Knowledge-based environment for investigating multicomputer architectures

T G Kim and B P Zeigler*

Multicomputers for massively parallel processing will eventually employ billions of processing elements, each of which will be capable of communicating with every other processing element. A knowledge-based modelling and simulation environment (KBMSE) for investigating such multicomputer architecture at a discrete-event system level is described. The KBMSE implements the discrete-event system specification (DEVS) formalism in an object-oriented programming system of Scheme (a LISP dialect), which supports building models in a hierarchical, modular manner, a systems-oriented approach not possible in conventional simulation languages. The paper presents a framework for knowledge-based modelling and simulation by exemplifying modelling a hypercube multicomputer architecture in the KBMSE. The KBMSE has been tested on a variety of domains characterized by complex, hierarchical structures such as advanced multicomputer architectures, local area computer networks, intelligent multi-robot organizations, and biologically based life-support systems.

multicomputers, parallel processing, knowledge-based environment, simulation modelling, architectures, object-oriented programming

Multicomputers for massively parallel processing will eventually employ billions of processing elements, each of which will be capable of communicating with every other processing element[1,2]. Recently, demands for such multicomputers have increased rapidly in many scientific/engineering applications, including artificial intelligence and expert systems. In spite of the demands, design of multicomputer architectures that support such large numbers of processors remains in its infancy[3]. The process for building a class of possible architectures with such complex design is extremely slow and expensive. Simulation modelling offers efficient ways of investigating the enormous number of alternatives for multicomputer architectures, thereby saving effort, time, and money.

A high-level multicomputer architecture can be viewed as a discrete-event system. At such a level of investigation, conventional discrete-event simulation languages, such as SIMSCRIPT II.5, can be used. However, such languages have limitations of expressive powers inherited from their underlying implementation language. Moreover, none of these languages allows a modeller to keep models in an organized library in modular form, enabling hierarchical assembly and disassembly as required in investigating design alternatives. Limitations of such languages in this regard have been illustrated[4].

New simulation environments have to be developed for rapidly prototyping simulation models for the proposed multicomputer systems. Such environments must overcome the limitations above, and therefore reduce design and testing time significantly. The environments must evolve a distinct paradigm for decomposing problems into object classes and associated behaviours and, therefore, offer special support for object-oriented programming style.

This paper describes one such new environment, a knowledge-based modelling and simulation environment (KBMSE), for modular, hierarchical construction of discrete-event simulation models of multicomputer architectures. In contrast to other knowledge-based simulation systems[5], the KBMSE is based on two formalisms: discrete event-system specification (DEVS) formalism and system entity structure (SES) formalism. The KBMSE allows a modeller to specify explicitly the structure of a simulation model using SES formalism and its behaviour using DEVS formalism. Such structural and behavioural specification of a model can be saved in a structural knowledge base called entity structure base and a behavioural knowledge base called model base, respectively. The KBMSE automatically synthesizes hierarchical, modular models from the model base resident components under the direction of structural knowledge in the entity structure base. This capability facilitates flexible combination and amendment of the structure and/or behaviour of multicomputer architectures, which makes it possible to investigate efficiently a large class of alternatives generated from the model base.

This paper first presents the architecture of the KBMSE and then briefly describes implementations of the two formalisms employed to develop the environment. It outlines a knowledge-based framework for mo-

Department of Electrical and Computer Engineering, University of Kansas, Lawrence, KS 66045, USA.
*Department of Electrical and Computer Engineering, University of Arizona, Tuscon, AZ 85721, USA

 0950-5849/89/100512-09 $03.00

delling and simulation within the KBMSE. To show the power of the KBMSE, a simulation model for a hypercube multicomputer architecture is developed. Conclusions of the paper introduce applications of the KBMSE to some ongoing research projects on artificial intelligence and distributed simulation.

KBMSE ARCHITECTURE

The KBMSE was developed by incorporating three areas[6]: artificial intelligence, system theory, and modelling formalism. Figure 1 shows the architecture of the KBMSE. Actually, the KBMSE implements the DEVS formalism and SES formalism in the object-oriented programming system in Scheme (a LISP dialect), which are called DEVS-Scheme and ESP-Scheme, respectively.

DEVS-Scheme creates the model base (Mbase), which contains models' behavioural knowledge; ESP-Scheme develops the entity structure base (Enbase), which contains models' structural knowledge. The behavioural knowledge base, Mbase, contains the DEVS definitions for models, which are either atomic or coupled. The models in the Mbase can be retrieved using retrieval rules[6] and transformed into components, ultimately being synthesized into a simulation model. The state variables of models and their values can also be saved in the Mbase so that they can be copied into those of isomorphic models as needed (detail of model isomorphism is available elsewhere[6, 7]).

The structural knowledge base, Enbase, comprises both entity structures and pruned entity structures (prune will be explained later), which represent structural knowledge of a simulation model. A facility save-entstr in the ESP-Scheme saves entity structures/pruned entity structures in the current environment into the Enbase. Inversely, both entity structures and pruned entity structures in the Enbase can be loaded into the current environment by a facility load-entstr in ESP-Scheme. The entity structure manager in ESP-Scheme maintains lists of entity structures and pruned entity structures.

The KBMSE then automatically synthesizes a hierarchical, modular simulation model by searching and retrieving component models that reside in the Mbase under direction of structural knowledge for the model maintained in the Enbase.

DEVS formalism and DEVS-Scheme

The DEVS (discrete-event system specification) formalism developed by Zeigler is a theoretical, well grounded means of expressing hierarchical, modular discrete-event models[8–10]. Within the formalism, it must be specified:

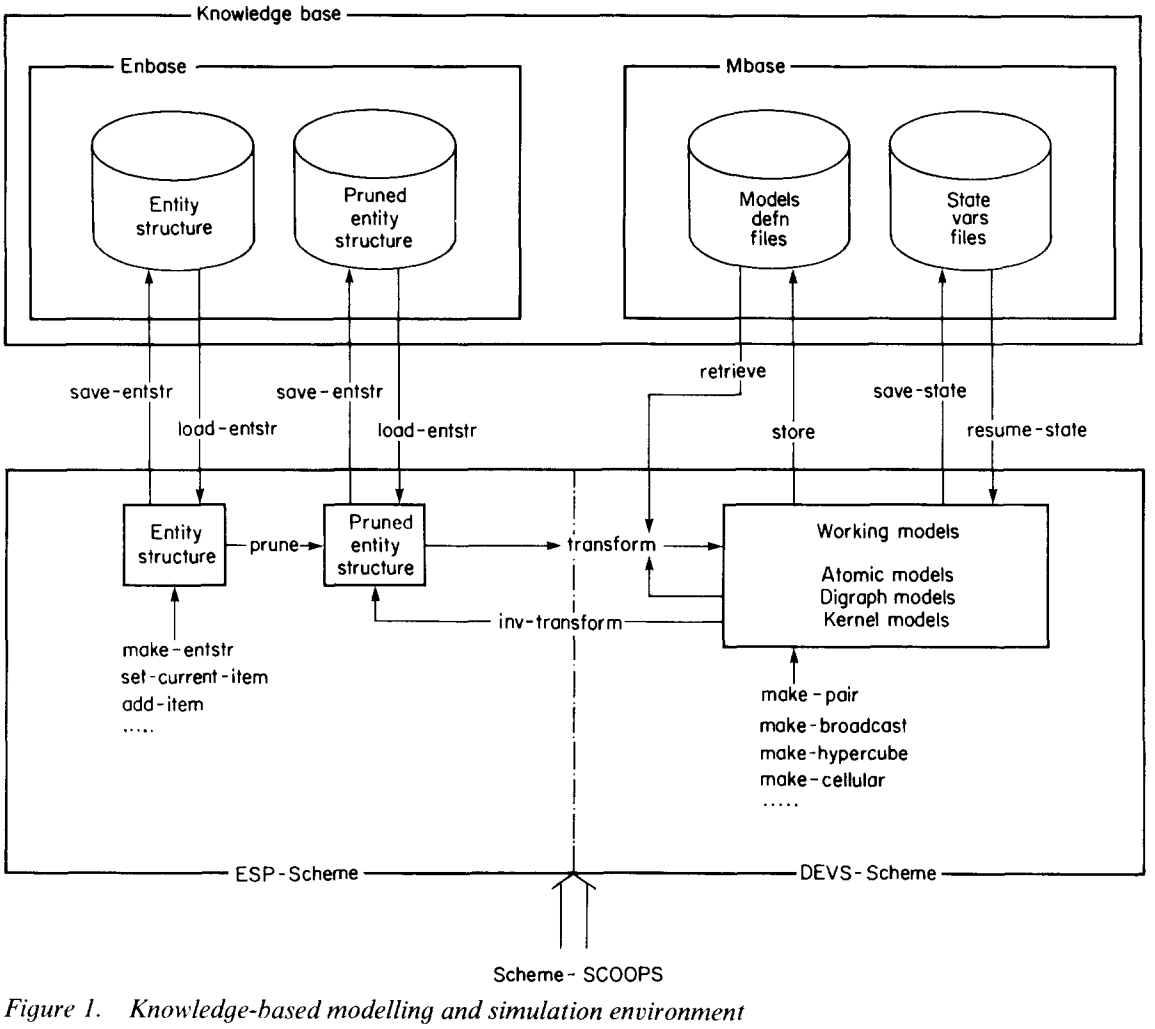


Figure 1. Knowledge-based modelling and simulation environment

- the basic models from which larger ones are built
- how these models are connected together in hierarchical fashion

Definitions for a basic model, called an atomic DEVS, and a general model, called a coupled DEVS, are given in Zeigler[10].

DEVS-Scheme, a general-purpose modelling and simulation environment, is an implementation of DEVS formalism in a LISP-based, object-oriented programming system[7,11–13]. It runs on DOS-compatible PCs and for the TI's Explorer LISP machine. DEVS-Scheme is implemented as a shell that sits on 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. As 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[14–16]. DEVS-Scheme also serves as a medium for developing computer architectures for distributed simulation of hierarchical, modular discrete-event models[17].

The class specialization hierarchy in DEVS-Scheme is shown in Figure 2. All classes in DEVS-Scheme are subclasses of the universal class 'entities', which provides 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 and for all.

'Models' and 'processors', the main subclasses of entities, provide the basic constructs needed for modelling and simulation. Models is further specialized into the major classes 'atomic-models' and 'coupled-models', which in turn are specialized into more specific cases, a process that 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, has three specializations: 'simulators', 'co-ordinators', and 'root-co-ordinators'. These specializations carry out the simulation of a model in a manner following the hierarchical abstract simulator concepts[10]. Due to its object-oriented realization, new classes, as required in new applications, can be readily added to DEVS-Scheme. As a result the DEVS-Scheme environment

- supports modular, hierarchical model construction
- allows independent testing of components models
- separates models from experimental frames
- supports distributed simulation

Details of all classes in DEVS-Scheme along with their instant/class variables and methods are given elsewhere[6]. The reader is also referred to the Scheme literature[18] for background.

SYSTEM ENTITY STRUCTURE AND ESP-SCHEME

A system entity structure (SES) is a knowledge-representation scheme that contains the decomposition, coupling, and taxonomy information for a system necessary to direct model synthesis[19,20]. Formally, the SES is a

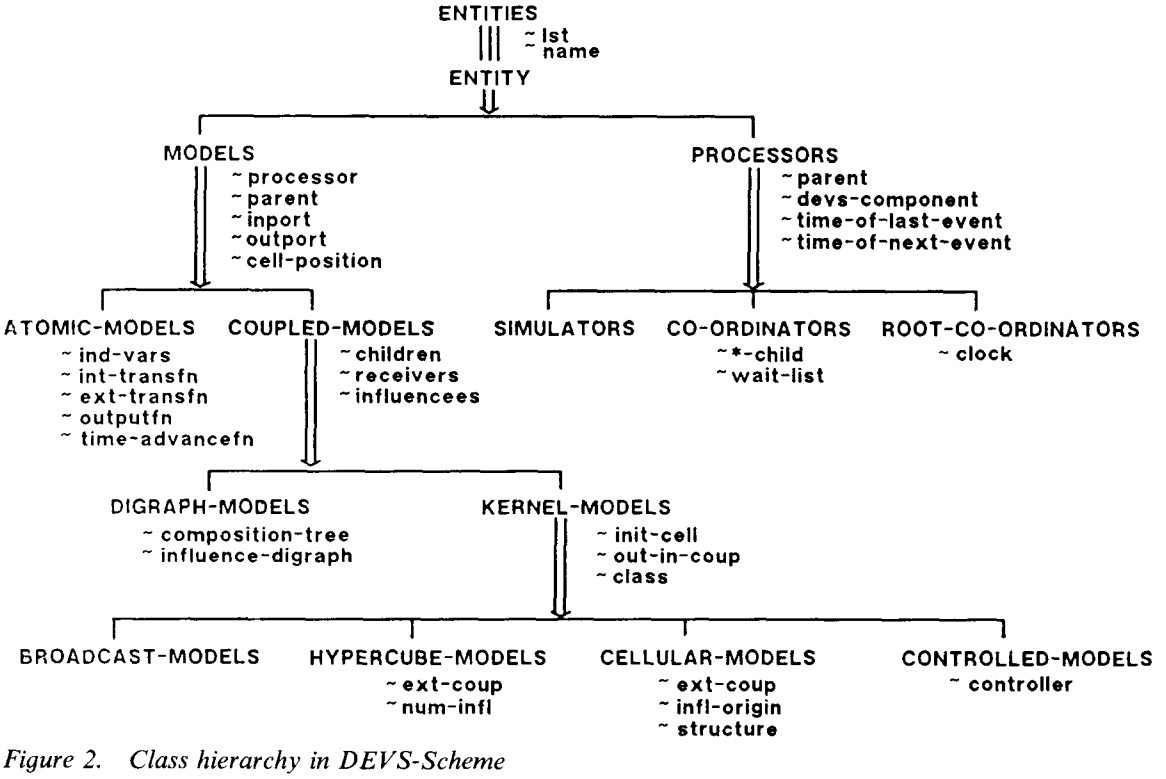


Figure 2. Class hierarchy in DEVS-Scheme

labelled tree with attached variable types that satisfies five axioms—alternating mode, uniformity, strict hierarchy, valid brothers, and attached variables. Detailed description of the axioms has been given[10].

There are three types of nodes in SES—entity, aspect, and specialization—which represent three types of knowledge about the structure of systems. The entity node, having several aspects and/or specializations, corresponds to a model component that represents a real-world object. The aspect node (a single vertical line in the labelled tree of Figure 3) represents one decomposition, out of many possible, of an entity. Thus the children of an aspect node are entities, distinct components of the decomposition. The specialization node (double vertical arrows in the labelled tree of Figure 3) represents a way in which a general entity can be categorized into special entities. As shown in Figure 3, attached to an aspect node is a coupling scheme, and to a specialization node a selection constraint. The coupling scheme specifies external input, external output, and internal couplings of a system and its components; the selection constraint designates rules to select a specialized entity from a generalized one in the pruning process. The coupling scheme is necessary to carry out the hierarchical synthesis of a simulation model.

A multiple entity is a special entity that consists of a collection of homogeneous components. Such components are called a multiple decomposition of the multiple entity. The aspect of such a mulitple entity is called multiple aspect (triple vertical lines in the labelled tree of Figure 3). The representation of such a multiple entity is as follows. A mulitple entity BS and its components Bs are represented by BS, three vertical lines, and B from the top down. Note that instead of presenting all Bs for BS's components, only one B is placed in the labelled tree. The number of Bs is specified by a variable, which is attached to the multiple aspect node.

Representation of SES module and its basic operations can be found elsewhere[6,21]. The operation 'prune' extracts a substructure of the SES by selecting one aspect and/or one specialization for each entity in the SES. It ultimately reduces the SES to a composition tree that contains all the information about the structure of a model. The operation 'transform' synthesizes a model in a hierarchical fashion from components in the model base developed by using DEVS-Scheme.

ESP-Scheme is a realisation of SES in Scheme LISP environment. Complete description of ESP-Scheme is beyond the scope of this paper. Details of ESP-Scheme, including hierarchical model structuring operations, reusability of structures, facilities, and others, are available elsewhere[21].

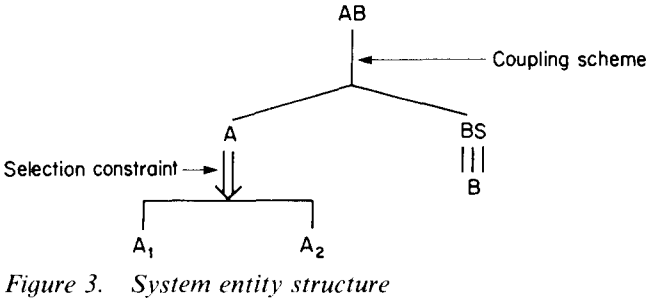


Figure 3. System entity structure

MODELLING AND SIMULATION METHODOLOGY

The KBMSE offers two methods for modelling and simulation of discrete-event systems. The first method makes use of only half the knowledge base, the Mbase, supported by the DEVS-Scheme. In this method, a hierarchical model is developed from bottom up. All the coupling information required to synthesize a hierarchical model must be provided explicitly using methods in digraph-models and kernel-models. The coupled model must be saved into, and retrieved from, the Mbase. Therefore, the models in the Mbase so saved are less reusable than those saved using both the Enbase and Mbase.

The second method takes advantage of both knowledge bases in the KBMSE. In this method, the ESP-Scheme underlies the DEVS-Scheme in the sense that the ESP-Scheme organizes models in the Mbase specified by the DEVS-Scheme. In the environment, modelling activity consists of three subactivities: specification of model structure, specification of model behaviour, and synthesis of a simulation model.

The structure specification and/or behaviour specification may exist in the knowledge base. If the structure specification is there, it will be in the Enbase as an entity structure. Likewise, if the behaviour specification exists, it will be in the Mbase as DEVS models. If the structural knowledge is not in the Enbase, a modeller needs to specify it by building an entity structure for it using ESP-Scheme. The entity structure characterizes a family of possible models. To construct one of them, the modeller needs to prune the entity structure. If the behavioural knowledge is not in the Mbase, it should be specified using DEVS-Scheme. Only atomic models need be constructed, although coupled models may be added to the Mbase if desired.

If both structure and behaviour of a model and an experimental frame are specified, the structure is synthesized into a simulation model. After simulating the model, the modeller may need to change the structure/behaviour of the model and simulate it again. Once he finishes his simulation, he may save the structure/behaviour knowledge of the model in the Enbase/Mbase for later use. Figure 4 summarizes the modelling and simulation methodology using the KBMSE of Figure 1.

HYPERCUBE ARCHITECTURE MODELLING: EXAMPLE

DEVS-Scheme facilitates specification of architectures that employ compound clusters. More specifically, the kernel models — broadcast, hypercube, or cellular — are particularly suited to model the recently introduced

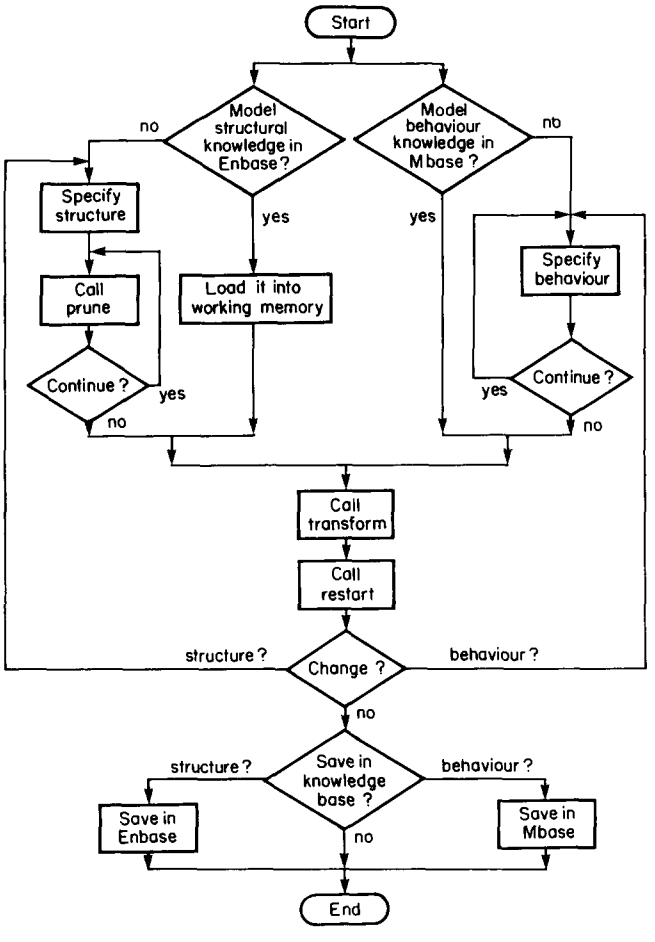


Figure 4. Modelling and simulation methodology

computer architectures for massively parallel processing employing such clusters[22]. Modelling and simulation of a hypercube architecture (see Figure 6) are described in the KBMSE. Computer models placed at the nodes of the hypercube (hc-NODES) are themselves kernel models, either broadcast or cellular. Each member of the broadcast or cellular model at a node is a processing element that consists of an active buffer and a processor.

When the hypercube model receives an incoming problem from outside, it sends the problem to all nodes (if its external coupling is 'broadcast') or the origin node only (if its external coupling is 'origin-only'). If it receives a solution of the incoming problem from its nodes, the hypercube model sends the solution to the outside. When a node of the hypercube receives a problem, it sends it to all processing elements in the node (if the node has a broadcast architecture) or to the specified processing element(s) (if the node has a cellular architecture).

The processor in the processing element is responsible for actual problem solving. The active buffer in the processing element accepts incoming problems for solution or retransmission. The active buffer accepts the incoming problem under certain conditions and sends it to the cascaded processor if the processor is free. If the conditions for acceptance are not met, the active buffer transfers the incoming problem to the output port unsolved, which takes a certain amount of time. The unsolved problem is retransmitted to other processing elements (the destination is based on the architecture of nodes) in the same node where the active buffer is.

If the active buffer receives an incoming problem while transmitting a problem to the 'unsolved' port, it allows itself to be interrupted and transfers the problem to the output port 'interrupted'. The unsolved problem transmitted through the interrupted port is retransmitted to the closest neighbourhood node(s) in the hypercube. If the active buffer receives an incoming problem while sending an unsolved problem to the port interrupted, however, the problem is lost.

The problem in designing the above parallel processing system, to be investigated with simulation, is how to take advantage of the speed in searching for free active buffers and processors afforded by the hypercube with nodes broadcasting/cellular coupled, while minimizing the overhead of problems solved more than once as well as not at all.

To experiment with the model of the parallel processing system, an appropriate experimental frame needs to be specified and coupled to the model. The experimental frame module E is a digraph model that contains a generator G for sending problems to the hypercube model, an acceptor A that makes decisions about run termination, and a transducer T, which keeps the performance statistics. These statistics include problems solution time, throughput, percentage of multiply solved problems, and percentage of lost problems.

The overall simulation model is a digraph model of DEVS-Scheme that consists of the experimental frame and hypercube model. Based on modelling and simulation methodology described in the previous section, first the architecture of the overall simulation model is specified. Figure 5(a) and (b) shows ESP-Scheme code and entity structure for the simulation model, respectively. Then it is necessary to specify the behaviour of atomic components such as generator, acceptor, transducer, active buffer, and processor using DEVS-Scheme. The behavioural specification includes definition of internal transition function, external transition function, and output function for such components. Figure 5(d) shows one such DEVS-Scheme specification for the atomic model processor.

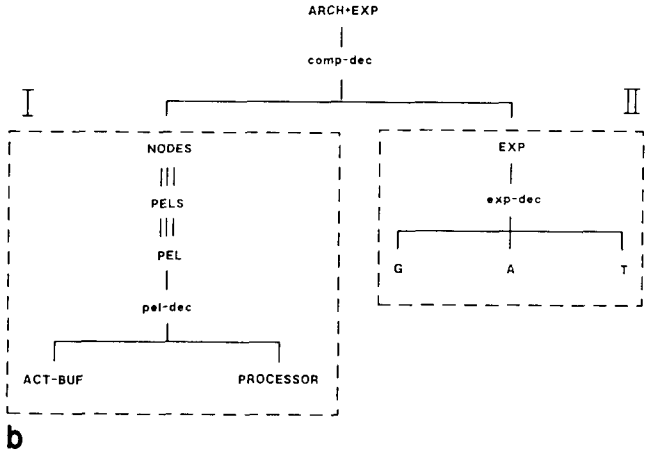
To simulate an alternative architecture of nodes in the hypercube, it is possible to select either broadcast or cellular coupling for its nodes in the pruning process. Figure 5(c) shows one such pruned entity structure in which the broadcast coupling is selected for processor elements in the nodes.

Figure 5(e) shows the knowledge base that contains both behavioural and structural knowledge of the model necessary for automatic synthesis of the simulation model. Using the operation 'transform', the KBMSE transforms the overall structure into a simulation model, which needs to be initialized before simulation. Such initialization includes creating new members of both hypercube and its nodes and setting up the coupling schemes of both hypercube and broadcast. Figure 6(a) and (b) shows the overall simulation model after such initialization.

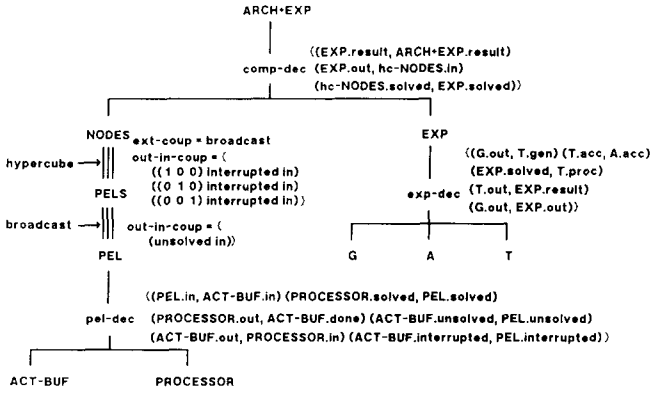
;;; ESP-Scheme Code for simulation model ;;;

```
;;; SES for multi-level architecture (region I of Figure 5 (b)) ;;;
(make-entstr 'nodes)
(add-mult-mult e:nodes 'node 'pel)
(set-current-item e:nodes 'pel)
     (add-item e:nodes 'aspect 'pel-dec)
     (set-current-item e:nodes 'pel-dec)
     (add-item e:nodes 'entity 'act-buf)
     (add-item e:nodes 'entity 'processor)

;;; SES for experimental frame (region II of Figure 5 (b)) ;;;
(make-entstr 'exp)
(add-item e:exp 'aspect 'exp-dec)
(set-current-item e:exp 'exp-dec)
     (add-item e:exp 'entity 'g)
     (add-item e:exp 'entity 'a)
     (add-item e:exp 'entity 't)

;;; SES combining architecture and experimental frame above ;;;
(make-entstr 'arch + exp)
(add-item e:arch + exp 'aspect 'comp-dec)
(set-current-item e:arch + exp 'comp-dec)
     (add-sub-entstr e:arch + exp e:nodes)
     (add-sub-entstr e:arch + exp e:exp)
```

a



b

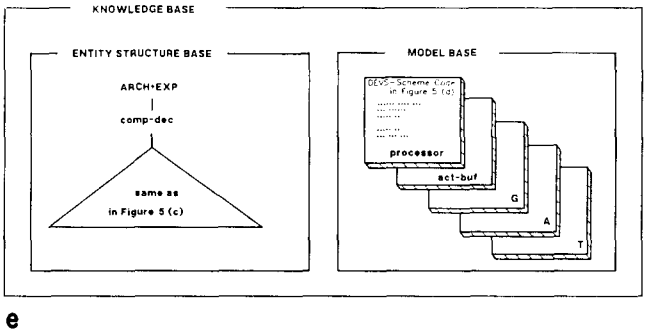


c

;;;; DEVS-Scheme Code for atomic-model processor ;;;;;

```
;;; make a pair of atomic model and simulator ;;;
(make-pair atomic-models 'processor)

;;; state s of type structure and its initialization ;;;
(send processor def-state '(packet-it))
(send processor set-s (make-state 'sigma 'inf 'phase 'passive 'packet-id 'unassigned))

;;; external transition function ;;;
(define (ext-p s e x)
    (let* ( (val (content-value x))
            (packet-id (car val))
            (processing-time (cadr val)))
        (case (content-port x)
            ('in
                (if (equal? 'passive (state-phase s))
                    (begin
                        (set! (state-packet-id s) packet-id)
                        (hold-in 'busy processing-time))
                    (continue)))
            (else (bkpt "error: invalid input port" (content-port x)))))
)

;;; internal transition function ;;;
(define (int-p s) (passivate))

;;; output function ;;;
(define (out-p s)
    (case (state-phase s)
        ('busy (make-content 'port 'out 'value (state-packet-id s)))
        (else (make-content)))
)

;;; assign the above definitions to processor ;;;
(send processor set-int-transfn int-p)
(send processor set-ext-transfn ext-p)
(send processor set-outputfn out-p)
```

d



e

Figure 5. (a) ESP-Scheme code; (b) entity structure for (a); (c) pruned entity structure of (b); (d) DEVS-Scheme code for atomic-model processor; (e) content of knowledge base for transformation

While the foregoing example illustrated a three-level architecture, DEVS-Scheme does not impose any limit on the depth of hierarchical structure, hence degree of compounding of clusters. In addition, the KBMSE allows the flexible investigation of a class of interconnection topologies for such clusters by selecting different coupling schemes in pruning process.

INVESTIGATION OF OPTICAL ARCHITECTURES FOR OBJECT-ORIENTED COMPUTING

Object-oriented programming, in which software is structured as a collection of interacting objects, has emerged as the common basis for a host of diverse

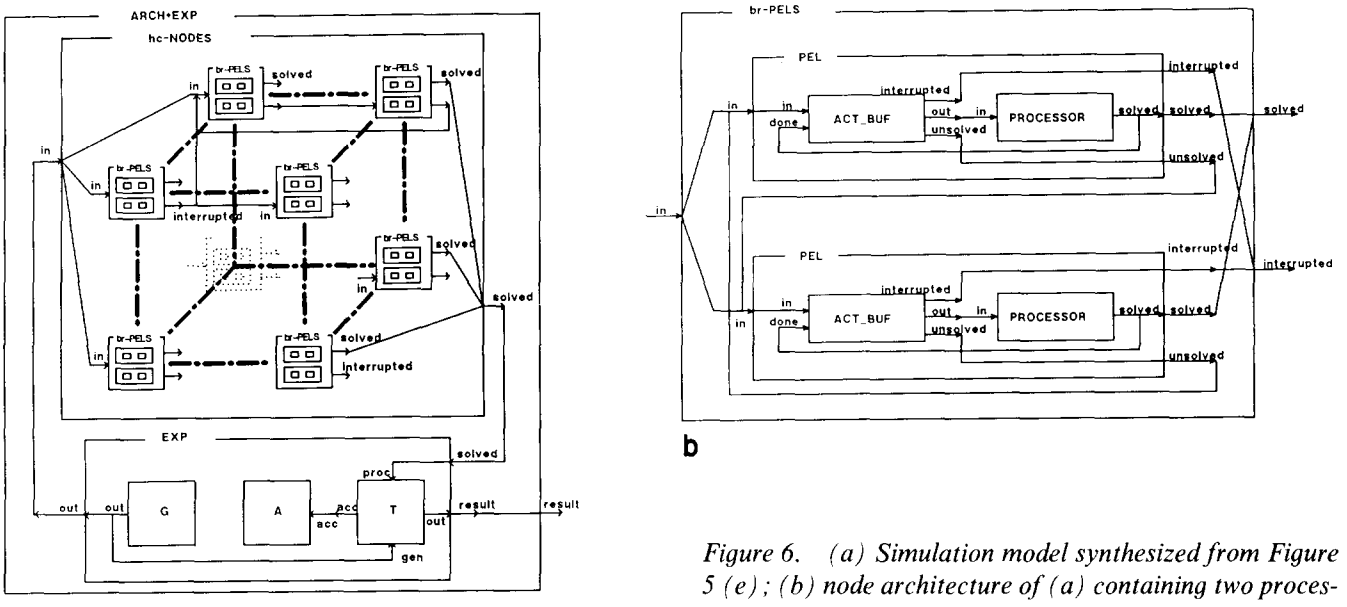


Figure 6. (a) Simulation model synthesized from Figure 5 (e); (b) node architecture of (a) containing two processor elements PEL

applications. Despite its division on many other aspects, the artificial intelligence community seems to be unifying around the object-oriented paradigm as a key component in its system development environments. Optics is, potentially, the ideal medium to exploit parallelism in AI object-oriented computation by breaking the bottlenecks that conventional electronics imposes. The parallel nature of optics and free-space propagation, together with their relative freedom from interference, makes them ideal for parallel computations. It is potentially feasible to build large optical interconnects that do not suffer from many of the limitations of their electronic counterparts. In addition, it is possible that optics can perform most of the basic operations required for object-oriented processing. In fact, optics may be a better alternative for some of the operations such as object matching, comparison, and searching, which are major bottlenecks for conventional electronics. These operations can be performed in parallel on two-dimensional planes of data at speeds that approach that of light (optical systems that can achieve these operations are optical correlators, matched filters, etc.). Moreover, optics may be able to provide a larger degree of connectivity between the processing elements than electronics. As an application of the KBMSE, the authors have begun to investigate a class of optical architecture models for AI object-oriented computing.

The architectures are structured as a cluster of optical planes, each plane forming the substrate for a set of processing elements that are realised with optical principles. In the sequel this architecture is referred to as the cluster-of-planes concept.

The cluster-of-planes concept uses local dataflow and global message-passing to exploit maximally available parallelism and to circumvent the purported lack of flexibility in optical systems in supporting MIMD-type (multiple instruction/multiple data) computations. A new technique called optical symbolic substitution will be used for the implementation as it appears to offer a high degree of parallelism with little overhead of communication, data addressing, and loop indexing. As location-addressible memory or random-access memory is not feasible (in the foreseeable future) in optics, the cluster-of-planes architecture uses inter-object message-passing to shift the burden from a memory-based to a register-based system. This transforms the problem from overcoming the centralized memory bottleneck to that of realising communications with a high degree of connectivity, solvable with optical interconnects. Such communication networks link objects on the same plane and on different planes.

To exploit the multi-dimensional nature of optical systems, objects are represented as rows of 2D planes of say 1000 × 1000 pixels of optical gates. Each object is allocated a set of rows of some plane and has slots (or registers) for its instance variables (structure) as well as optically interpretable code for its methods (behaviour). This object representation is efficient from an optics point of view, for it allows the entire plane to be reserved for computation (objects), while communications take place along the third dimension. Next, each class is assigned to a plane. This means that, first, there is a class (meta) object that contains the slots for class variables and the template for generating new instances of the class and, second, such instances are located in rows of the same plane. The template specifies the structure slots as well as the code. Objects as well as classes have unique optical tags by which they can be identified. Inter-object and inter-class communications are carried out through message-passing protocols via free-space optical interconnects.

Physically, each object is assigned to an optical processor (OP) on the processing plane. Each OP comprises three components: a processing element (PE), a com-

munication element (CE), and the communication network (CN). The PEs execute instructions and dataflow sequencing control. The CEs are responsible for implementing message-passing protocols and for transferring messages between communication PEs. The CNs represent the physical layer (buffers, registers, control mechanisms, etc.) that is needed for transferring messages between PEs. The separation of the OPs into PEs and CEs allows for more concurrent processing. By shifting the burden of message handling onto the CEs, communication and computation can proceed simultaneously. Thus a single processing plane is composed of several OPs. Each OP is assigned a unique optical tag (or address) with which it can be easily identified.

Several processing planes can be configured into a cluster-based architecture. These processing planes will communicate through programmable free-space interconnects. The global interconnects can be implemented using a realtime volume holographic medium such as photorefractive crystals. This hierarchy of interconnects (linear layout for communication within a processing plane and volume interconnects between planes) is intended to manage diffraction effects while providing a high degree of connectivity.

A family of three-level hierarchical models is being built to investigate key issues in the operation and performance of the cluster-of-planes architecture. Each optical plane is represented as a kernel-model in DEVS-Scheme similar to a node in Figure 6. The cluster of planes is represented by a kernel-model in like manner to the interconnection of nodes in the hypercube architecture of Figure 6. The broadcast-model specialization for the latter kernel-model offers a good start toward representing the high connectivity offered by the free-space interconnects in the cluster-of-planes architecture.

To investigate the various alternative architecture models, the authors will:

- identify a set of operations that are crucial to object-oriented concurrent computing and that maximally exploit available parallelism
- seek optical realisations of such operations within the context of the cluster-of-planes architectural framework
- assess the overall system performance of alternative realisations, in isolation and in combination, using the modelling and simulation tools of the KBMSE

The authors hope to draw conclusions on how well the architectural and computational requirements of symbolic AI object-oriented computing can be supported by optical techniques.

CONCLUSIONS

Simulation modelling is necessary to investigate the enormous space of alternatives for multicomputer architectures if the extremely slow process of building each architecture is to be obviated. Classes of relatively dense graphs and means of compounding them have been proposed for generating computer architectures for classes of problems that can exploit such structures[3]. DEVS-Scheme provides a powerful means for specifying such hierarchical compounded structures. As the structure and behaviour of the simulation model are readily manipulated in a knowledge representation, the modeller can freely change the structure and/or behaviour of the model and readily investigate a large class of alternatives generated from the model base.

The KBMSE supports the design and testing of complex discrete-event systems such as multicomputer architectures, computer-integrated manufacturing systems, intelligent multi-robotic systems, and others. The model construction, manipulation, and analysis features have all been implemented and tested in a variety of simulation domains. Here some applications of the KBMSE are introduced. Sevinc and Zeigler[23] describe a entity-structure-based design methodology for local area networks. Design of hierarchically constructed rule-processing computer architectures is an ongoing PhD project[24] (supported by NSF). A simulation system for intelligent multi-robot organizations for laboratory management in space is being designed[25] (supported by NASA). Interfacing the KBMSE to other existing simulation systems has been studied with successful results[26]. A hierarchical scheduling system in an AI-based control system has been designed using ESP-Scheme[27,28] (supported by the Biosphere II project). The KBMSE serves as a vehicle for research in knowledge-based system design using variant families of design models[19,29].

ACKNOWLEDGEMENTS

Research reported here was supported by US National Science Foundation under Grant CCR 8714148 for 'Intelligent simulation environment for advanced computer architectures'.

REFERENCES

1 **Falman, S E and Huntion, G E** 'Connectionist architectures for artificial intelligence' *IEEE Computer* Vol 20 No 1 (January 1987) pp 100–113
2 **Hillis, W D** *The connection machine* MIT Press, Cambrige, MA, USA (1986)
3 **Uhr, L** *Multi-computer architectures for artificial intelligence: toward fast, robust, parallel systems* John Wiley, New York, NY, USA (1987)
4 **Chen, T** 'Implementation of the model base concept in SIMSCRIPT II.5' *Master Thesis* University of Arizona, Tucson, AZ, USA (1987)
5 **Reddy, Y V, Fox, M S, Husain, N and McRoberts, M** 'The knowledge-based simulation system' *IEEE Software* Vol 3 No 2 (March 1986) pp 26–37
6 **Kim, T G** 'A knowledge-based environment for hierarchical modelling and simulation' *Technical report AIS-7 (PhD Thesis)* University of Arizona, Tucson, AZ, USA (May 1988)

7 **Kim T G and Zeigler, B P** 'The DEVS formalism: hierarchical, modular system specification in an object-oriented framework' in *Proc. 1987 Winter Computer Simulation Conf.* Atlanta, GA, USA (December 1987) pp 559–566

8 **Concepcion, A I and Zeigler, B P** 'DEVS formalism: a framework for hierarchical model development' *IEEE Trans. Soft. Eng.* Vol 14 No 2 (February 1988) pp 228–241

9 **Ziegler, B P** *Theory of modelling and simulation* John Wiley, NY USA (1976) (reissued by Krieger, Malabar, FL, USA (1985))

10 **Zeigler, B P** *Multifacetted modelling and discrete event simulation* Academic Press, Orlando, FL, USA (1984)

11 **Kim T G and Zeigler, B P** 'The DEVS-Scheme simulation and modelling environment' in **Fishwick, P A and Modjeski, R B (eds)** *Knowledge-based simulation: methodology and applications* Springer Verlag, New York, NY, USA (1989)

12 **Zeigler, B P** 'Hierarchical, modular discrete event modelling in an object-oriented environment' *Simulation* Vol 49 No 5 (November 1987) pp 219–230

13 **Zeigler, B P, Kim, T G, Sevinc, S and Zhang, G** 'Implementing methodology-based tools in DEVS-Scheme' in **Elzas, M S, Oren, T I and Ziegler, B P (eds)** *Modelling and simulation methodology: knowledge system paradigms* North Holland, Amsterdam, The Netherlands (1989)

14 **Elbert, J L and Salter, R M** 'Modelling neural networks in Scheme' *Simulation* Vol 46 No 5 (May 1986) pp 193–199

15 **Zeigler, B P and Reynolds, R** 'Towards a theory of adaptive computer architectures' in *Proc. 5th Int. Distributed Computer Systems* Denver, CO, USA (1985) pp 468–475

16 **Zeigler, B P and Kim, T G** 'Knowledge representation in variable structure modelling: an adaptive computer architecture example' *Trans. Society for Comput. Simulation* (accepted for publication)

17 **Concepcion, A I** 'Hierarchical computer architecture for distributed simulation' *IEEE Trans. Comput.* Vol 38 No 2 (February 1989) pp 311–319

18 **Abelson, H, Sussman, G J and Sussman, J** *Structure and interpretation of computer programs* MIT Press, Cambridge, MA, USA (1985)

19 **Rozenblit, J W and Zeigler, B P** 'Concepts of knowledge based system design environments' in *Proc. 1985 Winter Computer Simulation Conf.* San Francisco, CA, USA (December 1985)

20 **Zeigler, B P** 'Knowledge representation from Minsky to Newton and beyond' *Applied Artificial Intelligence* (1987) pp 87–107

21 **Kim, T G and Zeigler, B P** 'ESP-Scheme: a realization of system entity structure in a LISP environment' *Advances in AI and Simulation* (Simulation Series) Vol 20 No 4 (March 1989) pp 135–140

22 **Kim, T G and Zeigler, B P** 'The class kernel-models in DEVS-Scheme: a hypercube architecture example' *ACM Simuletter* Vol 19 No 2 (June 1988) pp 20–30

23 **Sevinc, S and Zeigler, B P** 'Entity structure based design methodology: a LAN protocol example' *IEEE Trans. Software Eng.* Vol 14 No 3 (March 1988) pp 375–383

24 **Lee, C G** 'Simulation-based investigation of hierarchical multilevel architectures' *PhD dissertation* (in progress) University of Arizona, Tucson, AZ, USA (1989)

25 **Zeigler, B P, Cellier, F E, and Rozenblit, J W** 'Design of a simulation environment for laboratory management by robot organizations' *Intelligent and Robotic Syst.* Vol 1 (1988) pp 299–309

26 **Kim, T G, Zhang, G and Zeigler, B P** 'Entity structure management of continuous simulation models' in *Proc. 1988 Summer Computer Simulation Conf.* Seattle, WA, USA (July 1988) pp 620–625

27 **Kim, T G, Mignon, G V and Zeigler, B P** 'Design of an AI-based control system for self-sufficient habitats' in *Proc. First Int. Conf. Engineering Applications of Artificial Intelligence* Tullahoma, TN, USA (June 1988) pp 1059–1065

28 **Kim, T G and Ziegler, B P** 'Hierarchical scheduling in an intelligent environmental control system' in *Proc. Second Int. Conf. Engineering Applications of Artificial Intelligence* Tullahoma, TN, USA (June 1989)

29 **Rozenblit, J W, Kim, T G and Zeigler, B P** 'Towards an implementation of a knowledge-based system design and simulation environment' in *Proc. 1988 Winter Computer Simulation Conf.* San Diego, CA, USA (December 1988)