DEVSJAVA: Basis for a DEVS-based Collaborative M&S Environment

Hessam S. Sarjoughian & Bernard P. Zeigler
AI & Simulation Research Group
Electrical & Computer Engineering Department
University of Arizona, Tucson, AZ, USA
{hessam | zeigler}@ece.arizona.edu
http://www-ais.ece.arizona.edu

ABSTRACT

A web-enabled DEVS modeling and simulation environment (DEVSJAVA) based on the DEVS formalism and implemented in Java language is discussed. This first generation of DEVSJAVA is an amalgamation of the proven object-oriented DEVS and key web-centric capabilities offered by the Java language. DEVSJAVA offers its users the ability to construct and experiment with dynamic models based on web technology. This modeling and simulation tool provides a graphically oriented interface through which users can learn and study the fundamentals of discrete-event modeling and simulation. It facilitates model development and simulation independent of hardware platform using Java-enabled browsers. Furthermore, it provides the foundation to enable collaborative modeling and simulation.

Keywords: *Collaboration, Concurrent, Distributed, DEVS, Hierarchical, Interactive Simulation, Internet, Java, Modeling, Modular, Multimedia, Object Orientation, Simulation, System Theory, Web-enabled*

1. INTRODUCTION

Applications of Modeling and Simulation (M&S) span numerous facets of science, technology, and business. An increasing number of government, corporate and private business decisions are based on simulation predictions and analysis. The discipline of modeling and simulation is well positioned to develop a genre of generic, yet domain-tailorable, environments that exploit the fertile and unexplored facets of the Internet and World-Wide-Web.

The Internet and World-Wide-Web are enabling a new breed of systems far surpassing most of their predecessors in terms of *use*, *complexity*, and *scale*. While the use of such systems is primarily due to the Web, the complexity and scale are not since it is the Internet capabilities that deal with these issues. From this vantage point, the role of M&S can be viewed from the intertwined perspectives of the Internet and Web in terms of their capabilities and limitations.

In the future, not only will there be demands for more powerful M&S tools, but also the extent of their use is expected to grow. The users and beneficiaries of modeling and simulation are expected to multiply many fold due to the ever increasing complexity and interdependencies of present and future systems and especially those that are distributed. The implication for these systems is that single-user M&S tools will have to be supplemented with those supporting collaborative engagement of individuals ranging from end-users and subject matter experts to M&S professionals. Therefore, it should not be surprising to the M&S community that it will need to provide web-enabled tools.

DEVSJAVA, an environment based on DEVS (Discrete Event System Specification), is the most recent of a lineage of earlier implementations of DEVS. The M&S framework is, in part, based on system theory and object-oriented

software. However, since it is web-enabled, it also relies on web-centric features. System Theory provides mathematical underpinning and Object-Orientation provides sound implementation practices and ensures computational integrity. The discipline of M&S has introduced fundamental concepts such as experimental frame, model-base, System Entity Structure (SES), model simplification, model validation, simulation verification, and endomorphism. System theory has introduced subsystem, system, decomposition, controllability, observability, and homomorphism among others. Object-orientation has introduced the concepts of abstraction, encapsulation, modularity, hierarchy, typing, concurrency, and persistence. Distributed collaboration, utilizing features such as code platform independence (write-once/run-anywhere), browser-enabled (applets), security, and multi-threaded processing are what the Internet and Web support.

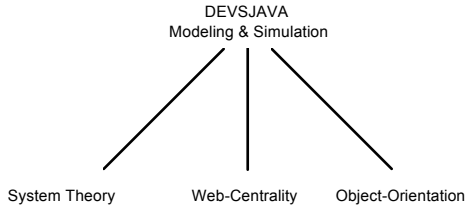

Figure 1: DEVSJAVA Modeling and Simulation Framework

The amalgamation of system theory, web-centrality, and object-orientation engineering (see Figure 1) provide a solid foundation for modeling and simulation spanning a wide range of areas. Being web-centric liberates M&S environments from being single-user and single-platform in a uniform manner. The DEVSJAVA environment is built based on DEVS, Object-Orientation, and the Web/Internet. In the remainder, we present the basis for a synergistic environment, DEVSJAVA that can support distributed collaborative modeling and simulation.

2. DEVS Foundation

The DEVS modeling approach supports capturing a system's structure from both functional and physical points of view. A DEVS model can be either an *atomic* model or a *coupled* model (Zeigler 1984). The latter models are hierarchical with well-defined characteristic. While a part of a system can be represented as an atomic model with well-defined interfaces, a system represented as a DEVS coupled model designates how (less complex) systems can be coupled together and how they interact with each other. Given atomic models, DEVS coupled models can be formed in a straightforward manner. Atomic and coupled models can be simulated using sequential computation and/or various forms of parallelism (Zeigler 1990). A DEVS model is defined as (Zeigler 1984):

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

- X set of *internal input events*;
- S set of *sequential state*;
- Y set of *external output events*;
- δ_{ext} *external transition function* specifying state transitions due to external events;
- δ_{int} *internal transition function* specifying state transitions due to internal events;
- λ *output function* generating external events as output;
- ta *time advance* function.

The sequential and parallel views play a central role in modeling and simulation of coupled models since each couple model is essentially comprised of multiple atomic models. Two different formalisms have been introduced. The sequential formalism (Zeigler 1984) treats components' simultaneous transitions sequentially, while a more recent formulation (Chow 1996) treats them concurrently enabling full parallelism.

3. DEVSJAVA

Using the Java programming language, the basic DEVS constructs have been implemented in an environment called DEVSJAVA. This environment provides the foundation upon which higher constructs of DEVS (e.g., endomorphism and variable structure) can be created using the basic, as well as the internet-based features, of the Java programming. The DEVS atomic and coupled models can be developed and simulated both as standalone applications and applets. The environment offers features that make modeling and simulation more useful, powerful, and attractive. In addition to being able to execute a simulation model anywhere from a browser, DEVSJAVA provides capabilities that are unique to DEVS modeling and simulation.

3.1 User's View

It is common practice to construct larger models from smaller ones — that is, build atomic models and synthesize coupled models from them. In the case of atomic models in the DEVSJAVA environment, the user not only is able to control the execution of the model, but also can examine its dynamics (receiving and processing inputs, state changes, or producing outputs) visually. Figure 2 shows a snap shot of an applet Graphical User Interface (GUI) in which a processor (an atomic model) and two of its essential state variables are shown. The user can examine an atomic model state-set and outputs generated at any time during

simulation as well as viewing state changes as the models traverses its state changes (AIS 1997).

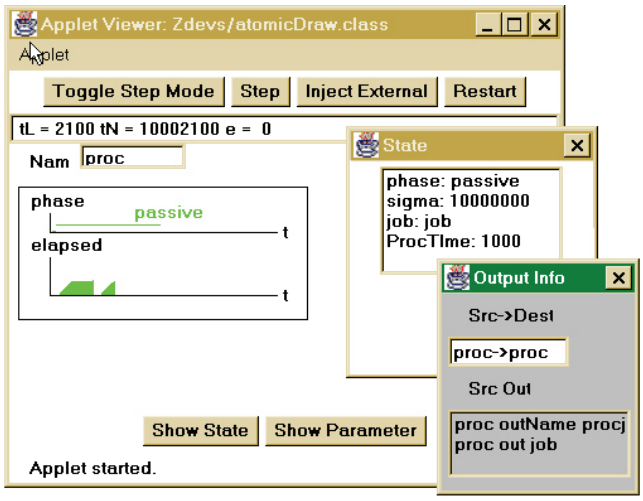

Figure 2: Processor Atomic Model Applet

The applet viewer "atomicDraw" as shown in Figure 2 contains (I) a set of buttons, (II) text fields, and (III) dialog boxes for any atomic such as "proc" which is the model for a processor without a queue. This model receives and processes jobs (input values) on its input ports. Upon completing the jobs (output values), they are sent out on output ports. The buttons are `Inject External`, `Toggle Step Mode`, `Step`, `Restart`, `quit`, `Show Parameters`, `help`, and `Show state`. The Inject External button opens a dialog box to inject an *input value* to the atomic model on a particular *input port*. This dialog box displays a list of predefined input ports with their assigned values. Selecting an entry from the list causes an input value to be sent to the atomic model's input port. The input can be sent either instantaneously or scheduled for a later time. Upon the receipt of a value on its input port, the external transition function (δ_{ext}) processes it. The Toggle Step Mode button allows the user to alternate between continuous execution and step mode. Step triggers the next internal transition (δ_{int}). Restart is used to start a new execution for a given initial state. The simulation of a model can be terminated using the Quit button. The Set Parameters button opens a dialog box to reassign parameters. The Help button displays the definition of all the other buttons.
The Show State button displays the current values of the states of the atomic model in a dialog box. For example, in the "proc" model, the dialog box says that the model is in *phase* "passive" with *sigma* set to "infinity" (1000000), name of the job being processed is "job", and time to process an incoming job is "1000". The other dialog box in Figure 2 is displayed every time an output is generated by an atomic model such as the Processor. This output dialog box contains information about the source and destination of the generated output (proc → proc) as well as the output port (out) and output value (job).

The text fields display name of the model, *tL*, *tN* , and *e*. In Figure 2, the name of a simple processor is "Proc". The last three provide the time of last event, time of next scheduled internal event, and the time elapsed in the current state, respectively. Given last event time (tL = 2100) and elapsed time (e = 0), the processor's current time is 2100. The time of next event (tN = 10002100) says that the processor is scheduled to undergo its next internal transition function at time infinity (i.e., the model has entered its passive phase and will not transition into another phase until an external input is received). The significant state variables *phase* and *elapsed* are displayed as the model dynamics change over time. The state variable *phase* depicts the current state the model is in and states changes due to the internal and external transition functions.

The interactive GUI for digraph (coupled) models offer greater capabilities as compared to atomic models since coupled models contain higher level knowledge and complexity. The user can study interactions among component models of a coupled model visually by examining outputs sent from one component model and received by others. The capabilities supported for an atomic model are also made available for coupled model due to the closure property of the DEVS formalism. Therefore, simulation of a coupled model offer not only insight into their coupling interactions but also to the dynamics of the individual atomic models used in synthesizing the former. Figure 3 depicts a digraph coupled model called "GPT" which is comprised of a Generator, a Processor, and a Transducer.

The applet viewer "devsDrawPanel" contains the text field *clock* instead of tL, tN, and e. The clock keeps track of a global time that is used to coordinate the interactions among the atomic models contained in the coupled model. This panel contains buttons as defined for the "atomicDraw" panel. In this panel, the buttons "`show state`" and "`show parameter`" are shown for each atomic model alone. The state variables and inputs/outputs can be examined as before for each atomic model. Each atomic model, just as in the "atomicDraw" panel, is identified with its own name (e.g., Transducer). This modular treatment of atomic models facilitates their individual examination. During the simulation, the output messages transmitted among components are displayed. In addition, the content of output (input) can also be displayed (again due to the modularity supported by DEVS).

Due to the web-centric features of DEVSJAVA, multimedia features such as animation and sound are possible (AIS

1997). In particular, in DEVSJAVA every atomic model can be animated as it goes through its phases. Also, sound clips can be assigned to each phase leading to multimedia simulation.

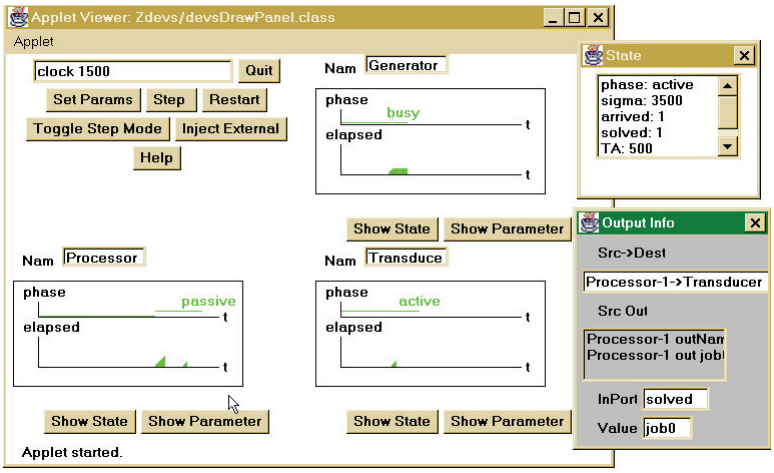

Figure 3 GPT Coupled Model Applet

3.2 *Architecture/Design*

The underlying structure for DEVSJAVA architecture shown in Figure 4 is Object Behavior Specification (Zeigler 1997b; Zeigler 1997c). The Object Behavior Specification is comprised of systems theory and object orientation principles (Zeigler 1997b). The DEVS-OBS provides a layer between DEVS set-theoretic formalism foundation and its implementation in Java (see Figure 4). Given such an architecture and OBS, an implementation of DEVS can be streamlined leading to a more robust environment. In the case of DEVSJAVA, new features of the JAVA language (JDBC and object serialization) can be incorporated into the middle layer OBS and used by the DEVS layer.

The design of DEVJAVA is based on the above architecture along with special features offered by the Java language and the Java Virtual Machine. The DEVS OBS class hierarchy has two main classes with their subclasses *container* and *devs* (see Figure 5) both derived from class *entity*. The *container* class and its subclasses provide the primitive utility classes such as *set, function* and *relation* (Zeigler 1997c). These classes provide the basic utilities to specify OBS DEVS (such as removing atomic models from a coupled model with its components stored in a set). The *devs* class, with main subclasses *atomic* and *coupled,* supports the means for DEVS modeling and simulation. There are two other subclasses of entity: *message* and *content*. The class content contains a *name* and a *value* identifying port name and value (instance of entity or any subclass) being sent from one model to anther. The class message contains instances of class content with queries (*port?* and *value?*) which return port name and value. For details of how simulation engine is implemented refer to (Zeigler 1997b).

One of the main features of Java is its support of concurrency. While Java does not support a truly multiprocessing system where each process can operate its own private address space, it does support execution of multiple processes using a shared address space. It allows concurrent execution using threads where each thread can step through its operation independently of others.

In DEVSJAVA, we have explicitly utilized threads so that each atomic model runs independently of others (due to modularity of OO-DEVS). The class devs has an instance variable which is assigned an instance of Java class, Thread. This leads to parallel execution of models in devs or any of its subclasses. In particular, the atomic class is assigned its own thread of control where it can be running either alone or as part of a coupled model. The DEVSJAVA environment utilizes this capability along with their execution inside a Java-enabled browser to enable users to experiment with any DEVS model from any machine at anytime and anyplace. The utility of executing multi-threaded simulation models from anywhere is further enhanced with GUI. A user of DEVSJAVA is able not only to run a simulation, but also interactively and visually control the simulation execution. (Note that visual control is in addition to the use of experimental frame.)

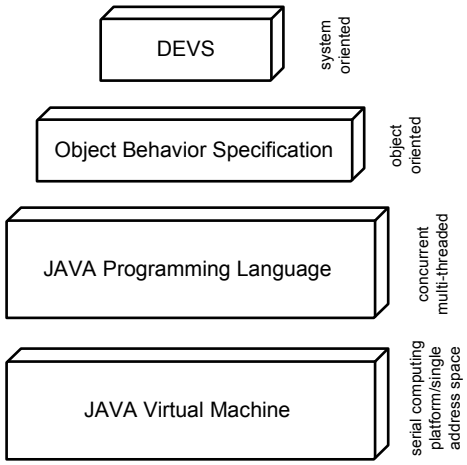

Figure 4 DEVSJAVA Implementation Architecture

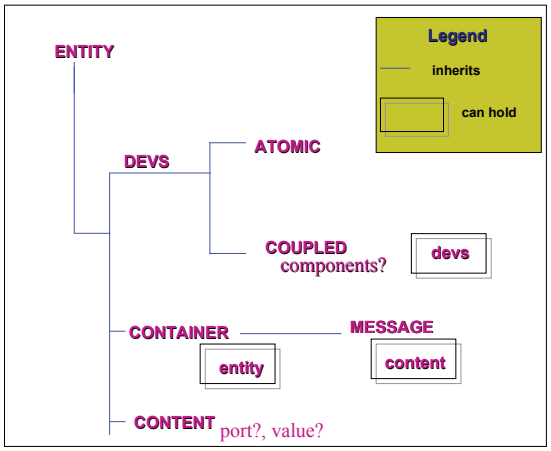

Figure 5 Simplified Class Hierarchy for DEVS Object Behavior Specification

3.3 *Model Composition*

The System Entity Structure provides a well-defined approach to *represent* a family of models depending on their decomposition and particular aspect of interest (Rozenblit 1985; Zeigler 1984). The concepts underlying System Entity Structure (SES) provide the basis for composing (synthesizing) a family of pre-built models. Given an SES, and an existing repository (model base), creating coupled models can be enhanced and made easier given an interactive GUI environment such as Graphical System Entity Structure (Au 1997). The GSES provides an interactive session (via a browser) in which hierarchical models can be synthesized by choosing atomic/coupled models and their interconnections through input/output ports coupling. Furthermore, families of models can be represented via decomposition and specialization given different aspects of a system. The user can prune an SES to select a well-defined model that satisfies the SES axioms. Such a pruned SES can then be transformed into an executable simulation model and simulated just as those created directly in the DEVSJAVA environment. The GSES provides a tree-structured representation of a system containing multiple aspects and specialization. This environment is also designed and implemented based on the Object Behavior Specification using the Java programming language .

4. Collaborative Modeling & Simulation

Earlier, we indicated the proliferation of distributed organizations that require M&S environments in support of their design, analysis, and understanding. Assuming that modeling of such entities requires the involvement of a multitude of individuals, the need for a collaborative M&S environment becomes evident. The concept of collaboration spans applications from *unstructured* to *structured* and highly controlled teamwork. We can think of a structured collaborative environment as one in which its users conform to well-defined engagement rules and protocols pertaining to the application at hand. In contrast, the unstructured collaborative environment provides a common denominator set of engagement rules that are detached from the domain and application itself.

Global operation and success of multi-organizations require teamwork not only among its own dispersed entities, but also with those of other businesses. The ability of team members to work and cooperate with each other has tremendous advantages. A team of simulation professionals, for example, can embark on the design of a system involving team members from different sub-organizations (as well as external organizations) without being completely restrained by time and space. Team members' physical locations, scarce resources, and conflicting schedules, for example, can be much less inhibiting given group-enabled technologies.

Group-enabled technologies refer to groupware that assists groups of people as opposed to single users. The functionality/limitation of these technologies depends on the interplay of *place* and *time* constraints (Grudin 1994). Each group-enabled technology provides a set of software and hardware communication tools and protocols supporting group interactions. Typical examples of these technologies

are, for example, electronic mail and meeting facilitation. We can categorize these technologies as either *distributed* or *non-distributed*. In our discussion, the term "distributed" is used when collaboration takes place across a distributed network (e.g., Wide Area Network). A *distributed collaborative* modeling and simulation environment, with its users located in physically distinct locations, would support capabilities that are specific to its users. A non-distributed collaborative M&S environment would support interaction among its users located in a single place.

4.1 Collaborative DEVSJAVA Environment

Modeling and simulation activities include simulation model formulation (construction, composition), simulation (execution), and analysis. Verification and validation, as part of model formulation, simulation, and analysis, are two very important activities that need to be supported. Given M&S activities, there exists an immense differential between the unstructured and structured collaborative environments. Using the basic model of a client getting service from a web-server (see Figure 6), the DEVSJAVA environment can support interactive model simulation, composition, and analysis by multiple users via applets given a model-base of pre-constructed and verified models.

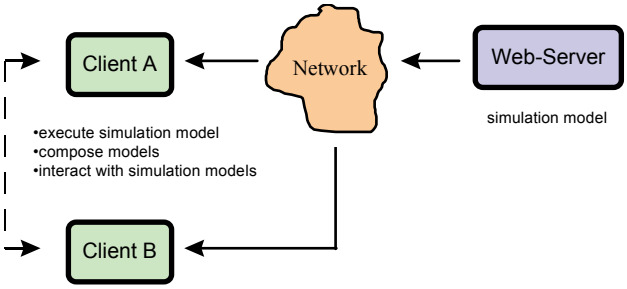

Figure 6 Client-Server Representation Supporting Modeling and Simulation

In this environment, for example, users can collaborate with each other on simulation analysis using *ad hoc* means (e.g., using a chat session and running the same model independently in their browsers). The solid arrows are shown to be unidirectional since the client would not be able to revise simulation models on the server.

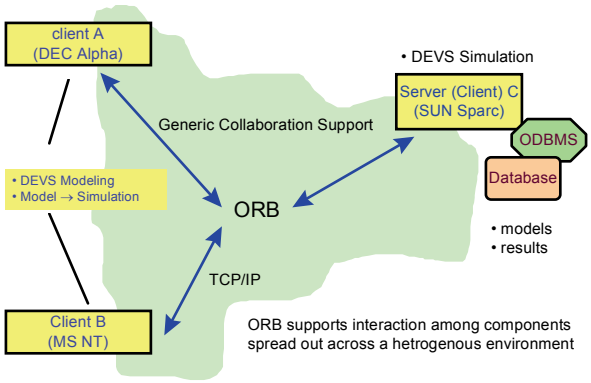

Figure 7 A Collaborative M&S Environment

However, what if a group of individuals comprised of end-users, subject matter experts, and M&S professionals needs to *construct* new models in addition to using pre-built and verified models? In this regard, it is helpful to look another collaborative M&S environment model (depicted in Figure 7). The basis for this model can be middleware such as CORBA or DCOM which would provide the necessary collaboration functionality (Evans 1997). For example, using the distributed architecture based on ORB, multiple clients (e.g., subject-matter-experts), can develop a model together and have their work stored in a database regardless of their whereabouts. The simulation can be performed on a single machine (e.g., server) and made available to the interested clients. There are numerous design issues related to how best support collaboration among dispersed users of a M&S environment. For example, what should be performed on the client versus the server need to be analyzed based on concerns such as latency, simulation type, data storage, and complexity. Given the generic DEVSJAVA framework and architecture, we believe an extended DEVSJAVA architecture, as proposed in (Zeigler 1997a) and depicted in Figure 7, will provide a suitable environment for collaborative DEVS modeling and simulation.

5. RELATED/FUTURE RESEARCH

There have been research efforts on collaborative applications from different directions . Chen and Cowie (Begole 1997) discussed Java and its role in distributed collaboration. Alexandrov et. al. have proposed an infrastructure in support of Java-Based Global Computing (Alexandrov 1996). Another technology, TANGOsim, aimed at collaborative visualization and simulation, has also been implemented (Fox 1996). This environment supports M&S as a typical application and consequently it does not take into account M&S dimensions explicitly.

The DEVSJAVA M&S environment presented is evolving. Currently, it supports some forms of variables structure

modeling as well as endomorphism. We have two ongoing efforts to extend DEVSJAVA to support collaboration. One is based on HLA/RTI paradigm and the other on DCOM. Furthermore, we are investing in what ways JAVA specific technology such as RMI and JAVA-BEANS can support collaborative M&S compared to other approaches. In a more complete collaborative DEVSJAVA environment, it is possible to support Geographic Information System (GIS) databases as well as well as object-oriented databases.

An important DEVSJAVA capability is for it to support new application areas. One area of interest for DEVSJAVA is modeling and simulation of Distributed Object Computing (DOC) (Butler 1995). Since DOC deals with large-scale systems by considering not only their dynamic structures, but also their topologies, distributed DEVSJAVA is well suited to handle the distributed nature of DOC. Many interesting systems (e.g., airline corporations) can be represented using DOC paradigm and therefore modeled and simulated by DEVSJAVA. Another interesting area is Business Process Reengineering. In BPR "to-be" processes are engineered via dynamic modeling of multiple organizations to better support business objectives (Sarjoughian 1997).

6. CONCLUSIONS

We have presented and discussed web-enabled object-oriented DEVSJAVA environment from the user's perspective. We described how users can use GUIs and web-browsers to better interact remotely with a simulation model by simply having access to the Internet. We also briefly discussed how DEVSJAVA supports model synthesis via a graphical-based SES. We argued that DEVS, Object-Orientation, System Theory, and web-centrality collectively provide a sound M&S foundation upon which to build a comprehensive, collaborative, DEVS modeling and simulation environment.

REFERENCES

AIS. (1997). "DEVSJAVA.", AI & Simulation Research Group, Electrical & Computer Engineering Dept., University of Arizona, http://www-ais.ece.arizona.edu/SOFTWARE/.

Alexandrov, A. D., et. al. (1996). "SuperWeb: Research Issues in Java-Based Global Computing.", http://www.npac.syr.edu/projects/javaforcse/javameettalks.html.

Au, V. (1997). "Multimedia Support for DEVS Modeling and Simulation in JAVA," Masters Thesis, Electrical & Computer Engineering Department, University of Arizona, Tucson, AZ., http://www-ais.ece.arizona.edu/REPORTS/.

Begole, J., Struble, C.A., Shaffer, C.A. (1997). "Leveraging Java Applets: Toward Collaboration Transparency in Java." *Internet Computing*, 1(2), 57-64.

Butler, J. M. "Quantum Modeling of Distributed Object Computing." *28th Annual Simulation Symposium*, 175-84.

Chow, A. (1996). "Parallel DEVS: A Parallel, Hierarchical, Modular Modeling Formalism and Its Distributed Simulator." *SCS Transactions on Simulation*, 13(2), 55-102.

Evans, E. a. R., D. (1997). "Using Java Applets and CORBA for Multi-User Distributed Applications." *Internet Computing*, 1(3), 43-55.

Fox, G. C. a. F., W. (1996). "Java for Parallel Computing and as a General Language for Scientific and Engineering Simulation and Modeling. http://www.npac.syr.edu/projects/javaforcse/javameettalks.html.

Grudin, J. (1994). "Computer-Supported Cooperative Work: History and Focus." *IEEE Computer*, 27(5), 19-26.

Rozenblit, J. R., Zeigler, B.P. "Concepts for Knowledge Based System Design." *Winter Simulation Conference*, San Diego, 223-231.

Sarjoughian, H. S., Vahie, S., Lee, J. "Group-Enabled DEVS Model Construction Methodology." *SPIE*, Orlando, FL, 256-267.

Zeigler. (1990). *Object-Oriented Simulation with Hierarchical, Modular Models: Intelligent Agents and Endomorphic Systems*, Academic Press, New York.

Zeigler, B. P. (1984). *Multi-Facetted Modeling and Simulation*, Academic Press, New York.

Zeigler, B. P., Sarjoughian, H.S., Vahie, S. "An Architecture For Collaborative Modeling and Simulation." *11th European Simulation Multiconference*, Istanbul, Turkey, K3-K16.

Zeigler, B. P., Sarjoughian, H.S., Au, V. "Object-Oriented DEVS." *SPIE*, Orlando, Florida, 100-111.

Zeigler, B. P. (1997c). *Objects and Systems: Principled Design with C++/Java Implementation*, Springer-Verlag, New York, NY.