Journal of Intelligent and Robotic Systems 1 (1988) 299–309.

Design of a Simulation Environment for Laboratory Management by Robot Organizations*

BERNARD P. ZEIGLER, FRANÇOIS E. CELLIER, and JERZY W. ROZENBLIT
Dept. of Electrical and Computer Engineering, University of Arizona, Tucson, AZ 85721, U.S.A.

(Received: 14 September 1988)

Abstract. This paper describes the basic concepts needed for a simulation environment capable of supporting the design of robot organizations for managing chemical, or similar, laboratories on the planned U.S. Space Station. The environment should facilitate a thorough study of the problems to be encountered in assigning the responsibility of managing a nonlife-critical, but mission valuable, process to an organized group of robots. In the first phase of the work, we seek to employ the simulation environment to develop robot cognitive systems and strategies for effective multi-robot management of chemical experiments. Later phases will explore human-robot interaction and development of robot autonomy.

Key words. Simulation environment, robot organization, laboratories for space station, robot cognitive system.

1. Introduction

This paper describes the design of a simulation environment capable of supporting the study of robot organizations for managing chemical, or similar, laboratories aboard Space Station. Laboratory management includes the servicing and calibration of equipment, the set-up of experiments to external specifications, the monitoring and control of experiments in progress, the measurement of results, and finally the recording and analyzing of data. The environment should facilitate a thorough study of the problems to be encountered in assigning the responsibility of managing a nonlife-critical, but mission valuable, process to an organized group of robots.

Our ultimate research goals are to employ the simulation environment to develop robot cognitive systems and strategies for effective multi-robot management of laboratory experiments. We seek an understanding of how to partition automation tasks between hard and soft forms, i.e., between 'intelligent' instruments and flexible robots. We shall assess the nature of human supervision initially required, and seek to develop workable man-robot cooperation protocols. Also, we seek to develop robot learning paradigms such that the autonomy of a robotic organization increases with experience, and consequently, the need for human supervision and intervention is diminished.

* Supported by NASA-Ames Cooperative Agreement No. NCC 2-525, 'A Simulation Environment for Laboratory Management by Robot Organizations'.

It is timely to begin exploration of advanced robot-controlled instrumentation. For example, handling fluids in orbit will be essential to many of the experiments being planned in manufacturing and biotechnology. However, the microgravity conditions of space necessitate radically different approaches to fluid handling than common on earth. As experience in space accumulates, approaches and instrumentation will likely undergo continual modification, enhancement, and replacement. Thus, robots for managing such equipment must be sufficiently intelligent and flexible so that constantly changing environments can be accommodated. Given its importance and novelty, we have chosen fluid handling in microgravity as the focus for our laboratory environment.

Discrete event simulation and AI knowledge representation schemes form a powerful combination, called knowledge-based simulation, for studying intelligent systems in a realistic manner [1, 3, 5–8]. DEVS-Scheme [14, 15] is a knowledge-based simulation environment for modelling and design that facilitates construction of families of hierarchical models in a form easily reusable by retrieval from a model base. The laboratory environment, implemented in DEVS-Scheme, is being constructed on the basis of object-oriented and hierarchical models of laboratory components at multiple levels of abstraction.

The robot cognition model is based on an 'action-by-exception' principle control of a knowledge base of Model-Plan Units (MPUs). Davis [2] described a similar knowledge based simulation environment in which agents are governed by a script (plan of actions) and a set of production rules for deciding when to proceed from one phase of the plan to the next, which detailed actions to execute within a phase, and what to do if one plan has to be replaced by another one. Holland's [4] classifer (parallel production rule) system provides concepts for sequencing robot actions.

In designing the robot models, we assume that necessary mobility, manipulative and sensory capabilities exist so that we can focus on task-related cognitive requirements. Such capabilities, the focus of much current robot research, are treated at a high level of abstraction obviating the need to solve current technological problems. Organizational issues are introduced from the beginning since individual robot capabilities may be much influenced by cooperative requirements.

A primary goal in the robot model design is to minimize the number of sensory inputs that the system must attend to at any one time. Except at critical selection points, attention is focused on only those aspects of the environment dictated by the currently activated MPU. While an MPU behavior lies within its envelope, no other MPU can supplant it, even if its initialization conditions better fit the current situation. One of the primary goals of the project will be to judge whether these principles provide a workable basis for intelligent robot design. For example, such robots may be so single-minded as to be incapable fo flexibily responding to an unknown or changing environment.

1.1. DEVS-SCHEME SIMULATION ENVIRONMENT

DEVS-Scheme [14, 15] is a knowledge-based simulation environment for modelling and design that facilitates the construction of families of models in a form easily

reusable by retrieval from a model base. The environment supports the construction of hierarchical discrete event models, and is written in the PC-Scheme language which runs on IBM compatible microcomputers and on the Texas Instruments Explorer System.

1.2. SYSTEM ENTITY STRUCTURE KNOWLEDGE REPRESENTATION

Model specification and retrieval in the DEVS-Scheme simulation environment is mediated by a knowledge representation component designed using the system entity structuring concepts [16, 19]. The system entity structure incorporates decomposition, taxonomic, and coupling knowledge concerning a domain of real systems [10, 12]. A user prunes the entity structure according to the objectives of the modelling study obtaining a reduced structure that specifies a hierarchical discrete event model [11]. Upon invoking the transform procedure, the system searches the model base for components specified in the pruned entity structure, and synthesizes the desired model by coupling them together in a hierarchical manner. The result is a simulation model expressed in DEVS-Scheme which is ready to be executed to perform simulation studies.

1.3. PROCESS LABORATORY MODEL

The laboratory environment is being constructed on the basis of object-oriented and hierarchical models of laboratory components within DEVS-Scheme. Laboratory configurations will be determined by pruning the entity structure knowledge representation. The laboratory model is being designed to be as generic as possible. However, as stated, the focus will be upon fluid handling in microgravity which presents a variety of problems that are unique to space.

Figure 1 illustrates the approach taken. The entity structure for Space Station Laboratory decomposes this entity into materials, action plans, a workspace, instruments, and the robot system. Each of the latter entities will have one or more classes of objects (models expressed in DEVS-Scheme) to realize it. Materials will be specialized by physical state into the classes gas, liquid, and solid, and will be further subclassified as needed. Action plans are composed of unit operations which decompose into four types: transportation (which changes the physical co-ordinates of a material, e.g. pumping), transformation (which transforms the state of a single material), combination (which produces a new material from several input materials, e.g. chemical reactions), and separation (which partitions a material into several components).

Unit operations are carried out with one or more instruments, which may be transporters, transformers (e.g. chillers), combiners (e.g. mixers), or separators. To illustrate the special nature of space, consider transporters. Since air/liquid interfaces are not permitted under microgravity conditions, standard earth-bound containers, such as beakers, cannot be used. A design of a space adapted 'beaker' would have an aluminium bottle containing an inflatable bag, which is the actual liquid

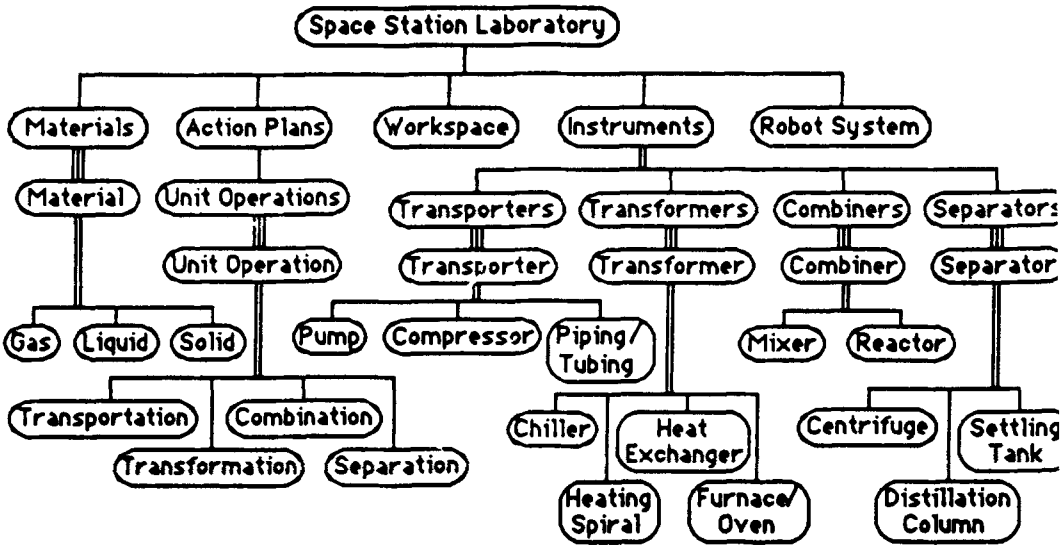

Fig. 1. System entity structure for Space Station Laboratory.

container; liquid is injected/extracted by means of syringes; air pressure between the outside of the bag and the inside of the bottle wall ensures that the bag remains 'full' at all times.

Action plans are sequences of unit operations with associated materials and instruments. For example, injecting several liquids into a bottle, placing the bottle in a shaker, and then placing it in a heating spiral is a sequence related to experimentation with chemical reactions. Action plans have associated models whose construction will be discussed in the context of the robot Model-Plan Unit (MPU).

Instruments have attributes which include operational conditions so that normal and abnormal operating behaviour can be studied.

To set up a particular laboratory environment, the laboratory entity structure is pruned to create a pruned entity structure which will transform into a laboratory model. Constraints on the possible configurations of components, especially those imposed by microgravity and space environments, are captured by appropriate synthesis and selection rules [9].

2. Robot models

2.1. DESIGNING MODEL-PLAN UNITS

The first stage in designing Model-Plan Units, involves modelling of continuous processes by discrete event models [13]. We start with a particular real process, such as heating liquid in a doubly-contained bottle. We identify regions of operations such as: "there is a sufficient amount of liquid in the bottle", "the liquid has reached the desired operational temperature", "the air pressure is too high", "the bottle has exploded". A continuous dynamical model is then developed for each region based

on physico-chemical considerations. Boundaries between regions are then identified, and a discrete event model is specified whose internal events represent such transitions from one boundary to another. Scheduling of such transitions is based on time-to-next-event values obtained from trajectories of the dynamical model. For example, if the initial quantity of liquid and the rate at which it heats up are known as well as the increase in air pressure with temperature, then the time to reach the "air pressure too high" region can be pre-determined, and hence scheduled.

For each action on the real process, a normal state trajectory is identified in the continuous model and projected into the space of sensor measurements. An envelope is determined to enclose this projected state trajectory. This envelope specifies the variation to be tolerated in sensor measurements while still accepting an observed trajectory as normal. For computational feasibility, sensors with binary states (or a small set of discrete states) will be preferred, circumstances permitting. For example for sterilizing a liquid, that liquid should be heated up to at least 70 degrees centigrade, and should be kept at that temperature for a prescribed period of time. For other reasons, it may not be advisable to heat the liquid beyond 80 degrees centigrade. So, we may employ a sensor whose binary output indicates the temperature lying within, or outside of, the range 70–80. In addition to sensory boundaries, we employ timing information to determine normal operation. From estimated uncertainties in the initial state and parameters, the continuous model yields the window in which the time-to-next-event must lie for each state transition [18].

The plan of an MPU specifies a sequence of unit-operations to be carried out to bring the real process from an initial state to a desired state. For example, an MPU for sterilizing water might specify filling a bottle with water, placing it in a heating spiral, and removing it when the required temperature of 70 degrees centigrade has been reached. Associated with each unit operation is a set of sensors for detecting its initialization and goal states together with the time window in which each transition time must lie. If, for example, the time for the temperature sensor to change to its high state is not within that allowed, the MPU is disabled; if the time is within bounds, the next action, removing the bottle, is carried out.

2.2. PLAN ABORTION AND RESTART

Following disablement of an MUP, other MPUs may be activatable in the prevailing state. MPUs may, for example, exist if the process state is still a normal one; as a special case, the last activated MPU may be still have its initialization conditions satisfied. Consider for example, a situation where the envelope timing bounds associated with the action "heat water" were exceeded because the power to the heating spiral was turned off. The Selector must prevent the aborted MPU from gaining activation and attempting indefinitely to use an inoperative heater. This might be done with a recency, or frequency-of-use component in the selector's conflict resolution method.

Diagnostics are associated with MPU plan abortion. Those diagnostics will attempt to discover faults which can be corrected to return the state to one in which normal

operation can be resumed. Such diagnostics are guided by the sensor envelope and time window violations which caused the MPU to abort. For example in the above situation, given that the expected heating time was exceeded, a diagnostic may deduce the heating spiral is not producing heat, and that one cause might be that power to it is turned off. The model underlying an MPU provides the basis for designing such diagnostic units.

Naturally, we expect that many unforeseen situations will emerge in simulation runs. In such cases, the robot system will fail. Since the current state of the process, as viewed in the last activated MPU and its sensor readings (or a record of the most recently activated MPUs) is available, we will be in a position to analyze what went wrong. It is expected that, in a real space laboratory, such events will also occur. Protocols will be investigated to alert human supervisors to such events, and facilitate restoring robot system operation.

3. Implementation

A simplified entity structure for the implemented robot organization is shown in Figure 2. This structure is transformed into a hierarchical model containing 'controlled-models' at two levels. 'Controlled-models' is a class in DEVS-Scheme which facilitates the construction of models containing arbitrary numbers of components which communicate with each other and with the outside world via a controller [17]. Thus at the top level, the robot-system is a controlled-model containing a space-manager as controller, and robots as components. Each robot contains a motion, a sensory, and a cognition sub-system. The cognition-system is itself a controlled model containing the selector as controller, and MPUS as components.

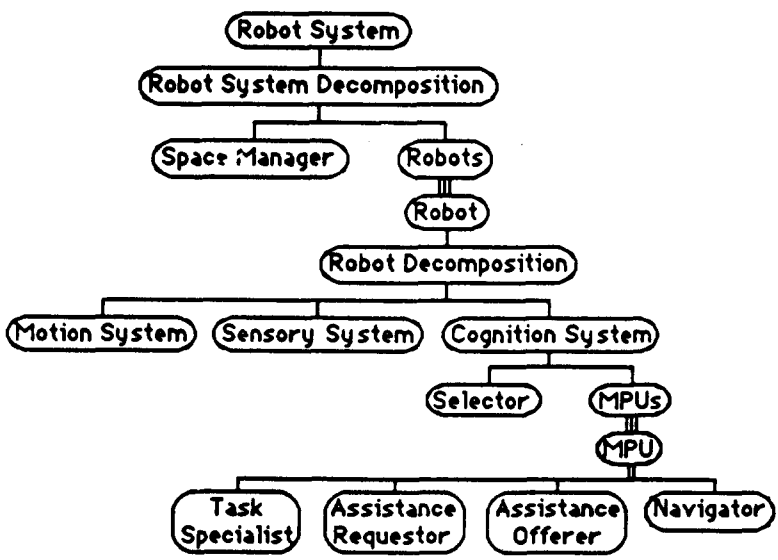

Fig. 2. Entity structure for the robot organization.

Robots use their sensory subsystems to communicate with each other via the space-manager. When a robot changes its position, its motion sub-system sends its new location to the space-manager which keeps track of the robots' positions. When a robot wishes to communicate with other robots, it sends its message to the SPACE-MANAGER which relays the message only to those robots that are located within the range of the sender. This range may vary depending on the channel on which the message is sent. In this way, different transmission media and sensory modalities may be modelled: light and vision, sound and hearing, pressure and touch, etc. Latency in message transmission, implemented by a delay in the space-manager, may also depend on the medium. Messages on certain channels, such as touch, are reflected back to the space-manager by sensory-subsystems upon arrival to a sensory subsystem, as well as being transmitted to the cognition system. Such echo messages are used by the original sender to ascertain relationships to the receiving robot.

Since the space-manager has complete knowledge of locations, it can detect collisions between robots. Space may be treated as a resource shared by its occupants so that collisions represent attempts to occupy the same space twice at the same time. The 'management' of the resource is 'dumb' if collisions are only *detected.* However, the space-manager may be given additional intelligence to co-ordinate the robots, e.g. to *prevent* collisions, thus modelling an artificial layer of supervision above the naturalistic one.

Within each robot's cognition system, action-by-exception control ensures that an MPU, once initiated, retains activation until its plan is successfully executed, or until a significant discrepancy arises between the actual results of carrying out the plan and the results expected by the model. The selector is essentially a bi-state device whose state is determined by the MPU responses. In the closed state, it passes on the incoming sensory inputs to the activated MPU. Upon completion of the activated plan or upon receiving a discrepancy alert, it switches to the open state in which MPUs may vie for activation. Incoming sensory input is broadcast to all MPUs. The first MPU to respond to the input in established as the activated MPU. Once an MPU activation has occurred, the closed state is resumed.

As stated, a primary goal in this design was to minimize the number of sensory inputs that the system must attend to any one time. This is achieved here by the fact that, in the closed state, the selector acts like a closed wire which uncritically transmits all inputs to the activated MPU. The latter only pays attention to those inputs which matter to achieve its goals.

In the future, we intend to implement more general conflict resolution schemes to determine which of the activatable MPUs will be granted permission to activate. MPUs will be arranged in a generalization hierarchy, an inverted tree with relatively few highly general MPUs at the lowest level. Such generalists cover most of the environment, but with less than fully efficient capabilities. Successive levels contain specialist MPUs with increasingly refined models and plans. Of those MPUs responding to an input in the open state, the selector may choose the one with the highest specificity level. Such an architecture supports learning of new MPUs in the manner described by Holland [4].

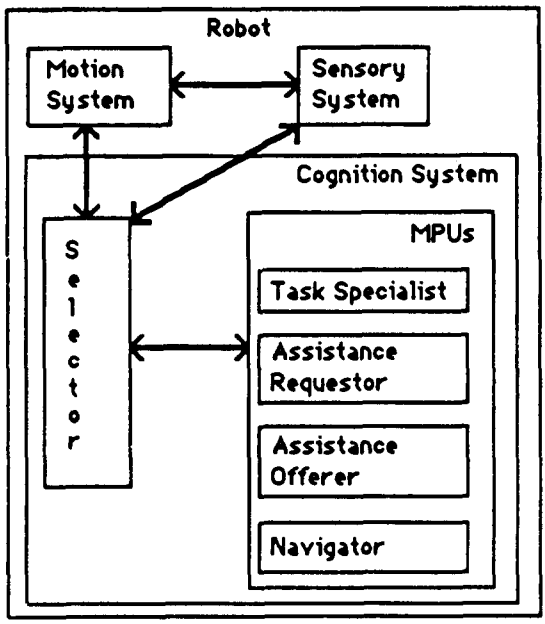

Fig. 3. Prototype robot model.

The MPUs comprising the robot brain are of two kinds: those specialized for carrying out specific laboratory tasks and those specialized for more general tasks involving communication, motion, co-operation, etc. A prototype minimal configuration illustrated in Figure 3 contains two robots each with the following MPUs:

Task Specialist MPU: specialized for executing a particular experiment related task, requests help when needed in performing this task by relinquishing control to the Assistance-Requestor.

Assistance-Requestor: MPU specialized for the task of requesting help from other robots. When it is activated, it initiates a protocol which tries to make contact with robots within its range and to engage one which can provide the needed assistance.

Assistance-Offerer: MPU specialized for the task of dealing with incoming requests for help emitted from Assistance-Requestors of other robots. When activated, it decides if help can be offered, and if so, engages in a dialogue with the Assistance-Requestor of the help-seeking robot and sets up a rendezvous. It relinquishes control to the navigator to bring the robot to the requestor's work site. The assistance offer is most easily activated when the robot is idle, and the selector is in its open state. To replace an already activated MPU (in its closed state), the latter MPU must be able to accept incoming requests for help and relinquish control.

Navigator: MPU specialized for directing the motion sub-system to bring the robot to a given destination. It requests the current motion state from the motion component, and sends it new parameters (direction, speed, and time-step) for travelling to the vicinity of the destination. Once there, it directs the motion component in

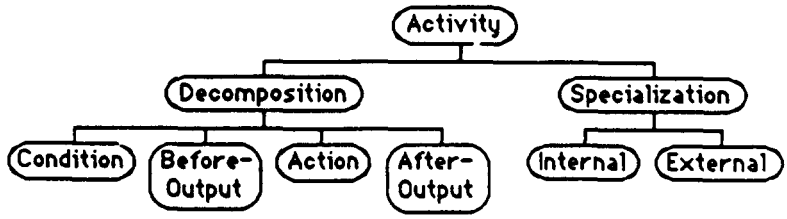

Fig. 4. Structure of an activity, i.e., rule for prescribing MPU state transitions.

physically contacting the object or robot at the destination. The touch channel is used for judging when contact has been made.

Elementary scenarios in which the model has been tested are: (a) one robot requests assistance, one robot available to offer help; (b) one Assistance-Requestor, two Assistance-Offerers available; and (c) two Assistance-Requestors, one Assistance-Offerer available. In case (b), the first offerer to respond engages with the requestor. The second one receives no confirmation and returns to its previous state. In case (c), the first requestor to engage with the offerer is helped. The second one continues to send out assistance requests.

The MPUs are developed as objects in the class 'forward-models' of DEVS-Scheme. Models in this class are specified in a rule-based programming paradigm. As shown in Figure 4, a rule, called an activity, is a structure which contains condition and action slots, as usual, and in addition, slots for specifying outputs to be produced before and/or after the action is performed. An action specifies a change in the state of the model. Rules for specifying both internal and external transitions have the same format. Internal transition rule conditions test the phase and state of the model. External transition rules include tests of the input and elapsed time in their conditions.

As an example, an informal presentation of some of the rules for the Assistance-Requestor is given below:

external activity:
R1. if phase is wait-for-info
and receive x on port motion-info
then record value of x as current position
and hold-in phase active for 1 unit

internal activities:
R2. if phase is active
and need help in executing task
then send out request for help
and passivate in wait-for-help
R3. if phase is active
then send to port starting
and hold-in phase working for 100 units
and send to port finished

R4. if phase is working
then passivate

Rule R1 is an external activity which activates the model when an external event arrives on the port 'motion-info' while the model is in the phase 'wait-for-info'. Rules R2, R3, and R4 are internal activities associated with rule R1. Rules R2 and R3 provide alternative courses of action that follow once R1 has placed the model in the 'active' phase. R2 starts a sequence of activities dictating what to do if help is needed. R3 bypasses this request for help, and immediately lets the model proceed to the 'working' phase. R4 dictates what happens while the model, with or without help, has completed its 'working' phase (namely nothing, since the activity itself has not been modelled so far except for the time it takes to execute it).

Rule R3 provides an example where both before- and after-outputs are specified. The before-output is generated just before the action is evaluated while the after-output is generated at the end of the interval specified by the hold-in primitive.

The inference engine underlying forward-models evaluates the rules in the order in which they are added to the model. One advantage of employing rules is apparent in the above examples: rules, be they internal or external activities, that are closely associated can be placed contiguously. This avoids breaking sequences of external and internal transitions apart, and thus, aids model comprehension. A second benefit: since outputs may be specified within the rules, the output specification is not separated from the transition specification as necessitated otherwise.

4. Conclusions

As a theory of cognition, the above model has the following properties:

(a) Except at MPU selection points, attention is focused on only those aspects of the environment dictated by the currently activated MPU. If a recording mechanism were to be added which is sensitive only to the current activity, the system, for the most part, would only be able to recall highly restricted portions of its sensory input history (selective attention and recall).
(b) While an MPU behavior lies within its envelope, no other MPU can supplant it, even if its initialization conditions better fit the current situation (cognitive hysteresis).

One of the primary goals of the project will be to judge whether these principles provide a workable basis for intelligent robot design. For example, such robots may be so single-minded as to be incapable of flexibly responding to an unknown or changing environment. Likewise, handling of interruptions, such as requests for help (see above), must be encoded in each MPU since the selector does not inspect inputs in the closed state.

References

1. Bobrow, D.G., *Qualitative Reasoning About Physical Systems*, MIT Press, Cambridge, MA (1985).
2. Davis, P.K., Applying artificial intelligence techniques to strategic-level gaming and simulation, in *Modelling and Simulation Methodology in the Artificial Intelligence Era* (M.S. Elzas, T.I. Ören, and B.P. Zeigler, eds.), North-Holland, Amsterdam (1986).
3. Hardt, S.H., Aspects of qualitative reasoning and simulation for knowledge intensive problem solving, in *Modelling and Simulation Methodlogy: Knowledge System Paradigms* (M.S. Elzas, T.I. Ören, B.P. Zeigler, eds.), North-Holland, Amsterdam (1988).
4. Holland, J.H., Escaping brittleness: the possibilities of general-purpose learning algorithms applied to parallel rule-based systems, in *Machine Learning: An artificial Intelligence Approach*, Vol. II (R.S. Michalski, J.G. Carbonell, and T.M. Mitchel, eds.), Morgan-Kaufmann Pub. Co., Los Altos, CA (1986).
5. Klahr, P., Expressibility in ROSS, an object-oriented simulation system, in *Artificial Intelligence in Simulation* (G.C. Vansteenkiste, E.J.H. Kerckhoffs, and B.P. Zeigler, eds.), SCS Publications, San Diego, CA (1986).
6. Rajogopalan, R., The role of qualitative reasoning in simulation, in *Artificial Intelligence in Simulation* (G.C. Vansteenkiste, E.J.H. Kerckhoffs, and B.P. Zeigler, eds.), SCS Publications, San Diego, CA (1986).
7. Reddy, Y.V., Fox, M.S., and Husain, N., Automating the analysis of simulations in KBS, *Proc. SCS Multi-Conference, San Diego, CA* (1985).
8. Reddy, Y.V., Fox, M.S., Husain, N., and McRoberts, M., The knowledge-based simulation system, *IEEE Software*, March 1986, pp. 26–37.
9. Rozenblit, J.W. and Huang, Y.M., Constraint-driven generation of model structures, *Proc. Winter Simulation Conf., Atlanta, GA* (1987).
10. Rozenblit, J.W., Sevinc, S., and Zeigler, B.P., Knowledge-based design of LANs using system entity structure concepts, *Proc. Winter Simulation Conf., Washington, D.C.* (1986).
11. Rozenblit, J.W. and Zeigler, B.P., Design and modelling concepts, in *Encyclopedia of Robotics* (R. Dorf and S. Nef (eds.), Wiley, New York (1988).
12. Sevinc, S. and Zeigler, B.P., Entity structure based design methodology: a LAN protocol example, *IEEE Trans. Soft. Engr.* **14**–3, 375–383 (1988).
13. Zeigler, B.P., *Multifacetted Modelling and Discrete Event Simulation*, Academic Press, London (1984).
14. Zeigler, B.P., DEVS-Scheme: A Lips-based environment for hierarchical modular discrete event models, Tech. Rep. AIS-2, CERL Lab., Dept. of ECE, Univ. of Arizona, Tucson (1986).
15. Zeigler, B.P., Hierarchical, modular discrete event modelling in an object oriented environment, *Simulation J.*, November (1987).
16. Zeigler, B.P., Knowledge representation from Newton to Minsky and beyond, *Applied Artificial Intelligence* **1**, 87–107 (1987).
17. Zeigler, B.P., Implementation of methodology based tools in the DEVS-Scheme Environment, in *Modelling and Simulation Methodology: Knowledge System Paradigms*, (M.S. Elzas, T.I. Ören, B.P. Zeigler, eds.), North-Holland, Amsterdam (1988).
18. Zeigler, B.P., The DEVS formalism: event-based control for intelligent systems, to appear in January 1989 issue of *Proc. IEEE* (in press).
19. Zeigler, B.P. and Zhang, G., Formalization of the system entity structure knowledge representation scheme: proofs of correctness of transformations, in *AI, Simulation, and Modelling* (L. Widman and K.A. Lopard, eds.), Wiley, New York (1988).