JOURNAL OF PARALLEL AND DISTRIBUTED COMPUTING 18, 77–88 (1993)

A Simulation Environment for Intelligent Machine Architectures*

BERNARD P. ZEIGLER AND AHMED LOURI

AI-Simulation Group, Department of Electrical and Computer Engineering, University of Arizona, Tucson, Arizona 85721

A modelling and simulation environment for intelligent machine architectures (IMA) should be able to represent both the symbolic and the numeric processing required in IMAs at various levels of abstraction. It should make useful predictions about an analyzed system in a time that is short compared to the time needed to build and test the actual system. Also, it should support modular construction and reuse of IMA component models. DEVS-Scheme is a knowledge-based, object-oriented, simulation environment to support design of IMAs. We show how it satisfies the given requirements. An example of an IMA developed in the DEVS-Scheme environment is discussed to illustrate its concepts and operation. © 1993 Academic Press, Inc.

1. SIMULATION-BASED DESIGN OF INTELLIGENT MACHINE ARCHITECTURES

The application of computer technology is expanding from pure data processing to information and knowledge processing. Knowledge-based system applications are characterized by symbolic processing, nondeterministic computations, dynamic execution, high potential for parallel and distributed processing, knowledge management, and open systems. However, the fundamental physical limits of current technology have not been overcome for the more sophisticated computation-intensive problems, such as predictive modeling and forecasting, design automation, large scale simulation, and artificial intelligence (AI). The combination of technological and economic factors make parallel and distributed computing systems attractive and effective for a large variety of intelligent machine applications [1]. However, the design complexity of large parallel distributed systems is much greater than that for single centralized processor systems. A multiprocessor system's dynamic behavior is too complex to be evaluated analytically. Analytic models based on queueing theory necessarily omit aspects of target system behavior that may be crucial [2].

Discrete-event simulation offers an efficient means of investigating the enormous number of alternatives for existing or proposed complex technical systems. Unfortunately, traditional discrete-event simulation languages (e.g., SIMSCRIPT, SIMAN, SLAM, GPSS) are not well suited to the study of intelligent machine architectures (IMA). They were originally developed to study systems with nonhierarchical transaction processing structures such as job shops and assembly lines with simple control decision rules. Moreover, communication between processors is a major concern in IMAs and such communication protocols are not easily represented in any of the common transaction-, event-, or process-oriented languages. From a modeling perspective, models developed using such simulation languages are nonmodular, not easily reusable, and inflexible. Conventional simulation languages do little to assist the modeler in describing system structure. Development of modular, reusable models requires object-oriented concepts such as data encapsulation, class inheritance, message passing, and dynamic binding. Object-oriented concepts also assist the modeler in developing the system structure in a hierarchical manner [3].

Consider more closely the requirements that a modeling and simulation environment for intelligent machine architectures should satisfy:

- Expressive power: As we have suggested, the environment should be able to represent (a) both the symbolic and the numeric processing required in IMAs, (b) the complex communication interactions between processors, and (c) the hierarchical structure of more advanced architectural designs.
- Levels of abstraction: Design of IMAs should be done at various levels of abstraction. For example, in a top-down design approach, first the functional requirements are worked out, then come successive levels of more detailed implementation until the level of hardware primitives is reached. The description of the IMA at each level of abstraction constitutes a virtual machine which should be verified as a correct implementation of the virtual machine above it. Simulation should not only contribute to such verification, but also provide feedback to the designer concerning the potential performance characteristics of the ultimate system.
- Reusability: The environment should support the construction of modular models, i.e., models that can be

* Research supported by NASA–Ames Cooperative Agreement NCC 2-525, "A Simulation Environment for Laboratory Management by Robot Organization," and NSF Grant "A Simulation Environment for Intelligent Architectures."

0743-7315/93 $5.00

developed and tested in a stand-alone manner. It should be possible to couple such models together easily as components in larger models that are modular as well. Such models should also be archived in model repository where they can be reused as components in subsequent design projects. Such resuability fosters rapid prototyping where it takes a relatively short time to construct a model and get useful predictions about its behavior and performance. The time to achieve such results should be short compared to the actual time needed to build and test the physical system.

This article shows how the modeling and simulation environment DEVS-Scheme satisfies many of the above requirements. Discrete event simulation and AI knowledge representation schemes form a powerful combination, called knowledge-based simulation, for satisfying the expressive power and multiple levels of abstraction requirements of IMA design. DEVS-Scheme is a knowledge-based simulation environment for modeling and design that facilitates construction of families of models in a form easily reusable by retrieval from a model base [4]. In contrast to other knowledge-based simulation systems, DEVS-Scheme is based on the DEVS formalism, a system theoretically grounded means of expressing hierarchical, modular discrete event simulation models. DEVS-Scheme is constructed in a layered fashion so that all of the underlying Scheme-based and object-oriented programming language features are available to the user. This layering also lends itself to model base organization using a knowledge representation scheme, called the system entity structure.

The plan of this article is as follows. We review the issues involved in designing intelligent machine architectures with emphasis on the need to explore multilevel schemes using discrete event simulation. We then briefly review DEVS-Scheme and illustrate it with a recent simulation study of multilevel architectures for heuristic search. Having presented this background, we introduce a model for concurrent computing in the object-oriented design paradigm. We then show how DEVS-Scheme can realize such computing models using its hierarchical, modular formalism to specify discrete event behavior. The article concludes with a multilevel intelligent machine architecture that is being designed using the DEVS-Scheme environment. Since the architecture is intended to support simulation-based design, this constitutes a kind of bootstrapping process peculiar to the simulation field.

2. DESIGN ISSUES IN INTELLIGENT MACHINE ARCHITECTURES

Intelligent machine architectures are targeted for functional areas such as vision and image understanding, natural language processing, speech recognition, mathematical theorem proving, game playing, and expert systems. They process data in symbolic, as well as numerical, form and therefore their processing characteristics differ fundamentally from those of conventional numerical processors. The main differences lie in (1) knowledge representation and management: symbol manipulation operations are often used, such as symbol comparison, selection, sorting, matching, searching, pattern retrieval, and recognition; (2) nondeterministic computations: the sequence of symbol manipulation operations is often highly data dependent and less amenable to compile-time analysis than in numerical processing; (3) intensive and irregular memory access patterns; (4) large potential for parallel processing: several paths may be pursued in parallel to reach a solution or a goal.

Until now most IMA designs have been built around a single level, centralized style of processing. However, recent evidence suggests that multilevel, heterogeneous, distributed control architectures offer the most promising road to support intelligence/knowledge processing [5]. Machine vision is a case in point. For low level vision processing, highly parallel SIMD (single instruction, multiple data stream) are very effective for regular algorithms such as image filtering in large numbers of small regions. However, the MIMD (multiple instruction, multiple data stream) strategy appears to be more effective for large regions [6]. In general, the granularity, or grain size of an individual task to be processed on a computer system determines both the size and the total number of processors. Different types of tasks have different granularities and hence different optimal processor sizes and numbers.

Another perspective is found in centralized versus distributed control of processing. Centralized computer systems have either a single processor or tightly coupled multiple processors. In a tightly coupled system, all processors share a single large main memory, and communicate via shared variables. By contrast, a distributed computer system is a collection of modular autonomous (not necessarily homogeneous) computers which communicate with one another via message passing [7].

Distributed systems promise to be very effective in handling the parallel and nondeterministic processes of AI and knowledge-based processing; however, communication in such distributed systems is critical to their success. VLSI, wafer scale integration (WSI), and 3-dimensional structures reduce the bottlenecks resulting from the pins that are used to link chips to one another. However, a true complete connection (each processing element in an N-node network must have $N - 1$ interfaces to other processing elements) is too expensive, and even impossible for large N. Thus a multicomputer system that combines small local clusters of processors with point-to-

point links is desirable. The processors may be interconnected in a hierarchical manner to compose a more powerful computer system. Uhr [8] has emphasized the need to explore clusters and compounds of clusters for massively parallel computers. These kinds of computer architectures can be synthesized in a rather simple and elegant manner by combining the clusters to form a multilayer system. Arranging the processing elements to match the natural problem structure is important for optimal performance.

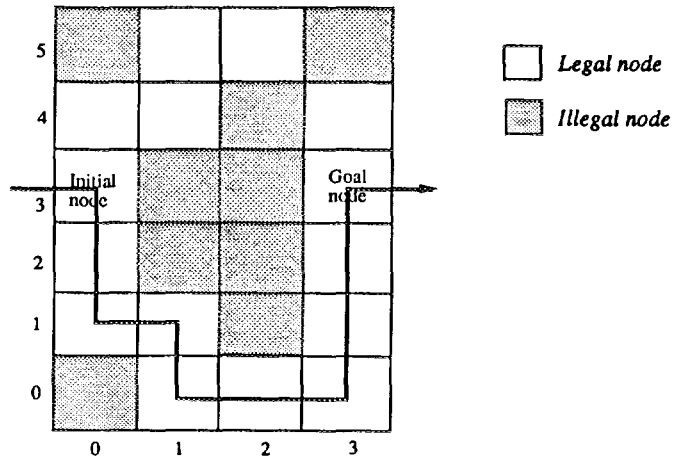
The field of distributed AI is concerned with the formulation of strategies of cooperation among the nodes in a distributed problem solving network [9]. However, such studies do not directly address the design of underlying computing architectures. VLSI or WSI (Wafer Scale Integration) multiprocessor systems for representing and manipulating semantic networks have been studied [10]. To date, limited success has been achieved for AI architectures due to the difficulty in parallelization of algorithms [11].

3. OBJECT-ORIENTED DESIGN

Empirical results, as mentioned above, indicate that different architectural schemes are matched to different computational requirements. Rather than trying to optimize an architecture based on a single structure, one can attempt to incorporate more than one paradigm in the same architecture. After briefly describing the DEVS-Scheme simulation environment, we illustrate how it helps to explore such multilevel, heterogeneous, distributed architecture designs. To understand the design of DEVS-Scheme as well as the design of intelligent machine architectures using object-oriented computing models, we need to review some basic concepts of object-oriented design.

3.1. Example

To set the stage for further discussion, let us consider an example of object-oriented programming. A maze search problem is illustrated in Fig. 1. This represents an elementary search problem [12] that we use as a running example. Let the white squares of a grid represent the allowable way-stations through the maze. A particular maze is completely specified by giving the set of white squares, an initial square, and a goal square, where the problem is to find a path through the white squares connecting initial and goal squares. Figure 1 defines a class of objects called *nodes* to form a tree that represents the search for a path from initial to goal node. Note from the class definition that every node object has instance variables, *my−x* and *my−y* (as well as others). We construct nodes to correspond, in a one-to-one manner, to squares by assigning the x and y coordinates of a square to the




```
class nodes
classvars
  initial:node
  goal:node
  goal-reached F
  legal-nodes:list

instvars
  my-x:int
  my-y:int
  myself:node
  myparent:node
  visited F

method generate-next-level()
visited:= T
Let nx,ny
 For each (i,j) in ((-1,0),(1,0),(0,-1)(0,1))
   nx := my-x + i
   ny := my-y +j
   if a node n, corresponding to (nx,ny) is found in the legal-nodes
   then send n expand (myself)
end

method expand(parent)
if not goal-reached
else if not visited
     then
          visited := T
          myparent := parent
          if goal = myself
          then goal-reached := T
          else (generate-next-level)
end
```

FIG. 1. Maze search formulated in object-oriented form.

my−x and *my−y* slots of the corresponding *node*. The root of the tree is the node that corresponds to the initial square.

Now we define two methods for the *nodes* class: *generate-next-level* and *expand*. As illustrated in Fig. 2, processing starts with the initial *node* executing its *generate-next-level* method. All four surrounding squares are generated but only their corresponding legal nodes are asked to do further processing using the *expand* method. In *expand*ing itself, a *node* checks whether its corresponding square has already been visited. If so, it is not visited again (this avoids endlessly looping cycles). To keep a trace of the path being generated, a new *node* records itself as a *parent* for each of its children. If the search terminates successfully, we can trace the found path by following the line of *parent* pointers back to the initial *node*. Note how class variables, such as *goal-reached*, are used to coordinate the search. Class vari-

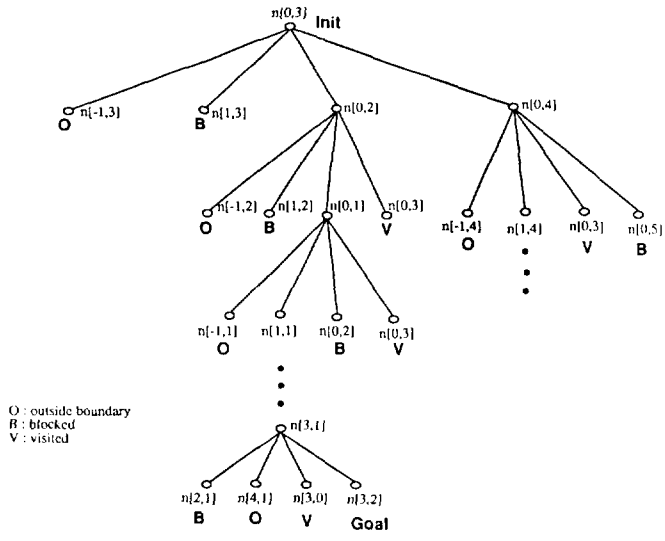


FIG. 2. The search tree for maze search.

ables are accessible to all member nodes and thus form a global means of communication among them.

4. A BRIEF REVIEW OF DEVS-SCHEME

DEVS-Scheme is a general purpose simulation environment for constructing hierarchical, modular discrete event models [4]. It employs the System Entity Structure (SES), a structural knowledge representation scheme that contains the knowledge of decomposition, taxonomy, and coupling relationships of a system necessary to direct model synthesis. The interaction of decomposition, coupling, and taxonomic relations affords a compact specification of a family of models for a given domain. This framework applies to all levels of abstraction in system design, from the high-level protocol specification for communication between processors to the low-level hardware description language. The SES serves as a means of generating the possible configurations of a system in a design solution space [13]. *Pruning* is a goal-directed process where the goal is formulated by the modeler (designer) to meet the system design requirements. Pruning reduces the set of candidate models to those suitable for the problem under study. The pruned entity structure (PES) contains all the information needed to synthesize a simulation model in a hierarchical fashion from components in the model base. To support synthesis, models in the model base are in modular form, i.e., they have recognized input and output ports through which all interactions with the external world are mediated. An important benefit of modular construction is that a model can be readily, and independently, tested by coupling test modules to it [4].

To test the putative advantages of multilevel computer architectures discussed earlier, Lee [14] employed DEVS-Scheme to study heuristic search problems on simulated parallel architectures using three control architectures, centralized, distributed, and multilevel, as portrayed in the SES of Fig. 3. The results provide preliminary evidence for the advantages of multilevel, heterogeneous, distributed architectures over conventional schemes.

The simulation study assumed that the controller (CONTROL) in the centralized architecture (implemented as a *controlled-model*) has a large store of search space states and local agents (LAGENT) send queries to the CONTROL, checking for novelty and potential worth of newly encountered search states. In the distributed architecture, the controller's capability and responsibility are distributed to all agent cells. Each agent cell (AC) consists of a distributed controller (DISTCON) and a local agent (LAGENT). The DISTCON has a relatively small store of encountered states. The LAGENT directs queries to the associated DISTCON. If not answerable in the DISTCON, the query is broadcast to other agent cells.

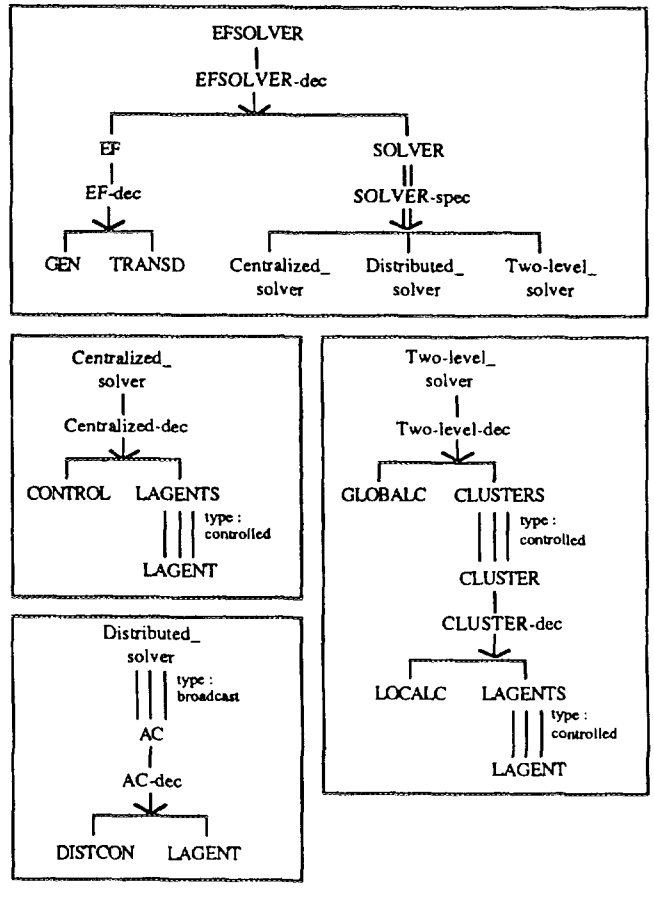


FIG. 3. System entity structure for family of alternative architectures.

The centralized control was also expanded to a multilevel architecture. Several agents were grouped to form a cluster, with each cluster having a local controller. The global controller (GLOBALC) communicates with the local controllers (LOCALCs) and the internal communications of a cluster are handled by the local controller. Each LOCALC has its own state store and controls the associated LAGENTs in the cluster.

Simulation verified that the communication bottleneck caused by the controller in the centralized model was reduced by distributing the control capabilities to the local controllers. However, communication traffic is much higher in the distributed architecture due to the broadcasting mode of information sharing. Therefore, the relative performances (measured in total problem solving time) of the architectures strongly depended on the characteristics of the interconnection network. Only with fast communication and message handling did the distributed architecture exhibit the best performance. In the multilevel architecture, the intensity of communication within clusters was greater than that between clusters. As a consequence, this architecture showed less sensitivity to communication delays, and exhibited the best performance for slow communication. It also performed close to the distributed architecture with fast communication. We can tentatively conclude that distributed control will be the architectural scheme of choice when faster transmission technologies such as optics become available, provided that message handling protocols are equally efficient. However, the results for the multilevel case bear out Uhr's contention that multilevel, clustered architectures should be further explored for their possible performance advantages.

5. MULTILEVEL ARCHITECTURE SUPPORTING SIMULATION-BASED DESIGN

We now show how DEVS-Scheme supports IMA design with an example of topdown development involving two levels of abstraction. We start with specification of an *object model* for concurrent programming of heuristic search. Then we consider the implementation of this model in a computer architecture. Both virtual machines are expressed within DEVS-Scheme.

The multilevel object model is suggested by examining the decision cycle in simulation-based design. Typically, this involves a two stage process: (1) simulation of a complex model is performed with the objective of evaluating its performance and (2) based on the results of simulation, the model, or its parameter set, is modified in an attempt to obtain improved performance. The cycle continues until satisfactory performance is obtained (or the model is terminated due to resource exhaustion). For example, in an aerodynamic simulation, the shape parameters of a wing may be adjusted in successive simulation runs until the sought-for lift and stability characteristics are found. Until now, there has been much research aimed at speeding up the *simulation* phase of the cycle [15]. However, we need to examine both phases, decision and simulation, of the cycle. Although typically only one simulation experiment is performed at a time, several such experiments could proceed concurrently under the control of the decision component. The latter decision component can be implemented using nonlinear optimization and AI strategies to select successive sets of experiments, rather than single experiments to be performed [16]. Genetic algorithms [17], for example, offer a promising, inherently parallel paradigm for parameter space search. They have been implemented on parallel machines but not for control of underlying simulation experiments.

Our first design goal is to enable a user to conveniently "program" simulation-based design studies in the object model. When this has been achieved, we can exploit the unique requirements of the object model to design a novel multilevel architecture to achieve significant speedup over conventional realizations.

To represent the decision and simulation components in the simulation-based design cycle we distinguish between "light" objects that organize the search, and the "heavy" objects they call on to perform computation-intensive simulation. As shown in Fig. 4, the model has two layers of objects: light and heavy. *Light* objects can fully interact with each other using direct messaging. *Heavy* objects are servers to light objects. A heavy object can receive commands from, and answer queries of, a light object that has its address. While doing so it cannot be interrupted by any other light object. Moreover, a heavy object cannot initiate any interaction with other light or heavy objects. Looking ahead, the implementation architecture will have corresponding levels for supporting light and heavy objects. The upper (light) level requires high connectivity but light computing power and the lower (heavy) level consists of independent heavy-

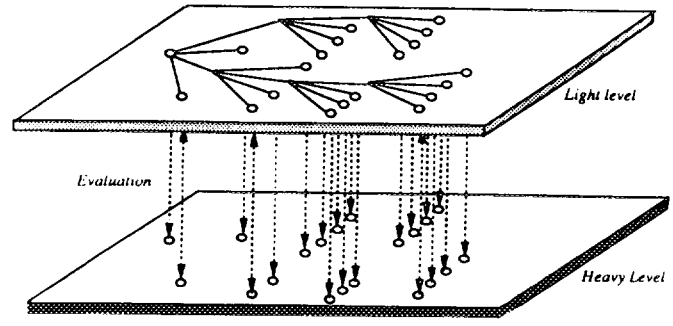


FIG. 4. Bilayered object model for simulation-based design.

duty computers linked only to the upper level. Each heavy-duty computer in the lower architectural level can itself be realized as a cluster of intercommunicating processors in order to achieve high performance simulation.

In the context of simulation-based design, the light objects represent search space points currently under investigation. The organization of the search is mediated by objects called *par-sets* which provide a concurrent implementation of optimization and AI search structures. *Par-sets* and other light objects are high level abstractions for convenient programming of heuristic search. They represent resources that need to be managed in a manner transparent to the programmer. We now consider how this can be done.

5.1. Concurrent Object-Oriented Systems

To understand the *par-set* concept first requires that we view objects as concurrently active processing agents. Several constructs such as futures and asynchronous messaging have been suggested to maximally exploit parallelism and concurrency in object oriented computing [18].

The *par-set* (parallel sets) concept is an object-oriented generalization of list structures. A *par-set* object has one instance variable, *members*, which is capable of holding a finite set of object names. Some of these names may be those of other *par-sets*. Methods for such a class include those for construction (e.g., inserting and deleting members) and querying (e.g., for membership of an object, for size of the set). However, the important methods concern those for coordination of member object behavior. Corresponding to LISP's map and map car forms are the methods *tell-all* and *ask-all*. When a *tell-all* message is sent to a *par-set*, the *par-set* retransmits the message arguments, the first of which is a method identifier, to all its members, thereby causing them to execute the message operation in parallel. An *ask-all* method is similar except that the transmitted message is a query and the *par-set* must package the returned values into a newly created *par-set* which is returned as its response to the *ask-all* query. Other methods are naturally provided to *par-sets*. For example, a variation of *ask-all* is *match-all* in which the members matching a given pattern are returned as a *par-set*.

To illustrate the *par-set* concept, let us redo the maze search examples as in Fig. 5. This time we use a *par-set* to perform a parallel search of legal nodes each time a next level candidate is generated. Since the *par-set* called *legal-nodes* queries all of its members in parallel it can potentially search for a match in constant time, i.e., in a time not depending on the number of elements in the list. Thus, the *par-set* concept embodies the decentralization spirit of object orientation and exploits the potential parallelism of such decentralization. A second form of *par-set* required for numerical ordering is discussed later.

```
class nodes
classvars
  initial:node
  goal:node
  goal-reached F
  legal-nodes:parset

instvars
  my-x:int
  my-y:int
  myself:node
  myparent:node
  visited F

method generate-next-level()
visited:= T
Let nx,ny
 For each (i,j)  in {(-1,0),(1,0),(0,-1)(0,1)}
   nx := my-x + i
   ny := my-y +j
   send legal-nodes tell-all (expand nx ny myself)
end


method expand(nx ny parent)
if not goal-reached
else if (and   x = my-x
               y = my-y
               not visited)
     then
         visited := T
         myparent := parent
         if goal = myself
         then goal-reached := T
         else (generate-next-level)
```

FIG. 5. Maze search formulated with *par-sets*.

5.2. Object Model Realization in DEVS-Scheme

Our task now is to show how the concurrent object model is realized in DEVS-Scheme. The *par-set* concept is the primary facility needed in the object model. It is expressed as a *forward-model* in DEVS-Scheme. A *par-set* object sends and receives messages through specific ports. It can be actively engaged in processing or wait passively for new input.

5.3. Search Example in DEVS-Scheme

To illustrate how the object model works, we implement the maze search example using the par-set realization just discussed. The SES (system entity structure) and PES (pruned entity structure) for the example are shown in Fig. 6. The SES has as its root a multiple entity, *objs*, so that when pruned and transformed, the resulting model is a *broadcast-model* containing any number of components of the types *par-set*, *node*, and *node-class-server*. After pruning, the PES reflects a choice of a single *par-set*, six *nodes* (so that the maze contains six legal squares), and one *node-class-server*. Figure 6 illustrates the resulting *broadcast-model* containing a *par-set*, *nodes*, and *node-class-server* as components. The *node-*

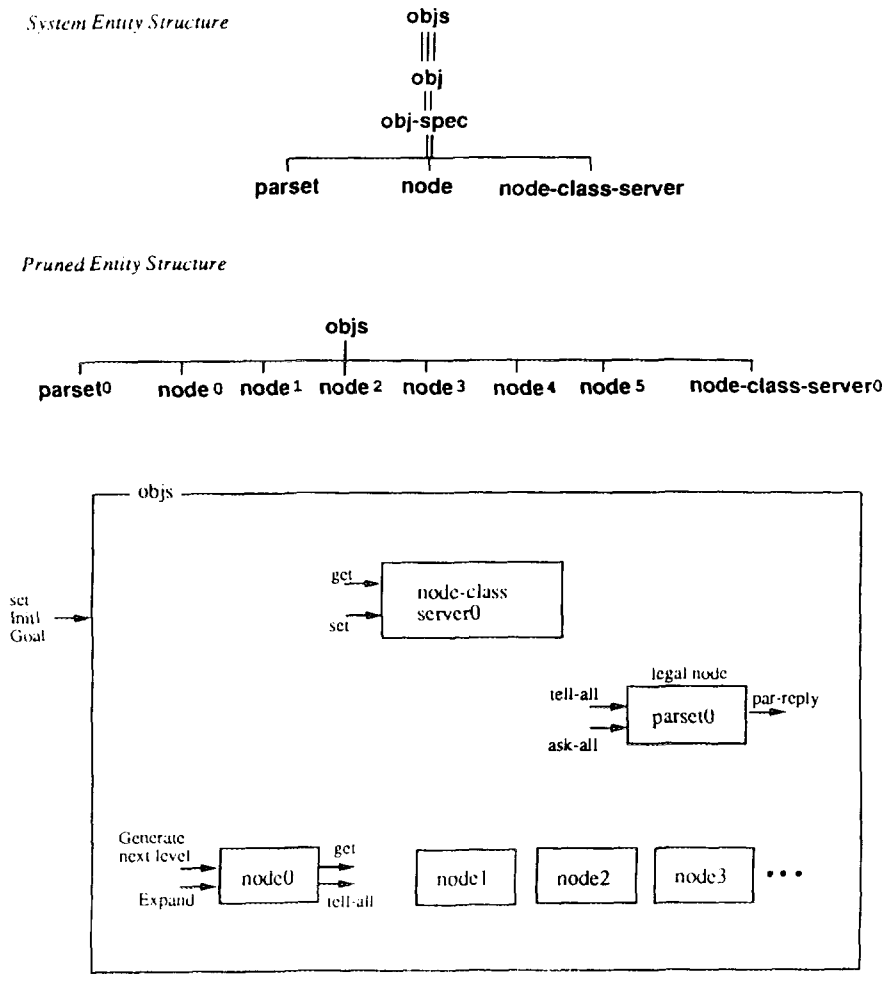


FIG. 6. SES, PES, and broadcast-model for the maze search.

class-server is needed to implement the global access afforded by the class variables. Such direct access is not allowed in a completely modular environment such as DEVS-Scheme where all interactions must be mediated by explicit input-output ports. The model is initialized so that the *members* slot of *par-set*0 is set to include all the *nodes*, each of which is also set to a corresponding white square. Then the *initial* and *goal* of *node-class-server* are set as desired. Finally, *node*0 is set into its initial state to start its method, *generate-next-level*. The simulation then starts. *Node*0 sends a *tell-all expand* message to *par-set*0 which relays the *expand* portion to all the *nodes*. The *nodes* all start their *expand* methods but only those that are adjacent to *node*0 actually generate next level children. Note that *nodes* are concurrently active, however, there is one bottleneck—*parset*0 is the only agent available to initiate next level processing. In the next example, we show how *par-sets* can be created on the fly so that such a bottleneck is removed.

5.4. *Optimization Search Specified in Object-Model*

For architectural considerations, Wah [19] classified AI search methods into divide-and-conquer, branch-and-bound, and general AND–OR graph searches. Genetic algorithms, GA [17], differ from such AI methods and are more akin to nonlinear optimization methods. Despite their differences, both GA and branch-and-bound employ numerical evaluation of search points to measure performance of the underlying simulated system and to guide the generation of new search points. In GAs, comparison of points is required to identify the best or the top k (an integer) best performers so that mating of pairs can produce new offspring. In branch-and-bound, lower bounds on heuristic function values are obtained to eliminate implausible candidates before actual evaluation. Note that here we consider the heuristic function as being produced in the lower simulation layer.

To facilitate numerical comparison of the above sort,

we employ a specialized form of *par-set* called *opar-set*, for ordered *par-set*. This subclass of par-sets will have a method for returning the k object members with the largest (or smallest) value of a given numerical attribute.

We now illustrate an optimization search "programmed" in the object model. In this example, the objective function shown in Fig. 7 is to be searched for its maximum value. The search takes place in the square cell with sides of length 16 as shown in Fig. 7. The search proceeds by continually dividing up cells into smaller cells and sampling the function values at the corners of each generated cell. The search is conducted so that as many cells are investigated concurrently as possible. Cells are light objects that have a method, *expand*, that evaluates the local square and also constructs offspring cells. At creation, a cell is given its corners description in a *par-set*. *Expand* asks each corner to evaluate the function at its *point* and stores the best (highest) value. Heavy objects are called upon to do the function evaluation. Each *point* stores its value to avoid reevaluation when called upon again. In simulation-based design applications, function evaluation would call upon a simulation and it would be important not to activate a heavy object needlessly. Children cells are concurrently *expanded* using a *par-set* and each one reports on the best value found. *Cells* continue expanding so long as their size is not below a criterion level. At termination, a tree of cells connected by the *children par-sets* is left and the best values from the left *cells* propagate upward, finally emerging as a single best value at the root.

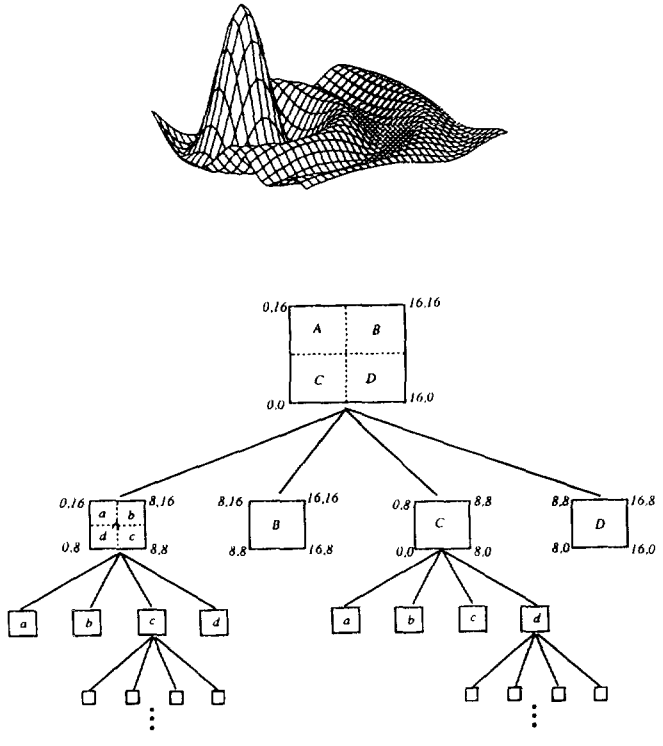


FIG. 7. Optimization search example.

Note that all the objects, both heavy and light, *can* be active concurrently in this example. However, whether they all *will* be active depends on whether there are enough resources at the implementation level to accommodate all this activity. In a moment, we turn to such architectural considerations. Usually, in a finite resource environment, however, intelligence must be brought to bear to guide the search process. Strategies such as branch-and-bound select nodes to expand based on estimated worth compared to the best found so far. Such strategies can be implemented with *par-sets* to provide global coordination, as does *legal-nodes* in the maze example.

5.5. *Architectural Support for Multilevel Object Model*

Figure 8 presents an SES laying out design alternatives for architectural support of the object model. Two major alternatives are apparent: (1) a multilevel architecture especially tailored to the object-model and (2) a "flat" conventional architecture which should provide a baseline of performance against which the tailored architecture can be measured. Note that the same atomic functional components—controllers (COS), message processors (MPS), simulation processors (SPS), and memory units (MS)—are found in both alternatives, but their organization and intercommunication are different. In the conventional architecture, the atomic units are packaged into computers (CS) that are interconnected in a network such as a hypercube. This kind of problem-insensitive distribution of work is forced by state-of-the-art, single-level, homogeneous architectures as alluded to earlier. In contrast, as in Fig. 8, the multilevel architecture features two interconnected layers of message and simulation processors coordinated by a global control and memory system.

The multilevel architecture combines the advantages of message-passing and shared-memory schemes in a novel way. The Light Layer (LL) contains the message processors (MPS) interconnected with a very high-speed interconnection network (HIN). The second Heavy Layer (HL) contains the simulation processors (SPS) interconnected with a distribution network (DN). The Global Control and Memory System (GCM), connecting the two layers, implements access to a global, shared-memory system for resource sharing. The major advantage of this multilevel interconnection strategy is that upper level communications between the decision-making processors can be fully overlapped with intensive processing at the lower level in order to achieve a high degree of parallel activity [20].

The LL provides high speed communication between

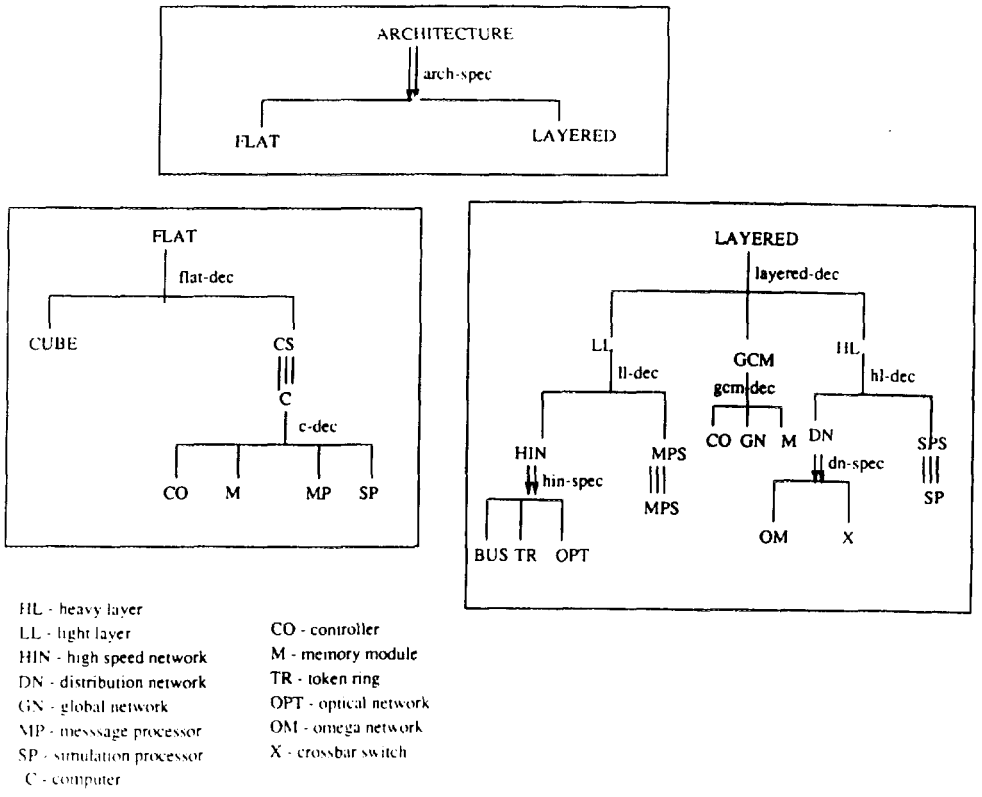


FIG. 8. SES for architectural alternatives.

light objects and *par-sets* that are implemented by the MPs. These latter need not have intensive processing capabilities. Fortunately, *par-set* communication requirements are restricted to transferring only short messages and tags between MPs. This minimizes the usual latency problems found in conventional networks since time-consuming data movement is reduced to a minimum.

The HIN supports multicasting whereby a *par-set* sends a message to its member light objects. Bus, token ring, and optical interconnection schemes are shown as alternative design choices. A conventional electronic broadcasting bus would not be practical for real world applications where the number of interacting objects may be in the thousands. A token ring network, such as employed in high traffic communications, is a second electronic alternative. However, the use of optics for high speed communications has been widely recognized as the ultimate solution to the fundamental problems of data communications in parallel systems. The parallel nature of optics and free-space propagation, together with their relative freedom from interference, make them ideal for high speed interprocessor communications. It is potentially feasible to build large optical interconnects that do not suffer from many of the limitations of their electronic counterparts [21]. Thus, an optical multicasting interconnection network is a design alternative worth exploring for the HIN.

The MPs request services from the SPs via the distribution network (DN). This network should provide good connectivity between a large number of MPs with a relatively small number of SPs. Multicasting is not needed at this level. The DN achieves a dynamic mapping using a reconfigurable network with a short response time.

Our approach to global control and memory is quite different from existing approaches. The GCM system acts like a scheduler and load balancer more than a controller. It keeps tack of busy and idle SPs, and allocates SPs to MPs. Once a SP is free, its status is reported to the GCM. Upon a request from an MP, and GCM checks its reservation table and dispatches an idle SP to the requesting MP. Global information such as a best-so-far objective function value is stored in the global register space in the GCM. Also, simulation results are stored in the GCM memory modules for rapid access by both MPs (to decide on next generation trials) and SPs (to initialize from states identical, or similar, to states already encountered). Thus the interconnection network in the GCM system must provide fast memory access to both MPs and SPs. This requires a circuit-switched network with extensive redundancy for conflict avoidance and fault tolerance. Several interconnection schemes such as omega

networks and crossbar networks have been proposed for this purpose [22].

It should be emphasized that the shared memory system is not used for process synchronization but only for computation. The SPs are not intended to directly communicate with each other. They store potentially voluminous data such as the final state reached after a simulation run (or even the whole trajectory) in the shared memory for subsequent use in a next simulation. They also report abstracted results to the GCM for use by the MPs. Fast means of accessing the global register file in the GCM, scheduling processes on the SPs, and reporting back to the MPs must be designed.

Note that the multilevel architecture combines fine and medium grain parallelism. The upper layer is composed of fine-to-medium-grain-size processors with emphasis on communication, and the lower layer contains medium-size processors with more emphasis on processing power. Such processors may be equipped with vector and array processing capabilities for parallel processing of structured data such as matrix manipulations, and numerical processing. In fact, each SP can be considered as a cluster of several processors with local storage and local interconnects connecting these processors. Clusters could be, for example, processor arrays dedicated to partial differential equation simulation [23], or discrete event distributed simulators [14].

Specialized capabilities to perform numerical ordering using content addressable memories for the realization of ordered *par-sets* should be considered. Another important aspect of the complexity of the processing elements is the total number of objects permissible at any given time and their allocation/deallocation to the MPs and SPs. Given that we have a finite computing and communication resources, strategies will have to be developed for allocating objects onto these resources and limiting parallelism explosion that, as we have seen, can easily arise in searching.

5.6. *Simulation-Based Design of Architectures*

The design of architectures for simulation-based design is itself an exercise in simulation-based design! We have formulated the design alternatives to be considered in a system entity structure (Fig. 8). By pruning the SES, we can generate alternative PESs that are transformed to models simulatable in DEVS-Scheme. However, before such performance-oriented simulations can be performed, experimental frames for solving simulation-based design search problems must be developed that provide realistic workloads for the architecture models. Existing examples such as the knapsack problem [19] and the traveling salesman problem [14] serve as a basis for developing such experimental frame problem sets. Examples of genetic algorithm applications are readily available in the literature [17]. Also, searching of the system entity structure itself offers an interesting class of problems. FRASES is an object-oriented representation of the SES with automated constraint-based pruning [12]. Constraints may still not resolve alternatives to unique selections, so pruning may result in a family of model candidates to be evaluated by simulation—a family whose size is combinatorial in the numbers of unresolved alternatives.

Having an experimental frame to determine the workload and the performance indexes of interest (e.g., total problem solving time), we are in a position to compare the performance of various alternatives that we can prune from the master SES. More than this, we can investigate the behavior of mechanisms underlying such alternatives. For example, mechanisms for controlling the "turnover" of search points (the number of new points replacing existing points) may be important to balance the task completion rates at the two layers of object-model (i.e., time to generate points versus time to evaluate points). As a consequence, an asynchronous approach to population generation may have load balancing advantages over the usual synchronous approach. Simulation of such mechanisms can provide insight into which ones are superior and why.

6. DISCUSSION AND CONCLUSIONS

We have illustrated the features of the DEVS-Scheme modeling and simulation environment in the context of design of intelligent machine architectures (IMAs). Several knowledge-based simulation environments exist [24] that are applicable to IMA design. However, DEVS-Scheme is the only simulation system to attempt to meet all three requirements enumerated earlier: numerical–symbolic expressive power, support of multiple levels of abstraction, and model-base reusability. Let us see how well DEVS-Scheme measures up against these requirements.

- Expressive Power: We have seen that the object model for simulation-based design requires both symbol manipulation, such as object matching, and numerical computation, such as objective function evaluation. We illustrated how DEVS-Scheme can provide such facilities using its underlying Scheme language base. The multilevel architecture realization of the object model exhibited hierarchical structure as well as the intense communication demanded by distributed architectures. Again we showed how DEVS-Scheme could model such hierarchical structure with its knowledge representation scheme, the system entity structure, and the communication among processors, with its coupling concepts.

• Levels of Abstraction: The object model was first given a direct implementation and then architectural support was discussed, both using DEVS-Scheme. Thus, we have illustrated DEVS-Scheme's ability to represent design abstractions at the conceptual level, where proof-of-concept can be established, and the functional level, where details of functionality can be worked out. Although design refinement could continue down further, say to the gate level, DEVS-Scheme is not designed to do so. It is in this realm that conventional VLSI design can take over. We have illustrated how DEVS-Scheme provides performance feedback through its efficient discrete-event simulation facilities. Also the hierarchical testing approach facilitated by its object-oriented base can provide a high degree of confidence in the correctness of an implementation. However, DEVS-Scheme cannot provide the complete confidence that a formal proof-of-correctness system can. It would be useful to combine a simulation environment with a theorem prover to achieve the best of both worlds. Since models in DEVS-Scheme are expressed in a formal manner (the DEVS formalism) such a combination might well be feasible.

• Reusability: We have seen that classes in DEVS-Scheme provide useful forms of atomic model specification and hierarchical coupling. These classes may be extended and refined for reuse. Moreover, a model base may be built up with reusable components and organized using the system entity structure tools. Some overhead is incurred in providing for such reuse. The user must make deliberate efforts to assure that models conform to the hierarchical, modular structure required by DEVS-Scheme. Moreover, the running time of such models tends to be slower compared to "flat" (nonhierarchical, nonmodular) models in traditional languages. However, this drawback is likely to become less important as symbolic languages such as Scheme become faster. Even now, a version of Scheme has been developed for the Motorola Delta workstation that runs only 50% slower than C. Moreover, the time saved in development through reusing existing components in the model base should vastly overshadow the time lost in slower simulation. Certainly, the time to achieve results with DEVS-Scheme is short compared to the actual time needed to build and test an ultimate physical system. The expressive power of DEVS-Scheme guarantees that models can be built with necessary fidelity so that such results are not only quickly obtained but also useable.

Finally, let us remember that our example of IMA design using DEVS-Scheme concerned architectural support of simulation-based design. If the results of such a design are successful, then we may eventually have high performance object-oriented parallel platforms for implementing simulation environments such as DEVS-Scheme itself. This is the bootstrapping process that we referred to initially!

ACKNOWLEDGMENTS

The first author, BPZ, performed much of the writing of this work while on sabbatical at the Basser Department of Computer Science, University of Sydney, whose excellent support is gratefully acknowledged. Jinwoo Kim, of the ECE Department of the University of Arizona, helped in the research and production of the manuscript.

REFERENCES

1. Hwang, K., and Degroot, D. *Parallel Processing for Supercomputers and Artificial Intelligence*, McGraw–Hill, New York, 1988.
2. Garzia, R. F., Garzia, M. R., and Zeigler, B. P. Discrete-event simulation. *IEEE Spectrum* (Dec. 1986).
3. Booch, G. *Object-Oriented Design with Applications*. Benjamin/Cummings, Redwood City, CA, 1991.
4. Zeigler, B. P. *Object-Oriented Simulation with Hierarchical, Modular Models: Intelligent Agents and Endomorphic Systems*. Academic Press, NY, 1990.
5. Denning, P. J., and Tichy, W. F. Highly parallel computation. *Science* **250** (1990). 1217–1222.
6. Willebeek-LeMair, M., and Reeves, A. P. Solving nonuniform problems on SIMD computers: Case study on region growing. *J. Parallel Distrib. Comput.* **8**, 2 (Feb. 1990), 135–149.
7. Gray, J. N. An approach to decentralized computer systems. *IEEE Trans. Software Engrg.* **se-12**, 6 (June 1986).
8. Uhr, L. *Multi-Computer Architectures for Artificial Intelligence: Toward Fast, Robust, Parallel Systems*. Wiley, New York, 1987.
9. Gasser, L. Distributed artificial intelligence. *AI Expert* (July, 1989), 26–33.
10. Delgado-Frias, J., and Moore, W. R. A WSI semantic network architecture, in J. Delgado-Frias and W. R. Moore (Eds.). *VLSI for Artificial Intelligence*. Kluwer Academic, Boston, 1989, pp. 144–156.
11. Dixit, V. V., and Moldovan, D. I. The allocation problem in parallel production systems. *J. Parallel Distrib. Comput.* **8**, 1 (Jan. 1990), 20–29.
12. Kanal, L., and Kumar, V. (Eds.) *Search in Artificial Intelligence*. Springer-Verlag, New York, 1988.
13. Rozenblit, J. W., Hu, J., and Zeigler, B. P. Knowledge-based design and simulation environment (KBDSE): Foundational concepts and implementation. In G. I. Doukidis and R. I. Paul (Eds.). *AI in Operations Research*. MacMillan Press, London, 1988, pp. 261–276.
14. Lee, C. A hierarchical modelling and simulation environment for AI multicomputer design. Doctoral Dissertation, ECE Dept., Univ. Arizona, 1990.
15. Misra, J. Distributed discrete event simulation. *ACM Comput. Surv.* **18**, 1 (1986), 39–65.
16. Birta, L. Optimization in simulation studies. In M. S. Elzas, T. I. Oren, and B. P. Zeigler (Eds.). *Simulation and Model-Based Methodologies*. Springer-Verlag, Berlin/New York, 1984, pp. 451–476.
17. Goldberg, D. Genetics-based machine learning: Whence it came, where it's going. In M. S. Elzas, T. I. Oren, and B. P. Zeigler (Eds). *Modelling and Simulation Methodology: Knowledge Systems Paradigms*. North-Holland, Amsterdam, 1989.

18. Yonezawa, A., and Tokoro, M. *Object-Oriented Concurrent Programming*. MIT Press, Cambridge, MA, 1987.
19. Wah, B. W., Li, G. J., and Yu, C. F. Multiprocessing of combinatorial search problems. *IEEE Comput.* (June, 1985), 93–108.
20. Almasi, G. S., and Gottlieb, A. *Highly Parallel Computing*. Addison–Wesley, Reading, MA, 1989.
21. Louri, A. 3-D optical architecture and data-parallel algorithms for massively parallel computing. *IEEE MICRO* (Apr. 1991).
22. Bhuyan, L. N. Interconnection networks for parallel and distributed processing. *IEEE Comput.* **20,** 6 (June 1987), 9–12.
23. Kahle, B. A., and Hillis, W. D. The connection machine model CM-1 architecture. *IEEE Trans. Systems Man Cybernet.* **19,** 4 (July/Aug. 1989).
24. Fishwick, P. A., and Mojeski, R. J. Knowledge-Based Simulation Environments. Springer-Verlag, New York, 1990.

BERNARD P. ZEIGLER is Professor of Electrical and Computer Engineering at the University of Arizona, Tucson. He received his B. Eng. Phys from McGill University, 1962, his M.S.E.E. from MIT, 1964, and his Ph.D. from the University of Michigan in 1969. He has published over 100 journal and conference articles in modeling and simulation, knowledge based systems, and high autonomy systems. His first book, "Theory of Modeling and Simulation" (Wiley, 1976), is regarded as one of the foundational works in the field. A second book, "Mutifacetted Modelling and Discrete Event Simulation" (Academic Press, 1984), was given the outstanding simulation publication award by TIMS College on Simulation in 1988. Zeigler's current research on simulation methodology is described in a new book, "Object-Oriented Simulation with Hierarchical, Modular Models: Intelligent Agents and Endomorphic Systems," published by Academic Press, Boston, 1990. His research has been supported by federal agencies including NSF, NASA, and the U.S. Army, as well as industrial sponsors including Siemens, McDonnell Douglas, and Motorola. He is founder of a university spin-off company for technology transfer of his simulation environment concepts and the recipient of an SBIR (small business inovative research) grant from the U.S. Army.

Received February 10, 1992; revised April 15, 1992; accepted June 10, 1992

AHMED LOURI earned the Diplome d'Engenieur (Engineer Degree) from the University of Science and Technology of Algeria in electrical engineering, an M.S. Degree in computer engineering, and a Ph.D. in computer engineering from the university of Southern California (USC). From 1982 to 1988, he was the recipient of a National Talent Search Scholarship. From 1986 to 1988, he worked as a researcher with the Computer Research Institute (CRI), USC, where he conducted extensive research in parallel processing, multiprocessor system design, and optical computing. Since September 1988, he has been on the faculty of the Department of Electrical and Computer Engineering at the University of Arizona, where he is currently an assistant professor. His areas of research include high-speed parallel processing, optical computing, and optical interconnects. He has published numerous journal and conference articles. His research has been supported by the National Science Foundation (NSF) as well as industrial sponsors. In 1988 he was the recipient of the NSF Research Initial Award. Dr. Louri is a member of IEEE, ACM, OSA, and SPIE.