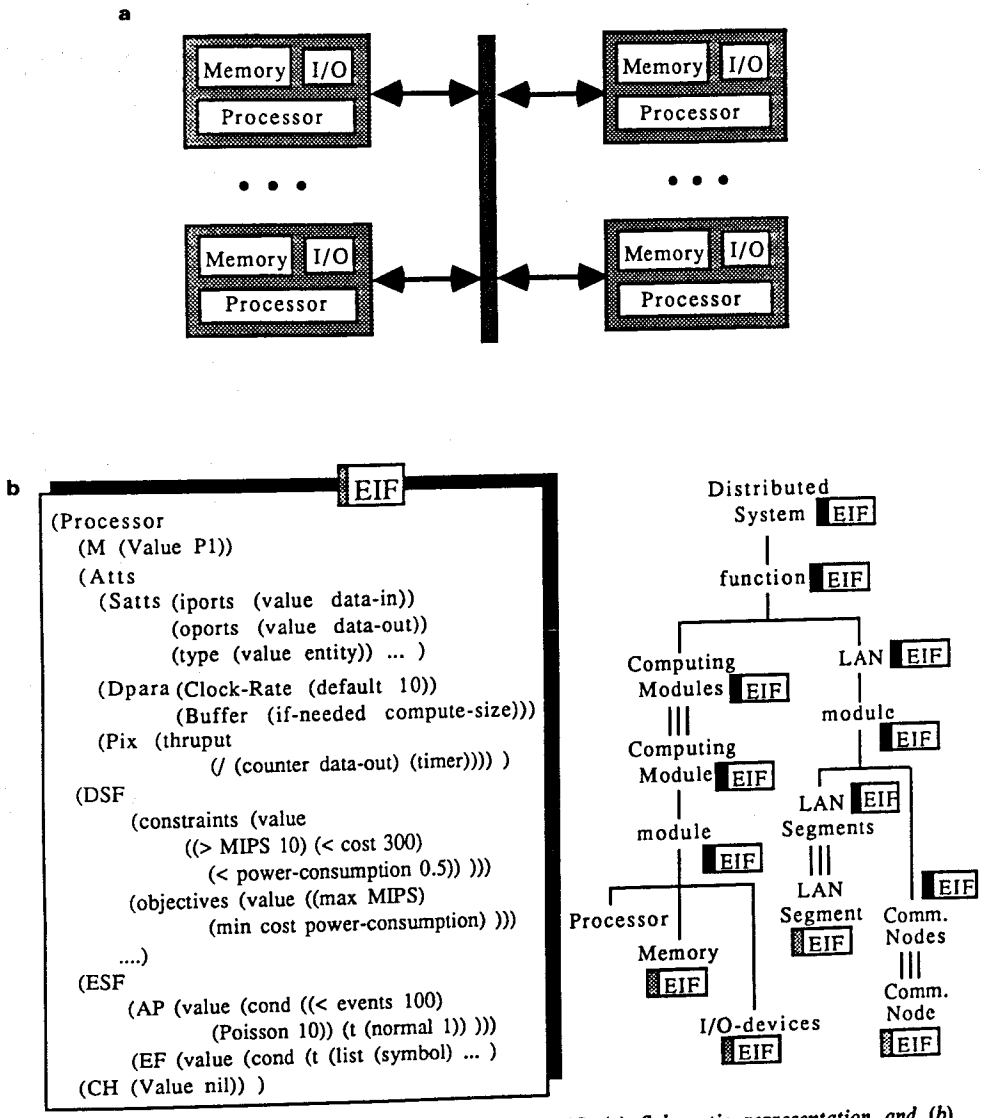


FIG. 2. *A* LAN-*based distributed system with FRASES.* (*a*) *Schematic representation and* (*b*) *FRASES representation.*

where *M* is the name of the associated model,
*ATT*s are attributes of *M*,
DSF is the design specification form,
ESF is the experiment specification form,
CRS are constraint rules for design model synthesis, and
CH are FRASES children of the focus node.

With FRASES representation, behavioural knowledge about objects is described by simulation models stored in the model base. *M* represents the name of the entity and serves as a major key to access its model.

*ATT*s are attributes used to characterize the associated object. Attributes of an entity are partitioned into two groups, i.e. static and dynamic. Static attributes are variables used to describe properties of an object that do not change over time. Dynamic attributes are related to dynamic behaviour of the models represented by entity objects.

The design specification form (*DSF*) accepts the specification of design objectives, constraints and criteria weighting schemes. The contents of the *DSF* define the system requirements that must be satisfied by the system to be designed. *DSF* information is used to guide the synthesis of design model structures. Each entity of FRASES has its own *DSF*. Once composition trees (or design structures) are generated based on the knowledge provided in the *CRS* slot, users are requested to

define the simulation experiment in the *ESF*. Finally, simulation is activated via automatic extraction and coupling of simulation models.

The experimental specification form (*ESF*) is applied to accept the specification of simulation requirements such as an arrival process, event structure and simulation control scheme. The *ESF* provides information to direct the automatic generation of experimental frames.[19] An experimental frame specifies a limited set of circumstances under which a system is to be observed or subjected to experimentation. Again, the *ESF* is placed together with entity nodes of a composition tree (i.e. a decomposition tree with information about the coupling schemes among model components).

Constraint rules for synthesis (*CRS*) contain heuristic rules for configuring design model structures. Formally, selection constraint rules for pruning alternatives are associated with specialization nodes, and constraint rules for synthesizing components are associated with aspect nodes. Model development driven by production rules will be described in the next section.

Rule-based synthesis of model structures

The production rule formalism supports automatic selection of entities from taxonomic relationships and synthesis of structures underlying the simulation models.

The process consists of defining selection and synthesis rules and associating them with entity information frames of the design entity structure. The modeller invokes the inference engine which, through a series of queries based on the constraint rules, allows him to consult on an appropriate structure for the modelling problem at hand. The result is a recommendation for a model composition tree.[9] The composition tree is used by the DEVS-SCHEME environment to retrieve models from the model base. The retrieved models are automatically linked in a hierarchical manner according to the coupling constraints.

The prototype pruning module was originally designed in PROLOG and called MODSYN (model synthesizer).[20] It was subsequently redesigned in COMMON LISP and incorporated in the ESPP shell.[21] The basic components of the pruner are the knowledge base and the inference engine.

To prune the design structure, we generate the following rule sets:

(i) *Selection rule set:* each selection rule stands for a choice of an entity in a specialization.

(ii) *Synthesis rule set:* after selection rules have been applied to the entity structure, synthesis rules ensure proper configuration of the selected entities. They also co-ordinate the actions of the selection rules. Certainty factors are employed to indicate the applicability of the rules.

Selection rules are associated with the specialization nodes whereas the synthesis rules are attached to the decomposition nodes of FRASES. Each rule set can be regarded as a module. Therefore the entire rule base is constructed in a hierarchical manner imposed by the entity structure.

The production rule formalism is used to express modelling objectives, constraints and requirements in the form of selection and synthesis rules. Domain experts provide knowledge about admissible choices of design components and their combinations, design data regarding expected performance given a particular component choice, etc.

Inference engine design

The inference engine uses the strategy of 'generate and test', i.e. it takes the initial data from the user and the hypothesis generated by the knowledge base to prune the search space tree. In other words, the engine attempts to match the data with the information contained in the knowledge base. If the data match, the engine 'climbs up' the tree, trying to prove the next hypothesis. We use aspect ordering in order to eliminate aspects not desirable in the model we are constructing, and specialization-oriented pruning to select unique entities for the model composition trees. A complete description of the shell can be found in Rozenblit and Huang.[20] The LISP realization of the shell provides facilities for top-down as well as bottom-up pruning and selection of different search control strategies.[21]

Unlike other applications, engineering designs usually require components of a system to be designed in a particular sequence. Essential components are always determined before other com-

ponents can be designed. The design sequence may be altered by environmental factors, problem domains or technical constraints. This requires a flexible search scheme to conduct the design reasoning process in the right sequence. In order to capture the dynamics of a design sequence, a weight-oriented FRASES inference engine (WOFIE) was proposed.[17] By appropriately setting up the priority of a specialization node, WOFIE is capable of emulating the design reasoning process conducted by a human expert.

DEVS-SCHEME SIMULATION ENVIRONMENT

DEVS-SCHEME,[22–27] a general purpose modelling and simulation environment, is an implementation of DEVS formalism in SCOOPS, the LISP-based, object-oriented superset of PC-SCHEME. It runs on DOS-compatible PCs and the TI's Explorer LISP machine. DEVS-SCHEME is implemented as a shell that sits upon SCHEME in such a way that all of the underlying LISP-based and object-oriented programming language features are available to the user. The result is a powerful basis for combining artificial intelligence and simulation techniques. Since structure descriptions in DEVS-SCHEME are accessible to run-time modification, it provides a convenient basis for development of variant family and variable structure simulation models. DEVS-SCHEME also serves as a medium for developing computer architectures for distributed simulation of hierarchical, modular discrete-event models.[28]

In DEVS-SCHEME, component models called atomic models are specified using SCHEME's semantics, which correspond closely to the formal definition of DEVS. The input and output sets consist of pairs (port, value). Thus, $x = (p, v)$ signals the receipt of a value v at an input port p. The elements of DEVS formalism take the following form in the DEVS-SCHEME:

Internal transition function: (*define* (*int s*) ...)

External transition function: (*define* (*ext s e x*) ...)

Output function: (*define* (*out s*) ...)

Time advance function: (*define* (*ta s*) ...)

where ... represents function body definitions expressed in SCHEME.

The atomic models may be coupled together to form a model at the coupled specification level. DEVS-SCHEME is still under development. Recently, new features for testing model morphism and model simplification have been incorporated in the shell.[23,25,29]

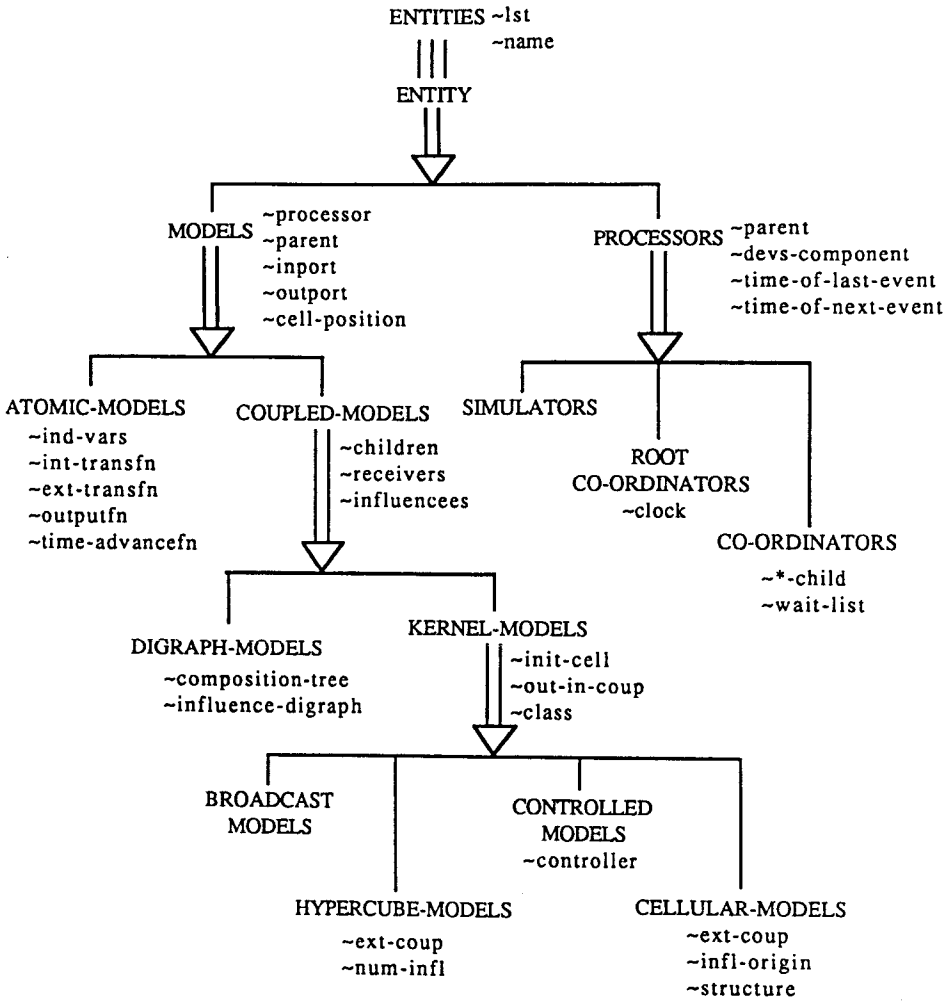
The class specialization hierarchy in DEVS-SCHEME is shown in Figure 3. All classes in DEVS-SCHEME are subclasses of the universal class entities which provide tools for manipulating objects in these classes (these objects are hereafter called entities). The inheritance mechanism ensures that such general facilities need only be defined once. Entities of desired class may be constructed using a method *mk-ent* and destroyed using a method *destroy*. More specifically, *mk-ent* makes the entity and places it in a class variable list which maintains the list of members of the given class; *destroy* removes the entity from this list. Every entity has a name which is assigned to it upon creation.

Models and *processors*, the main subclasses of entities, provide the basic constructs needed for modelling and simulation. Models are further specialized into the major classes *atomic-models* and *coupled-models*, which realize atomic DEVS and coupled DEVS, respectively. The *coupled-models*, in turn, are specialized into more specific cases, a process which may be continued indefinitely as the user builds up a specific model base. *Kernel-models*, one subclass of *coupled-models*, is a generalized class whose subclasses provide powerful means of defining complex, hierarchical multicomputer architectures formed by recursive compounding of component models for basic processing elements of such architectures. Class processors, on the other hand, have three specializations: *simulators*, *co-ordinators* and *root-co-ordinators*. These carry out the simulation of a model in a manner which follows the hierarchical abstract simulator concepts.[9,30]

Due to the object-oriented realization, subclasses of existing classes and new classes can be readily added to DEVS-SCHEME as required. As a result the DEVS-SCHEME environment:

(i) supports modular, hierarchical model construction,

(ii) allows independent testing of components models,



* Uppercase Letters: Classes
* Lowercase Letters: Class/Instance Variables

FIG. 3. *Class hierarchy in DEVS-SCHEME*

(iii) separates models from experimental frames, and
(iv) supports distributed simulation.

Details of all classes in DEVS-SCHEME along with their instant/class variables and methods are available.[23,25]

Rule-based model retrieval and transformation

A pruned entity structure can be synthesized into a simulation model by the operation transform. As the algorithm visits each entity in the pruned entity structure, transform calls upon a retrieval process that searches a model corresponding to the current entity. If one is found, it is used and transformation of the entity subtree is aborted. The retrieval process proceeds by evaluating rules, which consist of retrieval rules (pairs of condition and retrieval action) and conflict resolution rules, by which a rule is selected if there is more than one which satisfies conditions. Details of these rules are found in Zeigler.[29]

A rule for searching a model that corresponds to the current entity says that it first looks for the model in the working memory, then in the model base (MBASE) and finally, if the current entity is is a leaf, in the entity structure base (ENBASE). Before searching the model, another rule checks the name of the current entity. If the current entity has a base name and a non-trivial extension (the extension starts with numbers or '&'), the base name is used as an entity name for the retrieval process. As more than one rule is satisfied when evaluated, a conflict resolution rule fires

only one rule. For example, if both Rule 1 and Rule 2 are satisfied, then Rule 1 is fired. We employ context specificity, which means that the rule with a more specific condition than other rules is fired, in order to resolve such a conflict.

If a pruned entity structure is found in the ENBASE in the searching process, a transform is invoked and executed in a separate SCHEME environment so as not to interfere with the current environment. Since the self-invocation can occur in a leaf entity only, such local transformation is definitely recursive.

Hierarchical model construction in DEVS-SCHEME

The DEVS-SCHEME environment provides layers of objects and methods which may be used to achieve more powerful features. The knowledge base framework shown in Figure 4 is intended to be generative in nature, i.e. it should be a compact representation scheme which can be unfolded to generate the family of all possible models synthesizable from components in the model base. The user, whether human or artificial, should be a goal-directed agent which can interrogate the knowledge base and synthesize a model using pruning operations that ultimately reduce the structure to a composition tree.

As shown in Figure 4, model objects expressed in DEVS-SCHEME must reside in working memory in order to be simulated. Such an object can be reconstructed from disk file definitions by direct evaluation (the only possibility for atomic-models) or by applying the transform function to a pruned entity structure in working memory. The pruned entity structure is in turn obtained by

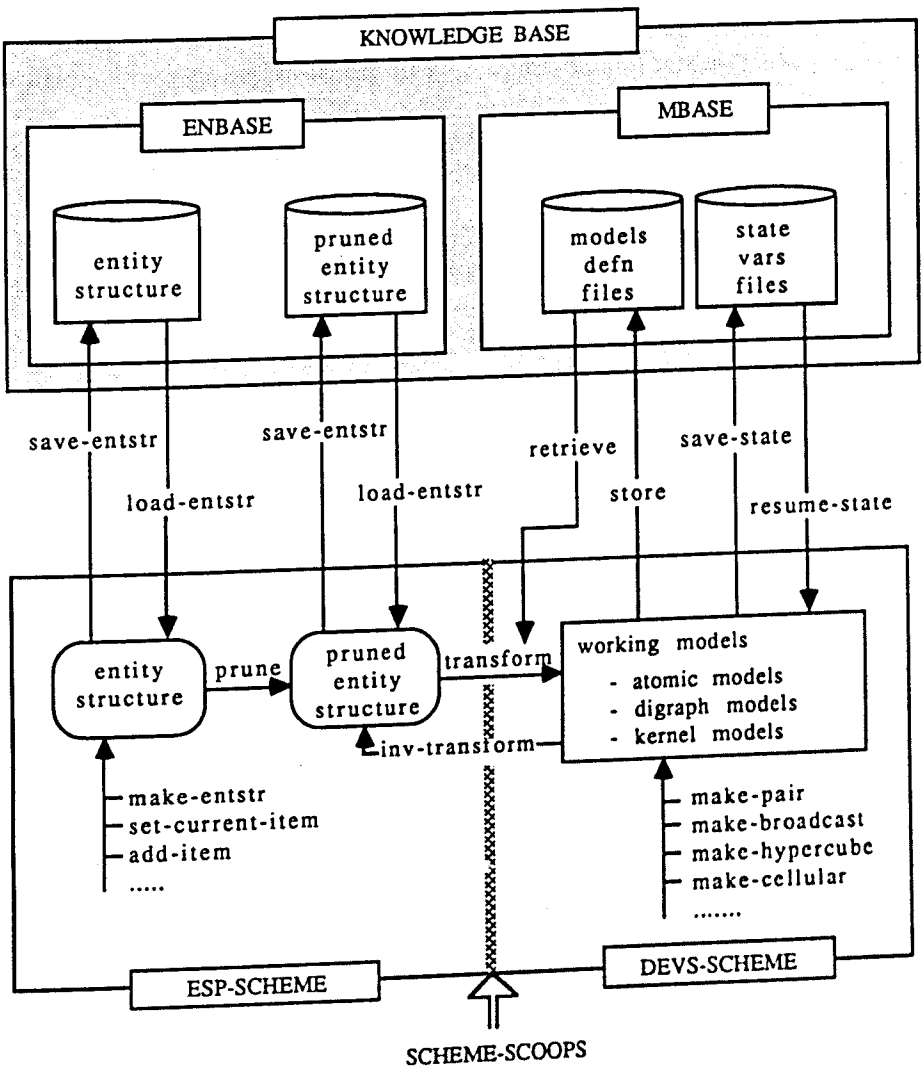


FIG. 4. *DEVS-SCHEME simulation environment.*

pruning an entity structure and selecting one possibility from the whole family spanned by the structure. The pruned entity structure is transformed into a hierarchical simulation model by the operation transform described in the previous section.

DESIGN PHASES OF KBDSE

KBDSE applies modelling and simulation concepts to unify engineering design activities and to develop a methodology for systematic design model construction and evaluation.

Design models are derived by identifying multiple conflicting objectives and requirements of systems. Therefore, design objectives play a fundamental role in guiding the synthesis of design models and the specification of experimental circumstances.

Evaluation of design alternatives is accomplished by computer simulation. The experimental frame concept[9] is used to specify a simulation study. Briefly, an experimental frame defines conditions with which a design model can be observed and experimented with. Simulation results are compared and traded off in preference to conflicting criteria. This results in a ranking of models and supports choices of alternatives that best satisfy the design specification.

Evaluation of design alternatives involves the following stages:

(i) Selecting the problem domain by retrieving the desired entity structure (FRASES).
(ii) Identifying system requirements (e.g. cost, performance, technology, resources, etc.) from the design specification.
(iii) Performing rule-based design reasoning to derive all possible alternative design models (composition trees).
(iv) Specifying simulation circumstances for arrival process, event format and simulation controls.
(v) Constructing experimental frames conforming to design objectives and simulation requirements.
(vi) Coupling the design model with experimental frames for simulation (i.e. transformation).
(vii) Analysing performance statistics and selecting the best design model by the application of multi-criteria decision making methods.
(viii) Reporting the best design.

A schematic representation of this design process is outlined in Figure 5. With KBDSE, the complex design process is handled intelligently and efficiently to reduce the overall design cycle and cost.

Example

To help understand the whole process of the KBDSE design methodology, design of distributed systems (Figure 6) will be used as an example.

Assume the design specification of the distributed system has been defined as follows:

(*DSF* (*constraint* (*value* (>*thruput 0.098*) (<*cost 300*)))
(*objective* (*max* (*value thruput*)) (*min* (*value cost*)))
(*criteria-weighting* (*value* (*rank thruput cost*)))
;; criteria preference: thruput > cost.

After the design specification is defined, the design pruning program is selected and activated to derive all possible alternative design models. For example, if MODSYN is employed, the design reasoning is performed in a backward-chaining manner. At each decision point, the user is asked questions to provide information for selecting design alternatives. For example, to determine the *MTS-technology*, the question about 'the degree of interaction among computer modules' will be asked. If the user indicates that the interaction among computer modules is low and resource sharing capability is desired, then the *local area network* (*LAN*) will be selected for *MTS-technology*. This design reasoning process will continue until all specialization nodes are traversed. Let us assume the following selections have been made:

medium-access-protocol: *CSMA/CD*
medium: *optical-fibre*
topology: *bus, ring*
access: *direct-access, cache*

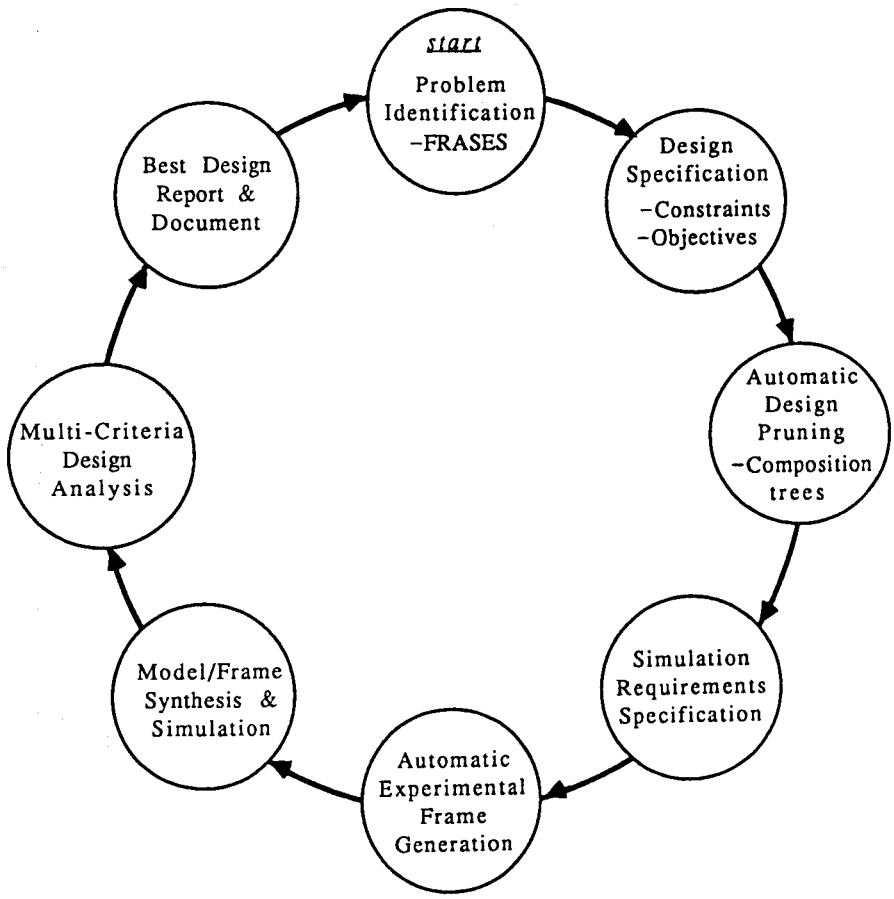


FIG. 5. *Design phases of KBDSE.*

The pruned FRASES (Figure 7) is then converted into two composition trees (Figure 8). Notice that two further composition trees are eliminated by detecting the synthesis rule:

IF LAN-segment.medium = optical-fibre then LAN.topology ≠ bus.

After the composition trees are generated, users may define the simulation requirements with ESF as follows:

```
(ESF (ap (value (cond (t (normal 20)))))
     ;; normal distribution with a mean 20
     (ef (value (cond (t (list (symbol) (number 1.0))))))
     ;; event format: (Job-1 0.72)
     (sc (value (cond ((>event 100) (stop)))))
     ;; stop simulation after 100 events
```

Experimental frame is then generated automatically[17] and coupled to the design model for performance evaluation. After simulation, the value of transducer (i.e. thruput) is collected for the best design selection. Assume the design cost and throughput for both design models are:

	thruput	cost
system-1	0.13	180
system-2	0.15	250

After rating parameters and assigning negative signs to the second set of parameters (i.e. minimizing the cost), the multi-criteria decision making (MCDM) model is constructed as follows:

	thruput	cost
system-1	0.867	−0.72
system-2	1.0	−1.0

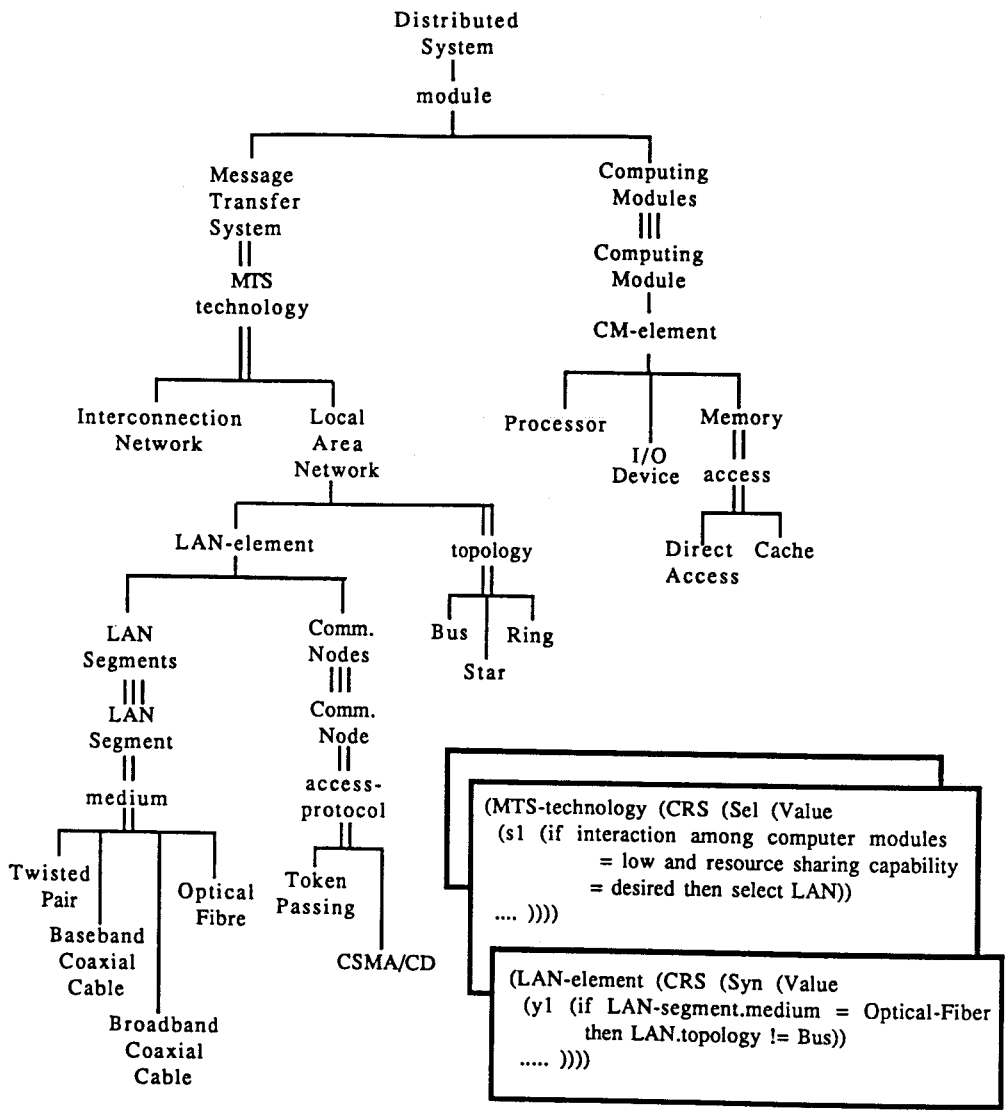


FIG. 6. *Distributed systems in FRASES.*

Since the criteria preference is expressed by weak ranking,[31] the extreme expected pay-off method can be employed to solve the MCDM problem. The partial average for each system is computed as follows:

	thruput	cost
system-1	0.867	0.0735
system-2	1.0	0.0

Finally, system-2 will be recommended (i.e. $1.0 > 0.941$).

CONCLUSIONS

We have presented a foundation and implementation of the knowledge-based design and simulation environment called KBDSE. We have employed the multifaceted modelling methodology as a theoretical basis for developing the KBDSE. To realize the multifaceted modelling methodology,

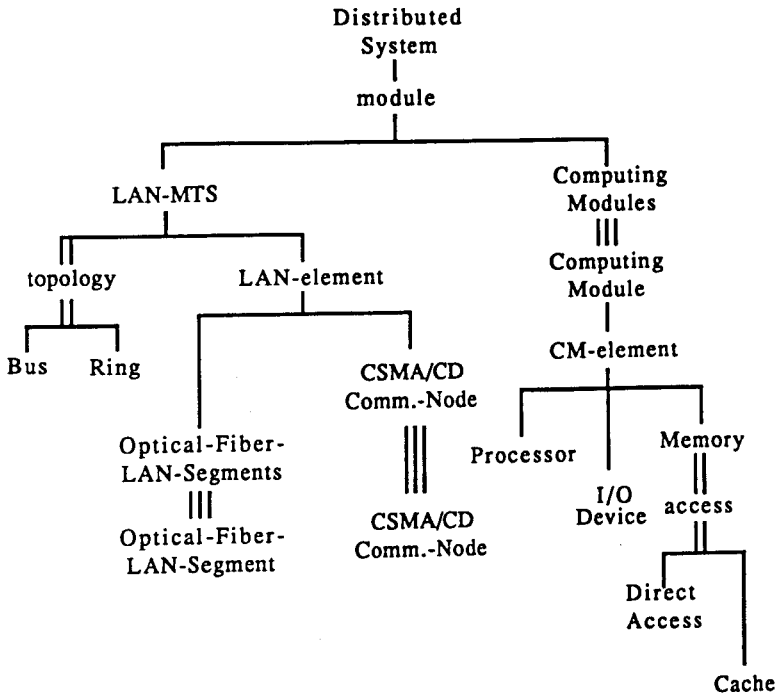


FIG. 7. *A pruned FRASES for distributed systems.*

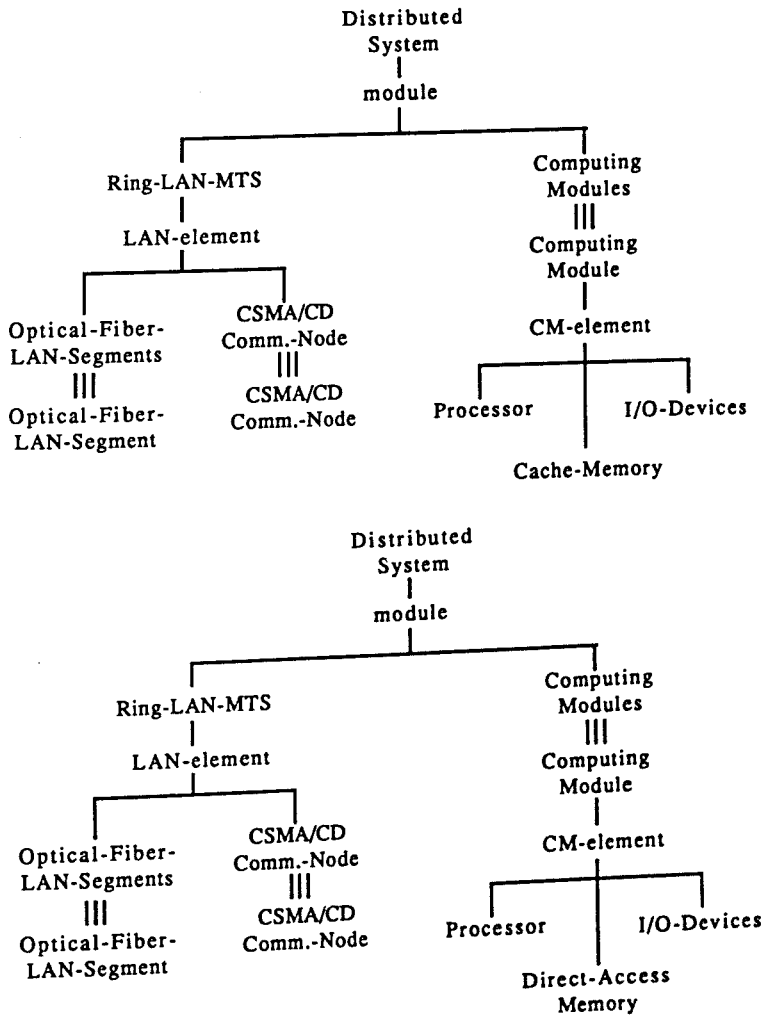


FIG. 8. *Composition trees for distributed systems.*

DEVS formalism and system entity structuring formalism have been implemented in a LISP environment. Such an implementation opens up a wealth of possibilities for investigating methodology-based support of modelling and simulation. The symbolic manipulation and object-oriented facilities of SCHEME make it relatively easy to code complex structures and their associated operations. The environment supports the development of discrete-event simulation models in a hierarchical, modular fashion. Many design examples of discrete-event simulation models—such as multi-level computer architectures, communication networks and multi-robotic systems—have been successfully run and tested in the environment.

REFERENCES

1. J. W. ROZENBLIT and B. P. ZEIGLER (1985) Concepts for knowledge-based system design environments. In *Proceedings of the 1985 Winter Simulation Conference*, San Francisco, California.
2. O. BALCI and R. E. NANCE (1987) Simulation support: prototyping the automation-based paradigm. In *Proceedings of the 1987 Winter Simulation Conference*, Atlanta, Georgia, December 1987, 495–502.
3. D. W. BALMER (1987) Modelling styles and their support in the CASM environment. In *Proceedings of the 1987 Winter Simulation Conference*, Atlanta, Georgia, December 1987, 478–485.
4. D. W. BALMER and P. J. PAUL (1986) CASM—the right environment for simulation. *J. Opl Res. Soc.* **37**, 443–452.
5. J. O. HENRIKSEN (1983) The integrated simulation environment: simulation software of the 1990s. *Opns Res.* **31**, 1053–1073.
6. JHYFANG HU, Y. HUANG and J. W. ROZENBLIT (1989) FRASES—A knowledge representation scheme for engineering design. In *Advances in AI and Simulation (SCS simulation series)* **20(4)**, 141–146.
7. JHYFANG HU (1989) Knowledge-based design support environment for design automation and performance evaluation. PhD Thesis, University of Arizona, Tucson, Arizona.
8. Y. M. HUANG (1987) Building an expert system shell for model synthesis in logic programming. MS Thesis, University of Arizona, Tucson, Arizona.
9. TAG GON KIM and B. P. ZEIGLER (1987) The DEVS formalism: hierarchical, modular system specification in an object oriented framework. In *Proceedings of the 1987 Winter Simulation Conference*, Atlanta, Georgia, December 1987, 559–566.
10. TAG GON KIM (1988) A knowledge-based environment for hierarchical modelling and simulation. Technical Report AIS-7 (PhD Thesis), University of Arizona, Tucson, Arizona.
11. TAG GON KIM and B. P. ZEIGLER (1989) The DEVS-SCHEME simulation and modelling environment. In *Knowledge Based Simulation: Methodology and Application* (PAUL A. FISHWICK and RICHARD B. MODJESKI, Eds), Springer Verlag Inc., New York.
12. M. MINSKY (1975) A framework for representing knowledge. In *The Psychology of Computer Vision* (P. H. WINSTON, Ed.), 211–277, McGraw-Hill, New York.
13. K. J. MURRAY and S. V. SHEPPARD (1987) Automatic model synthesis using automatic programming and expert systems techniques toward simulation modeling. In *Proceedings of the 1987 Winter Simulation Conference*, Atlanta, Georgia, December 1987, 534–543.
14. A. NEWELL and H. A. SIMON (1972) *Human Problem Solving*, Prentice-Hall, Englewood Cliffs, New Jersey.
15. N. PAN (1989) A LISP-based shell for model structure generation in knowledge-based system design. MS Thesis, University of Arizona, Tucson, Arizona.
16. J. W. ROZENBLIT (1985) A conceptual basis for integrated model-based system design. PhD Thesis, Wayne State University, Detroit, Michigan.
17. J. W. ROZENBLIT and Y. HUANG (1987) Constraint-driven generation of model structures. In *Proceedings of the 1987 Winter Simulation Conference*, Atlanta, Georgia, December 1987, 604–611.
18. J. W. ROZENBLIT and B. P. ZEIGLER (1988) Design and modelling concepts. In *Encyclopedia of Robotics*, pp. 308–322. John Wiley, New York.
19. J. W. ROZENBLIT and B. P. ZEIGLER (1986) Entity-based structures for model and experimental frame construction. In *Modelling and Simulation in Artificial Intelligence Era* (M. S. ELZAS *et al.*, Eds), North-Holland, Amsterdam.
20. J. W. ROZENBLIT and Y. M. HUANG (1989) Rule-based generation of model structures in multifacetted modelling and system design. ORSA *J. Computing* (submitted).
21. J. W. ROZENBLIT and JHYFANG HU (1989) Experimental frame generation in a knowledge-based system design and simulation environment. In *Modelling and Simulation Methodology: Knowledge System Paradigms* (M. S. ELZAS, T. I. OREN and B. P. ZEIGLER, Eds), pp. 451–466. North-Holland, Amsterdam.
22. J. W. ROZENBLIT, J. HU and Y. HUANG (1989) An integrated, entity-based knowledge representation scheme for system design. In *Proceedings of the 1989 NSF Engineering Design Research Conf. Amherst, MA*, 393–408.
23. SULEYMAN SEVINC (1988) Automatic simplification of models in a hierarchical modular discrete event simulation environment. PhD Thesis, University of Arizona, Tucson, Arizona.
24. B. P. ZEIGLER (1984a) *Multifaceted Modelling and Discrete Event Simulation*, Academic Press, London.
25. B. P. ZEIGLER (1984b) System-theoretic representation of simulation models. *IIE Trans.* **16**, 10–27.
26. B. P. ZEIGLER (1986) DEVS-SCHEME: a LISP-based environment for hierarchical, modular discrete event models. Technical Report AIS-2 CERL Laboratory, University of Arizona, Tucson, Arizona.
27. B. P. ZEIGLER (1987) Hierarchical, modular discrete-event models in an object-oriented environment. *Simulation*, **50**, 219–230.
28. B. P. ZEIGLER, TAG GON KIM, S. SEVINC and G. ZHANG (1989) Implementing methodology-based tools in DEVS-SCHEME. In *Modelling and Simulation Methodology: Knowledge System Paradigms* (M. S. ELZAS, T. I. OREN and B. P. ZEIGLER, Eds), North-Holland, Amsterdam.

29. B. P. ZEIGLER (1990) *Object-Oriented Simulation with Hierarchical Modular Models: Intelligent Agents and Endomorphic Systems*. Academic Press, Boston.
30. A. I. CONCEPCION and B. P. ZEIGLER (1988) DEVS formalism: A framework for hierarchical model development. *IEEE Trans. on Software Engineering*, **14**, 228–241.
31. A. OSYCZKA (1984) *Multicriterion Optimization in Engineering*. Ellis Horwood Press Ltd, Chichester.