Expert Systems With Applications, Vol. 3, pp. 329–342, 1991
Printed in the USA.

0957-4174/91 $3.00 + .00

Reducing the Validation Bottleneck with a Knowledge-Based, Distributed Simulation Environment

BERNARD P. ZEIGLER, JERZY W. ROZENBLIT, ERIC R. CHRISTENSEN

The University of Arizona, Tucson

Abstract—*This paper reviews the concepts of a theory of modelling and simulation that relate to the validation enterprise. The theory provides a vocabulary, concepts, and mathematically rigorous tools with which to tackle problems in simulation model validation. We have implemented a hierarchical, distributed, object-oriented, and knowledge-based modelling and simulation environment in Ada. The environment, DEVS-Ada, provides portability, a standard model specification language, the means to manage a model repository, the ability to reuse models, and distributed simulation. We show how DEVS-Ada can exploit the parallelism intrinsic in the multiple execution of simultaneous experiments required for model validation. Faster execution of such a computationally intensive process reduces the bottleneck that validation imposes in the model development process and enables greater confidence to be achieved in the results. We also discuss the desirability of a more global view of validation which requires parallel symbolic analysis of a newly created model relative to existing models in a model base.*

1. INTRODUCTION

SIMULATION MODEL validation entails a multidimensional set of issues (Balci & Sargent, 1984; Oren, 1984; Sargent, 1984) whose difficulty poses hard challenges for computerized support of the validation process (Oren, 1986; Sargent, 1986). Due to its difficulty, such support has greatly lagged behind the explosive growth of model development tools. The discrepancy is even more remarkable if one includes CASE tools in the latter—indeed, the situation is endemic: We can now build much more sophisticated systems than we can be sure of working. Yet simulation is expected to play an ever increasing role in system design, and the recognition has grown that the credibility of a simulation model is at the core of any recommendations or conclusions based on it. Indeed, such recognition has been formulated in requirements by government for accrediting models as part of all simulation-based tasks. So that while the issues involved are difficult, the need is urgent for greatly enhanced computerized support of validation.

This paper takes the position that computerized support of validation is achievable only on the foundation of a firm framework of formal concepts offering a standard vocabulary and definition. Such a framework need not capture all the dimensions of the problem—indeed, reality is forever beyond such formalization. However, such a framework must characterize the core of the validation enterprise and offer a set of standards to guide top-down development of computer-based support tools for this core.

We have implemented a hierarchical, distributed, object-oriented, and knowledge-based modelling and simulation environment (KBMSE) in Ada. The environment, a migration of DEVS-Scheme (Zeigler, 1990) to Ada, provides portability, a standard model specification language, the means to manage a model repository, the ability to reuse models, and distributed simulation. The DEVS (Discrete Event System Specification) and SES (System Entity Structure) formalisms provide methods for identifying the parallelism inherent in the system model which may be exploited by the mapping of model subcomponents to different physical processors. To maximize the exploitation of the identified parallelism a combination of the DEVS formalism and Time Warp (TW) mechanism is used.

In this paper, we first review the Theory of Modelling and Simulation (Zeigler, 1976) as it relates to model validation. Then we discuss the concepts and implementation of DEVS-Ada. Two kinds of validation procedures are then considered, local and global. We show how DEVS-Ada can exploit the parallelism intrinsic in the multiple execution of simultaneous experiments required for local validation. Faster execution of such a computationally intensive process reduces the bottleneck it imposes in the model development process

Supported by NSF Grant DCR 8714148 Intelligent Simulation Environments for Advanced Computer Architectures.

Requests for reprints should be sent to: Bernard P. Zeigler, AI and Simulation Group, Department of Electrical and Computer Engineering, The University of Arizona, Tucson, AZ 85721.

and/or enables greater confidence to be achieved in the results. We also show how global validation requires parallel symbolic analysis of a new model relative to the models in a model base. Parallel computing may render such validation feasible—where it is currently noticeable by its absence.

2. REVIEW OF THE THEORY OF MODELLING AND SIMULATION

Since the 1970s progress has been made on a theory of modelling and simulation (Zeigler, 1976) in which core concepts of model validity were formulated. In subsequent development, key validation concepts were clarified and formalized (Zeigler, 1984). Recently, an operational environment was completed which implements the theoretical framework and can serve as a test-bed for design and evaluation of automated validation tools (Zeigler, 1990).

The core validation concepts of the theory take off from a point similar to the informal characterization of the validation process given in the call for papers for a recent conference on simulation validation (Zeigler & Christensen, 1990):

1. identification of the variables to compared or predicted;
2. definition of the metric by which goodness of fit of prediction and reality are to be evaluated and the uncertainty of the goodness of fit;
3. identification of the domain of applicability of the simulation.

This concept of validity is in accord with the most basic level of validity identified by the theory—replicative validity, employing observable input/output behavior. In the following, we briefly summarize the concepts of the theory of relevance to validation.

The modelling and simulation enterprise concerns three basic objects (Fig. 1):

- the *real system,* in existence or proposed, which is regarded as fundamentally a source of data;
- the *model,* which is a set of instructions for generating data comparable to that observable in the real system.

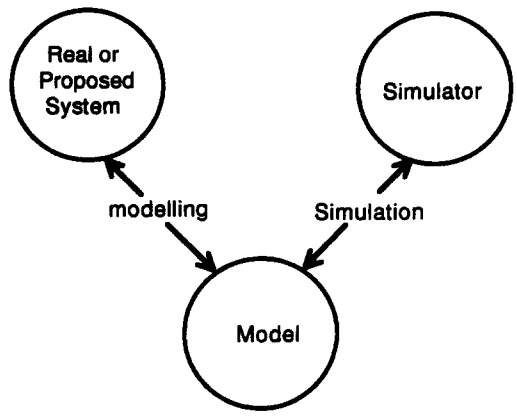

FIGURE 1. Entities and relations in simulation.

The structure of the model is its set of instructions. The behavior of the model is the set of all possible data that can be generated by faithfully executing the model instructions;

- the *simulator* which exercises the model's instructions to actually generate its behavior.

The basic objects are related by two relations:

- the *modelling relation,* linking real system and model, defines how well the model represents the system or entity being modelled. In general terms a model can be considered valid if the data generated by the model agrees with the data produced by the real system in an experimental frame of interest;
- The *simulation relation,* linking model and simulator, represents how faithfully the simulator is able to carry out the instructions of the model.

There is a crucial element which has been brought into this picture—the experimental frame. This captures how the modeller's objectives impact on model construction, experimentation, and validation.

An experimental frame specifies the limited circumstances under which a model is to be applied and/or experimented with. We separate the experimental frame from the model itself and require that the experimental frame provide a model-independent specification of the certain elements of the validation process. As rationale for the separation, consider that it should be possible to compare two models or a model and its real system counterpart under the same independently specified conditions—given by an experimental frame. As a consequence it is possible to consider the behavior of a model within more than one experimental frames.

Separation of models and experimental frames necessitates the ability to combine them together so that the behavior of a model in a frame is well defined. Prior to this we need to be able to ascertain when a model-frame combination can work: we call this the "applicability" relation. Based on mathematical systems theory, the theory of modelling and simulation provides the mathematical apparatus to characterize such applicability and the behavior of model-frame pairs where the frame is applicable to the model.

An experimental frame is given a mathematical structure as in Figure 2. It specifies:

- input variables—which will be stimulated in any model which accommodates the frame (i.e., to which the frame is applicable);
- output variables—which will be observed in a frame-applicable model;
- run control variables—which will also be observed but are there for experimentation control rather than output behavior observation;
- input segments: the allowable sequences (time segments) of inputs that will be sent to the model;
- run control segments: constraints of the combinations of run control variables (including temporal constraints) which capture the domain of operation

required by the frame. Input–output behavior of a model in this frame is accepted only so long as the run control constraints are not violated;
- summary mappings: statistical and other aggregations of the input–output behavior into reduced and manageable spaces.

The elements of an experimental frame are derived from the objectives of the modeller in relation to a frame-applicable model. For example, what input variables are we requiring that the model have? how are they to be stimulated? what output variables should the model have? what summary statistics are of interest? what is the domain of operation required? Often run-control variables in a frame are identified with state variables in a frame-applicable model, and run control constraints therefore formalize, in a model-independent manner, the "domain of validity" required of a frame-applicable model. This is because such a domain of validity specifies the operating region in which the model is valid. The frame specifies a required operating region. The applicability definition is such that a frame is applicable to a model only if the model has the requisite run control (state) variables. A model which is valid with respect to a real system in the frame (see below) will therefore match the real system's behavior in the operating domain required by the frame—this will be its domain of validity.

Using the formal definitions of Figure 2 and the underlying systems theory notions, (Zeigler, 1984) characterized the applicability relation and the I/O (input/output) behavior of a model/frame pair. Other concepts are developed for this purpose including the "derivability" partial order on frames and the "scope frame" of a model (the most inclusive frame in the derivability order applicable to a model).

Experimental frames are given concrete form as illustrated in Figure 3. A frame is realized by a system that is a coupling of three kinds of components:
- generator: generates the input segments sent to a model;
- acceptor: continually tests the run control variables for satisfaction of the given constraints (in other words, makes sure that the model is not straying from the intended operating region;
- transducer: collects the input-output data and computes the summary mappings

In the DEVS simulation environment (Zeigler, 1990) such frame components are constructed using

An Experimental Frame is a Structure

$$\langle T, I, O, C, \Omega_I, \Omega_C, SU \rangle$$

where

T	time base	
I	input variables:	which will be stimulated in any model which accommodates the frame (i.e., to which the frame is applicable).
O	output variables:	which will be observed in a frame-applicable model
C	run control variables:	which will also be observed but are there for experimentation control rather than output behavior observation
Ω_I	input segments:	the allowable sequences (time segments) of inputs that will be sent to the model
Ω_C	run control segments:	constraints of the combinations of run control variables (including temporal constraints) which capture the domain of operation required by the frame. Input-output behavior of a model in this frame is accepted only so long as the run control constraints are not violated.
SU	summary mappings:	statistical and other aggregations of the input-output behavior into reduced and manageable spaces.

FIGURE 2. Mathematical structure of experimental frame.

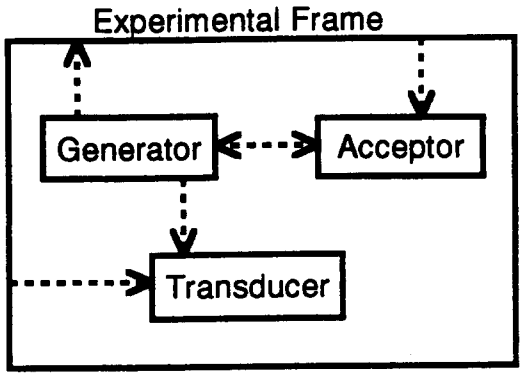

FIGURE 3. DEVS experimental frame components.

the same apparatus as are the models. This provides a powerful operational basis for maximal exploitation of the concepts.

The theory distinguishes three types of validity:

- *replicative:* I/O behavior of model in frame agrees with that of the real system in frame;
- *predictive:* model can be initialized to state corresponding with real system so that subsequent I/O behaviors agree;
- *structural:* morphism between structures of real system and model

As indicated above, replicative validity is the starting point for validation. To determine agreement of behaviors we need to specify goodness-of-fit metrics with a tolerance for judging agreement. Thus the experimental frame concept includes specification of output variables and domain of validity but goodness-of-fit criteria are considered part of a comparison process. Zeigler (1984) shows how these types of validity form a hierarchy of increasing difficulty.

In practice the complete infinite set of system and model behaviors cannot be compared. An actual test plan can obviously deal with only a finite subset. The larger this subset, the more confidence that might be had that the result of a test plan is the same as would be obtained from the infinite full complement of behavior sets. Statistical validation procedures, narrower interpretations of the above, provide operational measures of such confidence measures.

This computationally intensive requirement for increasing validation confidence begs for high performance computer architectures. In the sequel, we discuss how distributed simulation can help.

3. CONCEPTS AND IMPLEMENTATION OF DEVS-ADA

We begin with a brief review of concepts needed for our discussion of distributed validation.

3.1. System Entity Structure Formalism

A SES represents the specific decomposition, taxonomic, and coupling knowledge for a system necessary to direct model synthesis (Rozenblit, 1985a; Zeigler, 1987). Formally, a SES is defined as a labelled tree with attached variable types that satisfies five axioms: alternating mode, uniformity, strict hierarchy, valid brothers, and attached variables. A detailed description of the axioms is given in Zeigler (1984). There are three types of nodes in an SES—entity, aspect, and specialization, which represent the three types of structural knowledge. The entity node which may have several aspects and/or specializations corresponds to a model component which represents a real world object. The aspect node (single vertical line in the labeled tree of Fig. 4) represents one decomposition, out of many possible, of an entity. The children of an aspect node are entities, distinct components of the decomposition. The specialization nodes (a double vertical arrow in the labeled tree of Fig. 4) represent ways that a general entity may be categorized into special entities. As shown in Figure 4, attached to an aspect node is a coupling scheme, and to the specialization node a selection constraint. The coupling scheme specifies external input, external output, and internal couplings of the system and its components; the selection constraint designates the 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 the simulation model.

A multiple entity is a special entity that consists of a collection of homogeneous components. These components are called a multiple decomposition of the multiple entity. The aspect of such a multiple entity is called a multiple aspect (the triple vertical line in the labeled tree of Fig. 4). The representation of such a multiple entity is as follows. A multiple entity Battalions and its component battalions are represented by Battalions, three vertical lines, and Battalion from the top down. Notice, instead of presenting all Battalions for Battalions' components, only one Battalion is

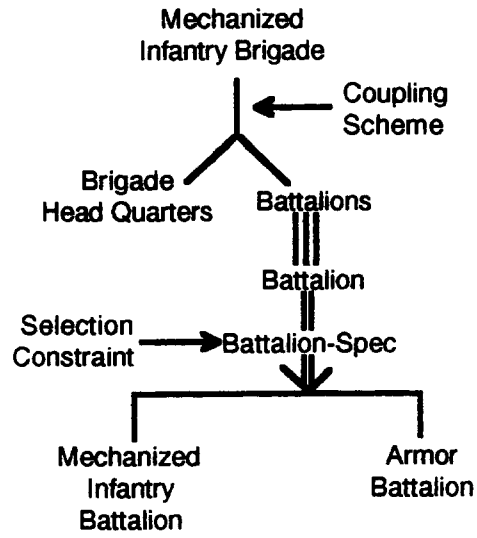

FIGURE 4. System entity structure.

placed in the labeled tree. The number of Battalions is specified by a variable, which is attached to the multiple aspect node.

Through the use of a "prune" operation, a substructure of the SES may be extracted by selecting one aspect and/or one specialization for each entity in the SES. The modeller using the "prune" operation for example, could reduce the SES shown in Figure 4 to a composition tree containing the structure information necessary to model a Mechanized Infantry Brigade. By using the selection constraint rules the appropriate mixture of Mechanized Infantry and Armor Battalions would be selected. The "transform" operation synthesizes a model hierarchically from the components in a model base. The details of DEVS-Scheme including hierarchical model structuring operations, reusability of structures, and other facilities may be found in Kim (1988).

3.2. Discrete Event System Specification Formalism

The Discrete Event System Specification (DEVS) Formalism provides methods for specifying systems in a modular and hierarchical manner (Zeigler, 1976, 1984). The specification of modular discrete event models requires the adoption of a different view than that supported by traditional simulation languages. As with modular specifications in general, the model must be viewed as possessing input and output ports through which all interactions with its environment must pass. In the discrete event case, events determine the values which appear on the ports. To be more specific, when external events arising outside the model are received on its input ports, the system specification must define how it responds to them. Additionally, internal events arising within the model change its state, as well as manifest themselves as events on the output ports to be transmitted to other model components. The DEVS formalism requires the specification of (1) basic models from which larger ones are built, and (2) how these models are to be coupled together in a hierarchical manner.

3.3. DEVS Abstract Simulators

The simulation of DEVS models is based upon the abstract simulator concepts developed as a part of the DEVS theory (Zeigler, 1984). The abstract simulator concepts are implemented by three specialized classes of processors: Simulators, Coordinators, and Root-coordinators as shown in Figure 5. The root-coordinator is the manager of the overall simulation process and is linked to the coordinator of the highest level coupled-model. Simulators and Coordinators are used to handle the atomic-models and coupled-models respectively. The simulation process is managed by passing messages between the specialized processors. The messages carry internal event, external event, and synchronization information.

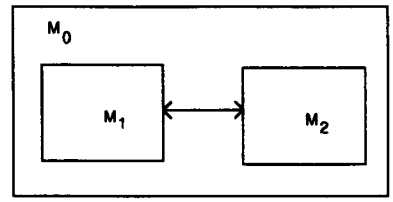

FIGURE 6. Multicomponent model structure.

We now illustrate the abstract simulator concept in more detail. Assume that a multicomponent model M_0 as presented in Figure 6 is expressed in DEVS formalism. An abstract simulator of M_0 takes the form depicted in Figure 7, where both S_1 and S_2 are abstract simulators and C_0 is a coordinator. The simulators S_1 and S_2 interpret the dynamics of model components M_1 and M_2, respectively. The coupling of S_1, S_2 and the coordinator C_0 is itself an abstract simulator that simulates model M_0. As we can see there is a one-to-one correspondence between the structure of a model and that of the simulator. We now briefly characterize the principles underlying the operation of the abstract simulator and coordinator. The reader is referred to (Zeigler, 1984, 1990) for further details.

The operation of an abstract simulator involves handling four types of messages: (*, t), (x, t), (o, t), and (y, t). In each case the right hand element is the global clock time of the simulated DEVS. When the simulator receives a (*, t) message it undergoes its internal state transition and sends a (y, t) message to its coordinator as an output. When it receives an (x, t) message, it undergoes external event-generated transition. The message (o, t) causes the simulator to send its output as a (y, t) message to its coordinator.

A coordinator carries out its task by mediating three types of messages sent to and from the parent coordinator (Fig. 8); denoted (*, t), (x, t), and (o, t), where the right hand element is the global clock time of the

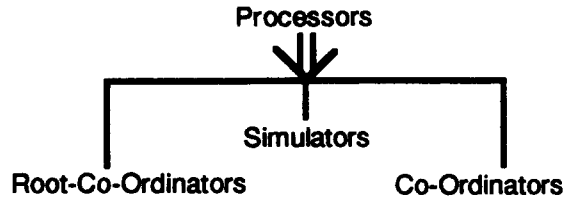

FIGURE 5. Abstract simulator classes.

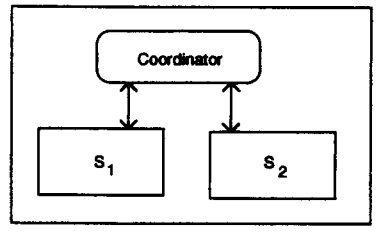

FIGURE 7. Abstract Simulator for model M_0.

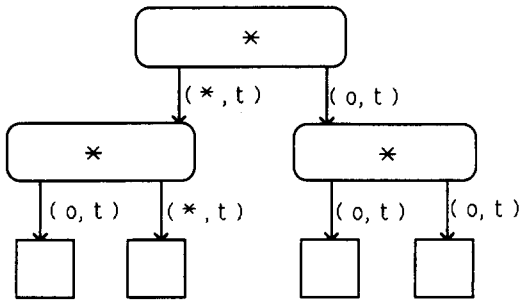

FIGURE 8. Propagation of (*, t) and (o, t) messages.

simulated DEVS. The (*, t) message indicates that the node should be activated, that is, an internal event should be executed in the DEVS at the node. When a (*, t) message is received by a coordinator, it is transmitted to the subordinate representing the imminent component DEVS. When (*, t) is received by a leaf simulator, it carries out the internal transition function of the associated DEVS. Upon receipt of a (*, t), a coordinator also transmits (o, t) messages to each of its subordinates requesting that each returns the output corresponding to its associated DEVS.

Finally, the (x, t) message indicates that an external event x is arriving at the global time t. When received by a coordinator, it consults its external-to-internal coupling table to generate appropriate (x, t) messages to the subordinates influenced by the external event. When (x, t) is received by a leaf simulator, it directly executes the external transition of the associated DEVS.

Although, an (x, t) message may originate from the environment external to the overall model, it may also be generated within the hierarchy. The latter occurs when the outputs received by an activated coordinator in response to its (o, t) request, are collected together using its internal-to-external coupling table. The resulting (y, t) message is sent to the parent coordinator for distribution as (x, t) messages to the subordinates influenced by the activated coordinator.

3.4. Time Warp

The Time Warp Mechanism is a synchronization strategy based upon the virtual time paradigm (Jefferson, 1985). An event in Time Warp is defined as a set of input messages which arrive for a specific object at a particular simulation time. Time Warp ensures that all causally linked events are eventually executed in the proper sequence. An event is considered to be causally linked if it is dependent upon the occurrence of another event earlier in time. However, events that are not causally linked may be executed in any order. If two causally linked events are initially executed out of order, the mechanism will undo all erroneous side effects and redo the events in the correct sequence. Thus, Time Warp embodies the optimistic synchronization approach relying upon process rollback as the fundamental synchronization tool for distributed simulation (Hontalas, Jefferson & Presely, 1989). Rollback is implemented using negative messages to undo the erroneous events.

3.5. Classic-Ada

The DEVS Formalism and its associated abstract simulator concepts have been implemented in Classic-Ada (Christensen, 1990). Classic-Ada provides the Ada programming language the ability to support inheritance, message passing, and dynamic binding, which in addition to the data abstraction and information hiding features already present creates a complete object-oriented programming environment (Bach, 1989; Cox, 1986; SPS, Inc., 1989; Stroustrup, 1988). Classic-Ada incorporates the semantics and syntax of Ada and adds the necessary language constructs to support dynamic binding and inheritance. The output of the Classic-Ada Processor is pure Ada which may be compiled using any validated Ada compiler.

3.6. The Distributed Simulation Environment

The DEVS formalism as mentioned earlier provides a very powerful method of decomposing large-scale systems into hierarchical and modular components and identifying parallelism inherent in the simulation models. However, the DEVS abstract simulator does not provide the means to exploit all the parallelism during simulation. The Time Warp mechanism provides means for the parallel execution of simulations but does not offer the modeller any support in decomposing a system into the required logical processes nor for identifying the parallelism in the system. By exploiting the strengths of DEVS and Time Warp an efficient, portable, knowledge-based distributed modelling and simulation environment has been implemented.

The combination of DEVS and Time Warp uses the DEVS formalism for model behavior specification, the SES formalism for model structure specification, and the abstract simulator concepts for simulation management on the physically distinct processors. The Time Warp mechanism implemented using Ada tasking is used to manage the global distributed simulation process, interprocessor communications and processor synchronization (Christensen, 1990).

4. MODEL VALIDATION IN DEVS-ADA

As indicated above, our concept of validation is a triadic relation: model M is valid with respect to real system R in experimental frame E. We shall further distinguish between local and global validation procedures. A *local validation* procedure works only with a triple M,R,E. This is the classical concept of vali-

dation as discussed by Sargent (1986) for example. A *global validation* procedure works not only with a single model, but potentially, with all the models in the model base that have been developed for the real system. It is therefore sensitive not only to the direct relationship of a model to a real system but also to its consistency with the other models that have been previously validated for, perhaps many different facets of, the real system.

4.1. Local Validation Procedures

Recall that there are three basic levels of validity: replicative, predictive, and structural. The real system R is a source of data either given by a historical data base or by a potentiality to generate such data (under experimental observation) or both. For purposes of the following discussion we replace R by a model B (for base model) that represents it. For historical data, this replacement merely supposes a means to "replay" time-indexed data records. For an existing technical system, it supposes a means by which the system can be connected to the simulation on-line so that it appears to the latter as an internal model. Nontechnical systems, such as ecosystems, that cannot be coupled on-line are excluded. With this assumption, validity concerns the holding of ternary relations:

M is (replicatively, predictively, structurally) valid wrt. B in frame E.

Testing of replicative or predictive validity involves, in principle, generation and comparison of an infinite number of input/output pairs. An actual test plan can obviously deal with only a finite subset. The larger this subset, the more confidence that might be had that the result of a test plan is truly representative of an ideal test plan based on infinite full complement of behavior sets.

The proposal in this paper is that a knowledge-based, distributed simulation environment such as DEVS-Ada, can greatly speed-up execution of a test plan and thereby ameliorate a major computation-intensive bottleneck in the model development process (the validation process is an inner loop with respect to an iterative process of successive approximation whereby a model is created and tuned to reality).

There are two parts to our claim: 1) the knowledge-based level of model specification provides user-friendly, fast and secure set up of multiple concurrent experiments, and 2) that the distributed simulation engine exploits the inherent parallelism in such set-ups to achieve significant speed-up.

The SES in Figure 9a represents a typical structure for replicative validation. Here an arbitrary number of versions of M are individually coupled to copies of E. Such versions of M might be distinguished by their

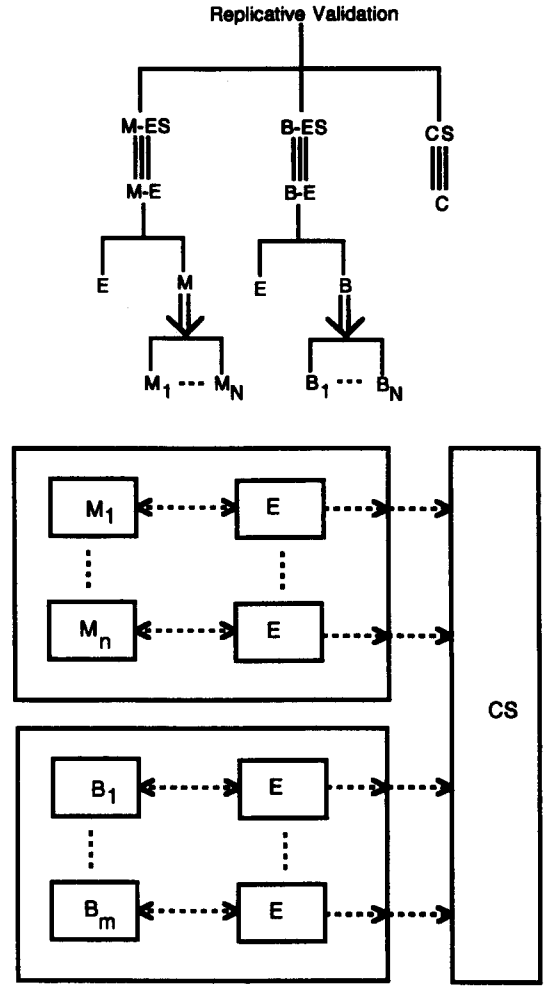

FIGURE 9. (a) Replicative validation system entity structure; (b) replicative validation-configuration.

distinct initial states, as might be the case in a stochastic process where pseudo random number generator components of M are placed into different initial states (seeds). Similarly an arbitrary number of versions of B (perhaps due to initial seed assignment) are individually coupled with copies of E. Note that copying of model objects can be performed by the environment regardless of the complexity of the original (when such models are hierarchical compositions, the copies are also compositions of copies with appropriate hierarchically structured names). The outputs of the M-ES and B-ES component models are fed to a comparator, C which does the actual computing of the goodness-of-fit and a measure of confidence in the result of the test plan (Law & Kelton, 1982).

Figure 9b portrays a simulation model in DEVS-Ada that would be created from the SES by the transform procedure. The obviously large amount of concurrency in such a model is readily exploited by the DEVS-Ada optimistic simulation approach. Indeed, all the model-frame experiments will be executed concurrently (given sufficient numbers of processors) with the comparator now being the computation bottle-

neck (it too could be decomposed so as to increase speed up).

Figure 10 provides an SES and transformed model for the case of predictive validity. Versions of the model M and base model B are paired together and receive the same input with their outputs compared for agreement. This is appropriate since for predictive validity we assume that the model can be synchronized to start in the "same" initial state as the real system. Thus each pairing of M and B represents a different initial state assignment. The outputs of the individual comparators are then combined to yield an overall measure of the goodness-of-fit. Comments similar to those made in the case of replicative validity relating to potential speed-up are applicable here as well.

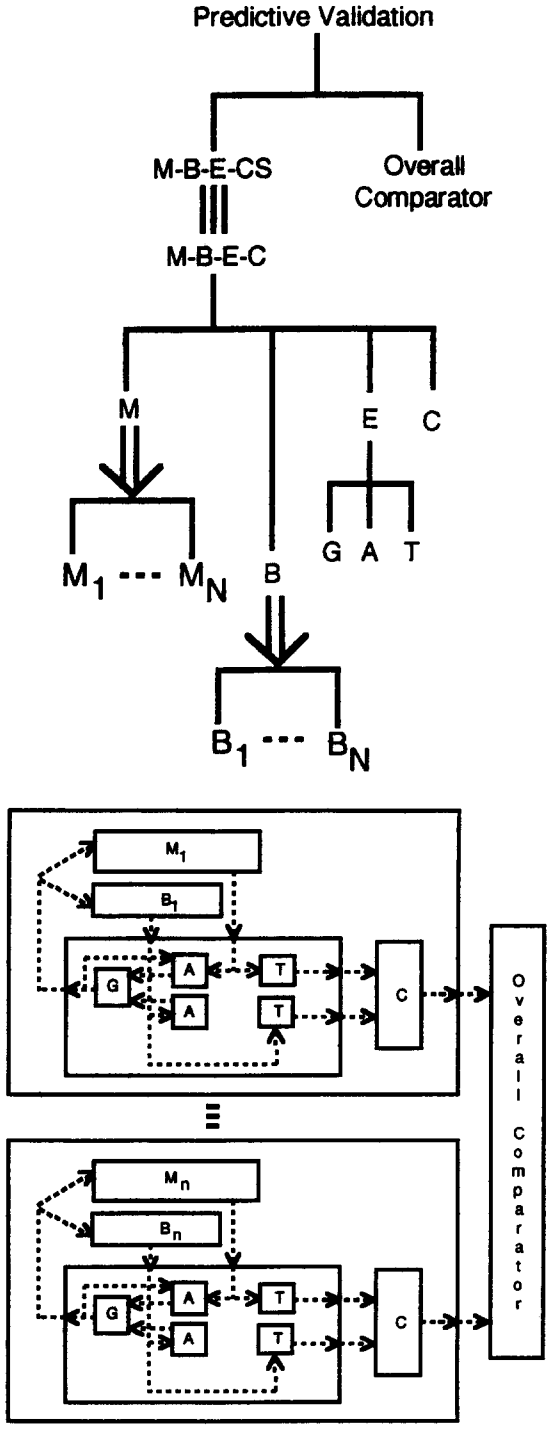

FIGURE 10. (a) Predictive validation system entity structure; (b) predictive validation simulation configuration.

4.2. Local Validation Experimental Results

To illustrate the potential speed-up that may be achieved when using DEVS-ADA, we will use the replicative validation of a model of a single server without queue (Fig. 11). The job interarrival times are normally distributed, as is the service time. Each local transducer monitors for each job, the arrival and departure times, the number of new jobs, the number of jobs solved, and the number of jobs lost. This transducer calculates the average of the turn-around times, the standard deviation of the turn-around times, and the server throughout at user specified observation intervals. The global transducer receives the averaged turn-around times, and throughputs from the local transducers. The global transducer then estimates the mean turn-around time, mean throughput, and confidence intervals for these statistics after receiving all of the local transducer outputs at the end of each observation interval.

The single server model/experimental frame pair and their associated abstract simulators are encapsulated in an Ada task type. The use of an Ada task type allows the creation of an arbitrary number of task copies. The copies will all have the same structure but separate and distinct states. Since a single copy of the global transducer is needed, the global transducer and its associated abstract simulators are encapsulated by an Ada task, instead of an Ada task type. It should be noted that the assignment of models to Ada tasks parallels the assignment of models to physical processors such that all Ada model tasks could be executed in parallel given a number of processors equal to the number of tasks. This ability to assign models to tasks/processors facilitates the exploitation of the external and internal event parallelism identified by the DEVS formalism. The assignment of models to tasks/processors is dependent upon the system decomposition and the interaction between the models.

The potential for speed-up was measured by executing a number of configurations of size N where N ranged from 1 to 80. This was done on a SUN 3/60 workstation having an Ada compiler. The results (Figs. 12a and 12b) indicate that the amount of overhead to execute N model pairs is approximately linear in N. Due the experimental setup (Fig. 11) the linear increase (Fig. 12a) shows that the distributed implementation of DEVS-Ada is exploiting the noninteraction of the

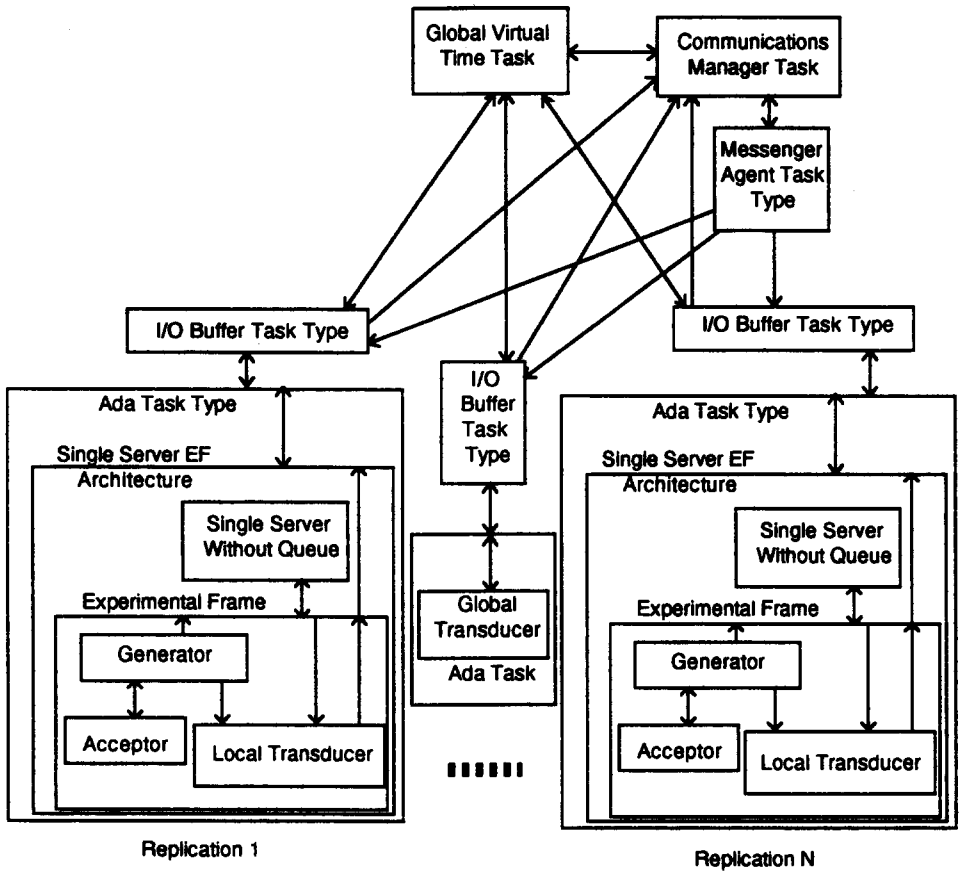

FIGURE 11. Replicative validation of single server without queue.

model-frame pairs. In this linear range, the execution of N models on N processors should take the same time as executing a single model on a single processor (this assumes the tasking overhead of the multiprocessor environment is the same as that for a single processor). The upward knee in the curve represents the bottleneck due to sequentiallism in the global transducer which starts to take effect as the number of feeder inputs increase. To defer the onset of the upward knee, the global transducer can be decomposed into a network of subcalculation transducers (for more detail, see Christensen, 1990).

5. A FRAMEWORK FOR HIERARCHICAL, DISTRIBUTED VALIDATION

This section extends the definition of experimental frame and discusses a framework for frame specification in a distributed DEVS simulation environment. So far, we have coupled models and experimental frames only at the highest level, that is, without taking into account the hierarchical structure of the model. Now, we present an approach to distributing an experimental frame within a hierarchical model. This means that we can attach frames to both atomic and coupled subcomponents of a hierarchical model. Distributing frames in this manner offers a means of further exploiting parallelism and modularity in distributed simulation (Rozenblit, 1985b). The DEVS abstract simulator is a basis of our considerations. We assume that an experimental frame is expressed in the DEVS form and we define model/frame coupling mechanisms for support of atomic and coupled model validation. Implementation of these concepts in DEVS-Ada has not yet been attempted. However, a prototype of has been built in DEVS-Scheme (Duh, 1988).

5.1. Hierarchical Specification of Experimental Frames

To illustrate the ensuing discussion, let us assume that a model consist of two atomic submodels as illustrated in Figure 6. The abstract simulator for such a model has the structure depicted in Figure 7. We now describe how an experimental frame module can be synthesized and coupled with the model's abstract simulator. The basic experimental frame/model coupling results in the architecture depicted in Figure 13. We now proceed to specify how such a coupling is defined.

Recall from the definition of the experimental frame realization that each component of the system S_E (Fig. 13) (i.e., generator, acceptor, and transducer) is a DEVS

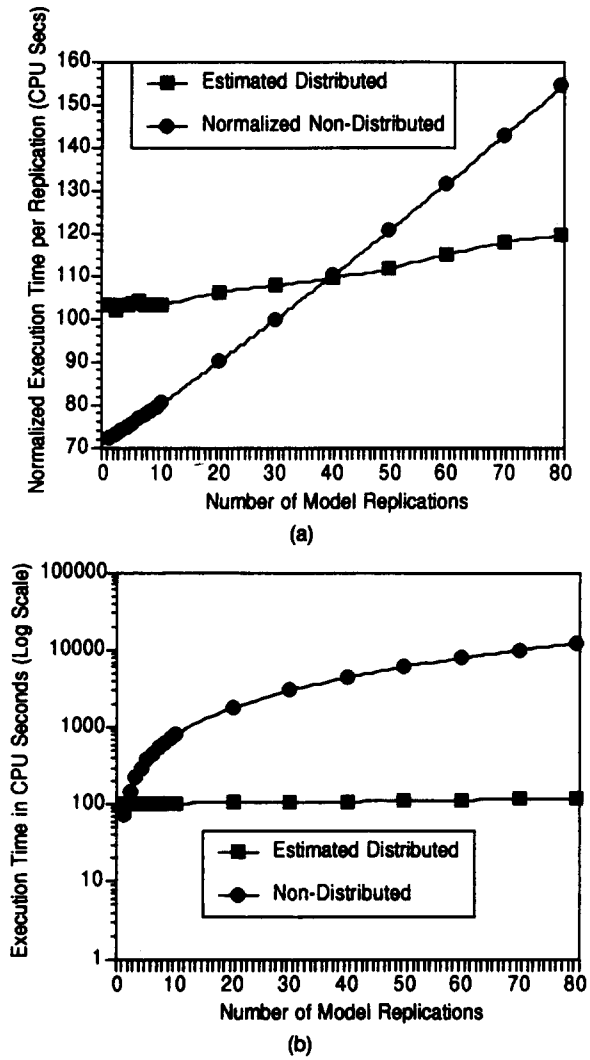

FIGURE 12. (a) Comparison of average model execution times in the non-distributed and distributed environments of DEVS-Ada; (b) Distributed validation speed-up potential.

model and thus may be realized as a hierarchical coupling of systems. At this point several alternatives for experimentation control arise. In the centralized architecture as illustrated in Figure 13, control is concentrated within the master experimental frame module S_E whereas the simulators S_1, S_2 are responsible for execution of model component dynamics.

The coupling of the frame module S_E and the abstract model simulator S is defined as follows: the generator S_G originates the messages (x,t) that are received by the root coordinator C_0 as external events to the model. The output statistics are gathered by collecting the (y, t) message from the root coordinator. This message defines an input signal to the frame transducer S_T. It carries the information about changes of output variables in each subordinate DEVS model simulator.

The realization of experimentation control requires that the coordinator of each abstract simulator be extended as follows: upon receipt of a (*, t) or (x, t) signal, a coordinator transmits (m, t) messages to its subordinates requesting that each return the message (c, t) corresponding to a change (if any) of control variables' values of an associated DEVS model component. The global message (c, t) is collected by the root coordinator and processed by the frame acceptor S_A which determines whether or not the run control segments lie within the admissible range.

Such a centralized architecture involves a single experimental frame module directly linked to the global coordinator. One possible manner in which this mode of experimentation in a distributed simulation environment can be realized and executed is to use the blackboard framework (Hayes-Roth, 1985; Nii, 1986). The experimental frame module would be a blackboard structure containing objects (data) from the solution space (simulators). These objects could be hierarchically organized into levels of analysis, for example, input data, partial solutions and final solutions. Thus, the function of the blackboard would be to hold computational and solution-state data needed by and produced by the knowledge sources, that is, models.

In the blackboard framework, the domain knowledge needed to solve a problem is partitioned into Knowledge Sources (KSs) that are kept separate and independent. In our methodology, each model can be interpreted as a knowledge source. The KSs respond to changes on the blackboard. A set of control modules are used to monitor the changes on the blackboard and decide which actions to take next. This could be accomplished by the coordinator modules of DEVS simulators.

Although the blackboard framework provides a set of concepts for carrying out a simulation experiment in a distributed environment, the realization of the components S_G, S_A and S_T might prove very complicated due to the complexity of the functions they execute. We propose that the components of experimental frames be distributed in a manner that corresponds to the hierarchical, distributed structure of models they are applicable to. This requirement has been stipulated in the literature by Dekker (1984) (the concept of a cosystem), Oren (1984) (GEST implementation of local frame segments), and Biles (1985) (distributed evaluation of a network of microprocessors).

In order to specify a distributed experimental frame, we first establish the scheme for its top-down decomposition. First, we consider the input generation process. Assume that at any given level of the model composition tree (hierarchy of model decomposition) a model M_i has constituent models $M_{i,1}, \ldots, M_{i,k}$. In the centralized mode of experimentation, a generator for this model, G_i has to be defined and coupled to M_i through its input ports. In order to realize G_i as a coupling of subcomponent—possibly less complex—generators, we have to identify the structure of the input

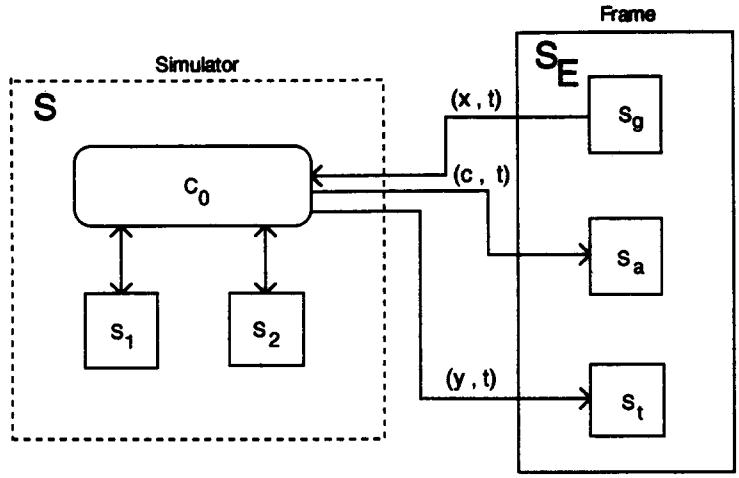

FIGURE 13. Centralized experimentation in distributed simulation.

segments received by the model M_i. In the most general case, we can assume that an input segment is decomposed into mutually independent segments $\omega_{i,1}, \ldots, \omega_{i,k}$ that are applied directly to model components $M_{i,1}$ through $M_{i,k}$ and the segment $\omega_{i,0}$ which accounts for input to their coupling, that is, M_i. In other words G_i generates segments $\omega = (\omega_{i,0}, \omega_{i,1}, \ldots, \omega_{i,k})$. We decompose G_i into generators $G_{i,0}, G_{i,1}, \ldots, G_{i,k}$, and couple them with their respective model simulators. The coupling is accomplished by a parallel composition of DEVS-specified models that realize the generators $G_{i,0}, G_{i,1}, \ldots, G_{i,k}$. The parallel coupling of component generators is a DEVS in a modular form in which no component influences another component.

Notice that any model component may itself be composed of submodels. Then, its corresponding generator is decomposed in the manner described above. Such a process is carried out recursively down to the leaf nodes of the model composition tree.

The decomposition process of the output transducer is similar to that of the input generator. The transducer T_i collects global output segments $\rho = (\rho_{i,0}, \rho_{i,1}, \ldots, \rho_{i,k})$ where $\rho_{i,0}$ may represent correlated output of the components $M_{i,1}, \ldots, M_{i,k}$ while $\rho_{i,1}, \ldots, \rho_{i,k}$, are mutually independent, local output segments. We carry out the decomposition of the output transducer as follows: T_i is decomposed into $T_{i,0}, T_{i,1}, \ldots, T_{i,k}$ that are coupled to their respective model components $M_i, M_{i,1}, \ldots, M_{i,k}$.

Notice that the above specification establishes observation frames at any two subsequent levels of the system composition tree and that the process of associating transducers with model components can be carried out recursively down to the leaf nodes of the tree.

The run control acceptor A for the model M_i is decomposed in exactly the same way as the output transducer. The component acceptors $A_{i,0}, A_{i,k}, \ldots, A_{i,k}$ monitor the run control trajectories $m_{i,0}, m_{i,1}, \ldots, m_{i,k}$, respectively. Conceptually, $A_{i,0}$ checks for acceptance of the global run control segment pertaining to M_i while the components acceptors monitor the control segments local to $M_{i,k}, \ldots, M_{i,k}$. Once again this establishes the specification framework for any two subsequent levels of the composition tree and this process is recursive with respect to the number of levels in the tree. The hierarchical specification of the run control acceptor is analogous to the specification of the transducer.

We now proceed to describe how an experimental frame is mapped onto a distributed architecture of a DEVS simulator.

5.2. Mapping Hierarchical Specification of Experimental Frames onto the DEVS Abstract Simulator

The design of a methodology for mapping the decentralized frame specification onto the corresponding abstract distributed simulator should satisfy the following requirements:

- The coupling of the simulator and frame must be closed, that is, it must result in an abstract simulator.
- The degree of decentralization of experimentation should be maximal. In other words, a means of assigning an experimental frame local to each model component should be provided.

Motivated by the above guidelines, we suggest the following procedure for establishing the frame/abstract simulator mapping: At the level (i) of the model composition tree, a DEVS simulator of a model component must simulate the model with a pertinent experimental frame. Recall that the frame components are defined as DEVS systems and thus can be realized by an abstract simulator as well. However, coordination is required between the simulators of the model, generator,

acceptor and transducer. To assure such coordination, we introduce a model/frame coupler (MFC).

An MFC is a coordinator defined in the same way as a coordinator of the abstract simulator. It performs the following functions: At the level local to its frame and model, the MFC invokes the frame generator which in turn send its output back to the coupler. This message is forwarded by the coupler to the model's simulator. The simulator interprets the message as an external event. Output messages generated by the simulator are forwarded by the MFC to the local frame transducer. Control messages (c, t) are sent to the acceptor. The coupler also serves as a communication port with the parent coordinator specified at level ($i - 1$) of the simulator hierarchy. Its function as an i/o port consists in transducing the (*, t), (x, t), (o, t) and (m, t) signals to (from) the parent coordinator from (to) the simulator of the model component at the subordinate level. A detailed description of the MFC coupler concept is given in the Appendix. A comprehensive example of a simple operating system with a distributed experimental frame, implemented in DEVS-Scheme, is presented in Duh (1988).

To ensure consistency in attaching a frame to a simulator we verify that the frame-simulator/model-simulator coupling relations are valid. More specifically, the input segments produced by the generator must apply to the input ports of the model, the output ports of the model must match the input ports of the transducer, and the variables monitored by the acceptor must match those designated as the run control variables. The MFC module checks if the above requirements are met.

The proposed mapping results in an abstract simulator that simulates the combined model/frame DEVS. Since each experimental frame module is a special form of a DEVS simulator, that is, a DEVS-generator, acceptor, and transducer, the coupling of frame and model simulators results in a DEVS simulator. The correctness of simulation is then ensured by the correctness of the DEVS simulator (for a formal proof of DEVS simulator correctness see Zeigler 1984).

Since a means for coupling of an experimental frame to a model component at any level of the hierarchy are provided, it is apparent that the maximum decentralization of the experimentation can be achieved. In future research, we intend to employ DEVS-Ada to study the utility of framework just presented for distributed simulation.

6. GLOBAL VALIDATION PROCEDURES

Local validation procedures may be severely limited in their application. For example, historical data does not exist in system design where the real system does not yet exist and cannot serve as validation standard. More generally, local validation is of limited potential where access to real system data is restricted. In such cases, local validation (to the extent it is possible) should be supplemented with consistency checking of the new model with relevant models in the model base. For example, assumptions made in constructing the new model should not contradict those made in constructing other existing models. Also, while a system under design does not yet exist, many of the components from it is built may exist and have validated models in the model base. Such comparison of features of one model simultaneously with a (potentially, large) number of others, is once again a situation that lends itself to parallel processing. To our knowledge, there is a nonexistent literature on such global validation procedures. We believe that they may become feasible within an environment such as DEVS-Ada, and that they would therefore seem to warrant increased attention.

7. CONCLUSION

The theory of modelling and simulation provides a vocabulary, concepts, and mathematically rigorous tools with which to tackle problems in simulation model validation. Accepted as a standard, such a framework would greatly facilitate the design of much needed computer-based environments to support the validation process. The distributed knowledge-based modelling and simulation environment, DEVS-Ada, provides portability, a standard model specification language, the means to manage a model repository, the ability to reuse models, and distributed simulation. DEVS-Ada combines the DEVS Formalism and its associated abstract simulators with the Time Warp mechanism. In particular, this article shows how DEVS-Ada can exploit the external and internal event parallelism intrinsic in the multiple execution of simultaneous experiments required for local model validation. Faster execution of such a computationally intensive process reduces the bottleneck it imposes in the model development process and/or enables greater confidence to be achieved in the results. The article also presents two areas in need of further research: distributed experimental frames and global validation. Distributing frames among hierarchical model components offers a means of exploiting further parallelism and modularity in distributed simulation. We also showed how global validation requires parallel symbolic analysis of a new model relative to the models in a model base. Parallel computing may render such validation feasible—where it is currently noticeable by its absence. The DEVS-Ada environment appears to provide a useful platform to study these issues.

REFERENCES

Bach, W.W. (1989). RADA: An object-oriented language. *Journal of PASCAL, ADA and MODULA-2*, March/April, 19–25.

Balci, O. & Sargent, R.G. (1984). Bibliography on the credibility, assessment and validation of simulation and mathematical models. *Simuletter*, **15**(3), 15–27.

Biles, W., Daniels, C. M., & O'Donnell, T. J. (1985). Statistical considerations in simulation of a network of microcomputers. *Proceedings of the* 1985 *Winter Simulation Conference* (pp. 388–393).

Christensen, E.R. (1990). *Hierarchal optimistic distributed simulation: Combining DEVS and Time Warp.* Doctoral Dissertation, University of Arizona, Tucson, AZ.

Christensen, E.R. & Zeigler, B.P. (1990). Hierarchical, distributed, object-oriented, and knowledge-based modelling and simulation. 58th Military Operations Research Society Symposium, U.S. Naval Academy, Annapolis, MD.

Cox, B.J. (1986). *Object-Oriented Programming: An Evolutionary Approach.* Reading, MA: Addison-Wesley.

Dekker, L. (1984). Concepts for an advanced parallel architecture. T.I. Oren, B.P. Zeigler, & M.S. Elzas (Eds.), *Simulation and model-based methodologies: An integrative view,* (pp. 235–289). New York: Springer-Verlag.

Duh, J. (1988). *DEVS-Scheme implementation of a distributed experimental frame architecture.* Master Thesis, University of Arizona, Tucson, AZ.

Hayes-Roth, B. (1985). A blackboard architecture for control. *Artificial Intelligence* **26**(3), 251–321.

Hontalas, P., Jefferson, D., & Presely, M. (1989). Time warp operating system version 2.0. Users manual. Pasadena, CA: Jet Propulsion Laboratory.

Jefferson, D.R. (1985). Virtual time. *ACM Transactions on Programming Languages and Systems* 7(3), 404–425.

Kim, T.G. (1988). *A knowledge-based environment for hierarchical modelling and simulation.* Doctoral Dissertation, University of Arizona, Tucson, AZ.

Law, A.M. & Kelton, W.D. (1982). *Simulation modeling and analysis.* New York: McGraw-Hill.

Nii, H.P. (1986). Blackboard systems: The blackboard model of problem solving and the evolution of blackboard architecture. *AI Magazine,* 7(2).

Oren, T.I. (1984). GEST—A modelling and simulation language based on system theoretic concepts. In T.I. Oren, B.P. Zeigler, M.S. Elzas (Eds.), *Simulation and model-based methodologies: An integrative view,* (pp. 281–336). New York: Springer-Verlag.

Oren, T.I. (1984). Quality assurance in modelling and simulation: A taxonomy, S. In T.I. Oren, B.P. Zeigler, & M.S. Elzas (Eds.), *Simulation and model-based methodologies: An integrative view,* (pp. 477–517). New York: Springer-Verlag.

Oren, T.I. (1986). Artificial intelligence in quality assurance of simulation studies, S. In M.S. Elzas, T.I. Oren, & B.P. Zeigler (Eds.), *Modelling and simulation methodology in the artificial intelligence era,* (pp. 267–278). Amsterdam: North-Holland.

Rozenblit, J.W. (1985a). *A conceptual basis for integrated, model-based system design.* Doctoral Dissertation, Department of Computer Science, Wayne State University, Detroit, MI.

Rozenblit, J.W. (1985b January). Experimental frames for distributed simulation architectures. In *Proceedings of the* 1985 *SCS Multiconference, Distributed Simulation,* San Francisco, CA.

Rozenblit, J.W. & Zeigler, B.P. (1985 December). Concepts for knowledge based system design environments. In *Proceedings of the* 1985 *Winter Computer Simulation Conference,* San Diego, CA.

Sargent, R.G. (1984). Simulation and model validation. In T.I. Oren, B.P. Zeigler, & M.S. Elzas (Eds.), *Simulation and model-based methodologies: An integrative view* (pp. 537–555). New York: Springer-Verlag.

Sargent, R.G. (1986). An exploration of possibilities for expert aids in model validation. In M.S. Elzas, T.I. Oren, & B.P. Zeigler (Eds.), *Modelling and simulation in the artificial intelligence Era,* (pp. 279–298). Amsterdam: North-Holland.

Software Productivity Solutions, Inc. (1989). *Classic-Ada User Manual.* Indialantic, FL: Author.

Stroustrup, B. (1988). What is object-oriented programming? *IEEE Software* **5**(3), 10–20.

Zeigler, B.P. (1976). *Theory of modelling and simulation.* New York. Wiley.

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

Zeigler, B.P. (1987). Hierarchical, modular discrete-event modelling in an object-oriented environment. *Simulation,* **49**(5), 219–230.

Zeigler, B.P. (1990). *Object-oriented simulation with hierarchical, modular models: Intelligent agents and endomorphic systems.* New York: Academic Press.

Zeigler, B.P. & Christensen E.R. (1990). A formal framework as a standard for simulation model validation. Military Operations Research Society Simulation Validation Mini-Symposium, Albuquerque, NM.

APPENDIX: MAPPING DISTRIBUTED EXPERIMENTAL FRAME ONTO DEVS SIMULATOR ARCHITECTURE

This appendix explains the details of the frame mapping discussed in Section 5.2.

An MFC is a coordinator defined in the same way as a coordinator of the abstract simulator, which performs the following functions: at the level local to its frame and model (i.e., the level (i), it sends the (*, t) message to the frame generator. This message results in an internal transition of the generator and a message (y, t) being output by the generator. This (y, t) message is sent back to the model/frame coupler and forwarded directly as an external event (x, t) to the simulator of the model component. The MFC also forwards a local (y, t) message generated by the model simulator, to the local frame transducer and a (c, t) message to the local acceptor, respectively. Notice that both, the transducer and acceptor are passive DEVS systems. This significantly simplifies the design of the MFC since it only has to schedule the internal transitions of one active component, that is, the generator. The coupler also serves as a communication port with the parent coordinator specified at level ($i - 1$) of the

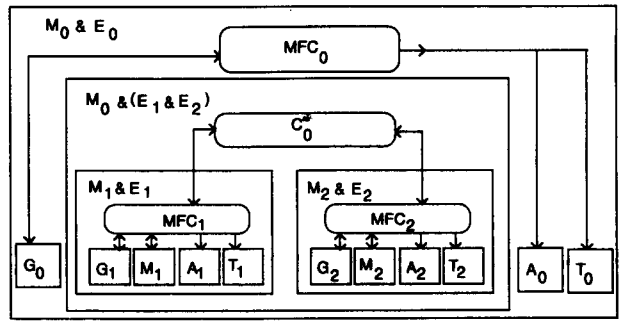

FIGURE 14. Abstract simulator of distributed model with experimental frames.

simulator hierarchy. Its function as an i/o port consists in transducing the (*, t), (x, t), (o, t), and (m, t) signals to (from) the parent coordinator from (to) the simulator of the model component at the subordinate level.

To exemplify the discussion let us consider the simulator presented in Figure 7. (Fig. 14 shows a fully distributed frame structure coupled with this simulator). The coupler MFC_1 coordinates the simulator of the model component M_1 and corresponding simulators of G_1, T_1 and A_1. It broadcasts message (*, t) to the generator which responds by producing an output signal (y, t). This output signal is in turn transduced by MFC_1 to the simulator M_1. The coupler collects the messages (y, t) and (c, t) from M_1 and transduces them to T_1 and A_1, respectively. The composition of MFC_1, M_1, G_1, T_1 and A_1 constitutes the simulator for the component M_1 with its corresponding experimental frame E_1, denoted as M_1E_1. The simulator M_2E_2 is realized in the same manner. Both simulators are coupled by the standard (in the sense of Zeigler's definition) coordinator C_0. The role of MFC's in the coupling is restricted to serving as input/output ports to the combined model/frame simulators. They simply transduce the messages between C_0 and M_1 and M_2. Notice, however, that in spite of the fact that C_0 is the root coordinator it is still necessary to simulate the model M_0 and its frame E_0. To achieve this, MFC_0 is created to coordinate the actions of the simulators M_0, G_0, T_0 and A_0. This model/frame coupler is linked to the root coordinator. The experimental frame/model coupling resulting at this level is M_0E_0 as shown in Figure 14. The model/frame coupler transmits the generator's outputs as external event messages to the coordinator C_0. It also receives the global (y, t) output and (c, t) control messages. These messages are sent to the transducer and acceptor, respectively.