Fault Tolerant Framework in MPI-based Distributed DEVS Simulation

Bin Chen, Xiao-gang Qiu and Ke-di Huang *

Abstract—**Distributed DEVS simulation plays an important role in solving complex problems for its reuseability, and composability of component models. Using MPI to be the communication middleware, the distribution increases the performance. But even the tiny faults of computing resources can lead to crash. Hence Fault Tolerant is necessary to maintain the simulation reliability. This paper introduces a DEVS framework supported Fault Tolerant. The optimistic distributed simulators implement the distribution in DEVS simulation. Fault Detection, States Storage and Fault Recovery are integrated into the framework to avoid crash at runtime. Experiments are carried out to find the optimal Timeout for Fault Tolerant framework. The results indicate that the framework has to be adjusted along with the changing of simulation requirements.** ***Keywords: DEVS, Fault Tolerance, Fault Detection, States Storage, Fault Recovery***

1 Introduction

Discrete EVent system Specification(**DEVS**) has been accepted as a sound formalism in modeling and simulation. The distributed protocol of **DEVS** has been developed to meet the requirements for solving the particularly complex problems[1]. The overall simulation time can be reduced greatly because of many nodes' participation. Many distributed frameworks are available to achieve the goal of distribution under the instructions of [2]. But even the most robust system can not maintain its reliability. So it is necessary to design a mechanism to guarantee the integration of simulation . The distribution provides the way to backup the simulation entities in case of faults.
[3] presents a roll back based optimistic fault-tolerance scheme. The stable global virtual time(SGVT) is defined to make sure the safe roll back. [4] uses the buddy processor to store the copy of tasks. If the original processor fails, the buddy process will be processed. [5] gives a Fault Tolerant framework based on HLA. The generic Fault Tolerant model is presented to solve the fault problems in HLA-based simulations. The model is optimized on the basis of system's features.
[3], [4] give the principles of Fault Tolerant, [6] adjust it to the dynamic distributed system. [5] emphasizes the framework to solve the problems in HLA-based simulation. The **DEVS** field hasn't been covered yet. In this paper, we provide a way to do the distribution of **DEVS** simulation. The distributed simulator with optimistic synchronization is introduced. Fault Detection, States Storage and Fault Recovery are discussed in detail. In section 2 we give the **DEVS** formalism. Section 3 introduces the distribution of **DEVS** simulation. Section 4 explains how to realize Fault Tolerant in distributed **DEVS** simulation. Section 5 presents the experimental data to show the improvement of distributed simulation and the optimal value for timeout. Finally, the conclusion is summarized in section 6.

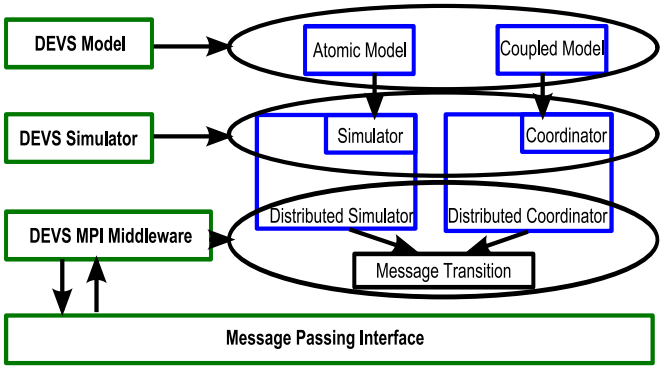

Figure 1: MPI-based Distributed DEVS Architecture

2 DEVS Formalism

The DEVS formalism provides a theoretic base of modeling the discrete event models. The formalism specifies the discrete event model in a hierarchical, modular manner. The Atomic Model, AM, is specified as follows:

$$AM = \langle X, Y, S, \delta_{int}, \delta_{ext}, \lambda, t_a \rangle \quad (1)$$

X is the set of inputs and Y is the set of outputs;
S is the set of sequential states;
$\delta_{int} : S \to S$ is the external state transition function;
$\delta_{ext} : Q \times X \to S$ is the external state transition function;
$\lambda : S \to Y$ is the output function;
$t_a : S \to \Re_0^+ \cup \infty$ is the time advance function;
$Q = (s, e) | s \in S, 0 = e = t_a(s)$ is the set of total sets.
Several atomic models could be coupled in the coupled

*System Simulation Lab, School of Mechatronics and Automation, National University of Defense Technology, Changsha, China. Email: nudtcb9372@gmail.com

model. The coupled model could also be added into a larger coupled model according to the closure of it. The hierarchy is constructed by building coupled models. The Coupled Model, CM is defined as follows:

$$CM = \langle X, Y, D, M_d, I_d, Z_{i,d}, Select \rangle \quad (2)$$

X: a set of input events and Y: a set of output events;
D: a set of component references;
M_d: a Classic DEVS model;
I_d: a set of influencees of d;
For each i in I_d
$Z_{i,d} : Y_i \rightarrow X_d$ to d ouput translation function;
$Select$ is the subsets of $D \rightarrow D$: tie-breaking function.

3 MPI-based Distributed DEVS Simulation

3.1 MPI-based Distributed DEVS Architecture

In this paper, we apply Message Passing Interface(MPI) to realize distributed **DEVS** distribution . MPI is seemed as the infrastructure for message communication mechanism. The four layer distributed architecture is shown in Figure 1. Based on the **DEVS** Model and **DEVS** Simulator, **DEVS** MPI Middleware and MPI are constructed to achieve distribution. Distributed Simulators are the key parts to add the distributed features to **DEVS** simulation without touching the modeling aspect. Atomic and Coupled model are still simulated in the sequential simulator and coordinator. Distributed Simulator and coordinator collect and send the messages from sequential ones to Message Transition module. This module organizes and sends the messages to MPI. In another way, Message Transition receives the messages from MPI. Distributed simulators subsequently pass them to the sequential simulators to accomplish the message circulation.

3.2 DEVS Simulator Distribution

DEVS Simulation Distribution is realized by the distributed simulator. We implement optimistic distributed simulator on the basis of sequential simulator under the instructions from [7] and [8]. The Partition distributes models first, then distributed simulators are created separately on different nodes accordingly. The Root node that simulates the Root Model starts and stops the simulation while all the models are connected by messages. Messages are divided into Local Message and Remote Message. Remote Message is used to connect models wherever they are located. Using Time Warp, optimistic simulator increases the simulation speed greatly.

3.2.1 Partition

Partition is the division of simulation models running in different nodes in distributed simulation. We give the definition of partition below:

$$P = \cup P_i, N = \cup N_i, i \in \{1, \ldots k\}, P_i = \sum_{j=1}^{l} M_{i,j} \quad (3)$$

$$M = \sum_{i=1}^{k} \sum_{j=1}^{l} M_{i,j}, M_{i,j} \in \{a, c\} \quad (4)$$

The P_i represents the partition in every simulation node N_i. M is the set of simulation models, composed of $M_{i,j}$ means the indexed j model located in the N_i. $M_{i,j}$ is either the Atomic(a) or Coupled(c) model. We don't touch the partition too much in this paper, only give the principles of partition in MPI-based Distributed DEVS Simulation:
(1)$\exists P_r = M_{r,0}, M_{r,0} = root(M)$, Root Model is the fundamental model in DEVS simulation, its simulator is a bit different from the common model simulator. For this reason, the N_r only simulated Root Model is named Root node.
(2)$\forall M_{i,j} \in P_i, M_{i,j} \in \{c\}, \forall M_{i,k}, M_{i,j} \neq parent^n(M_{i,k})$, $1 \leq n \leq depth(hierarchy)$. Partition does not permit the model allocated together with its parent, even the n-level parent. If the coexistence happens, it makes no sense to distribute the model. Obviously, the messaging speed inside the coordinator is much faster. When the models share the same parent are partitioned in the same node, the messages between them have to be transmitted in MPI DEVS Middle. The extra transmission slows the speed too much. And the burden on node is not relieved either.

3.2.2 Optimistic Distributed Simulator

We apply Time Warp algorithm in Optimistic Distributed Simulator, and in consequence the simulator is extended to implement the optimistic time advance. **DEVS** Models, Local Simulators, Message Manager and Simulator Queues constitute the simulator structure shown in Figure 3. Meanwhile, Simulator Queues containing State Queue, Input Queue and Output Queue are also seemed as the message window in DEVS MPI Middleware. State Queue is used to save model states in time order. Input Queue and Output Queue are linked to the Message Manager which handles the messages in and out of the Local Simulators. The two queues store Remote Messages. The messages to be received by Local Simulator are stored in Input Queue, and the messages to be sent are stored in Output Queue. DEVS MPI Middleware is designed to collect the messages from optimistic simulator and send them to MPI infrastructure. Message

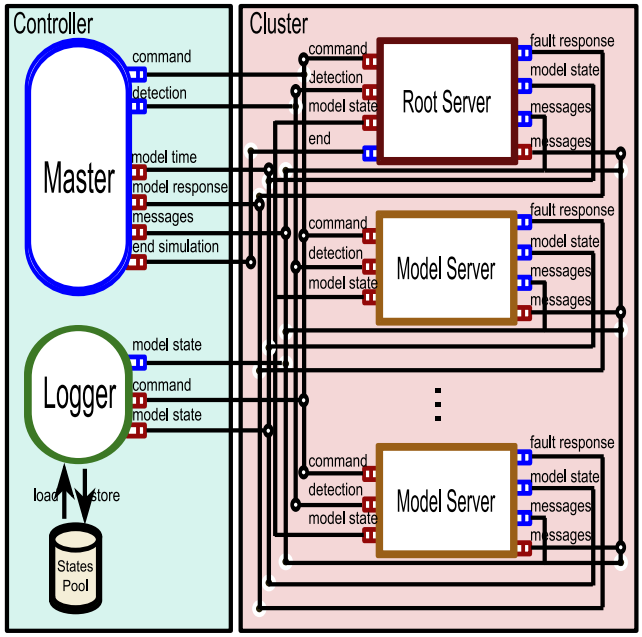

Figure 2: Fault Tolerant supported Distributed DEVS Framework

Transition wraps and unwrapps the Remote Messages. Init, X and Y messages are treated respectively due to the different message entrance in Message Manager.
At runtime, the partition is published to all the simulation nodes, then the models are loaded. Init messages are broadcasted to all the models to the initialization. When the simulation is started, all the simulators send the local star messages to themselves to time advance. During the simulation, when the destination of the messages is located remotely, the messages are sent to DEVS MPI Middleware and store in the Output Queue. In a reverse manner, simulators inquiry DEVS MPI Middleware if received some messages, and store them in Input Queue. The mechanisms in Message Manager to handle the messages are different in distributed simulator and coordinator.
In distributed simulator, local simulators process the messages directly without delivery. Local star messages which lead to outputting Y messages are sent by the simulator itself. There only exist the Init/X messages in Input Queue.
In distributed coordinator, the messaging process is shown in Figure 3. Message Manager transmits Init, X and Y messages to hierarchical simulators if the Input Queue is not empty, otherwise star messages are sent to local simulators to advance time. Compared to the sequential coordinator, receiving star messages have to obey the following rules.

1. The sub models located remotely are ignored in the selection of immediate star model.

2. The star messages usually lead to the Y messages from the coupled model. The Y messages are wrapped into the Remote Messages and subsequently sent to MPI DEVS Middleware. In the mean time, they are stored in the Output Queue.

3. The termination condition has to be inquired after receiving star message in Root node.

It is worth to note that Init/X messages and Y messages are treated respectively in Message Manager. We take the three level hierarchy coordinator in Figure 3 for example. The traces in blue signify Init/X messages while the red traces represent Y messages.
If the Message Manager receives Init/X messages from Input Queue: (1) The Init/X messages arrive at Coordinator-0 first.
(2) Identified by the coupled model, the input messages are transmit to the sub simulators: Simulator-0 and Coordinator-1. If they are not the local simulator, the messages have to be wrapped into remote Init/X message and sent to relative nodes again.
(3) Coordinator-1 continues to transmit the messages to its sub simulators: simulator-1 and simulator-2. The transmission is finished when the Init/X messages cover all the local simulators.
If the Message Manager receives Y messages from Input Queue: (1) The parent simulator is found based on the parent model information in Y messages. The Figure 3 shows that Simulator-1 is the parent simulator. (2) Simulator-1 generates its own Y message by receiving input Y message, soon after the Simulator-1's Y message is sent to parent Coordinator-1. (3) Coordinator-1 converts the Simulator-1's Y message to be its own X message and Y message. X message is sent to Simulator-2 and Coordinator-1's Y message is sent to parent Coordinator-0. (4) Similar to Coordinator-1, Coordinator-0 converts the Y message to be its own X message and send to simulator-0. The message traces end when all the local simulators are influenced by the input Y messages.

4 Fault Tolerant in Distributed DEVS Simulation

We introduce the design and implementation of Fault Tolerant in this section. Even the most robust simulation system can not claim to be immune from faults. The distribution of **DEVS** simulation brings the hardware redundancy. So it is possible to develop a framework to support Fault Tolerant.

4.1 Fault Tolerant supported Distributed DEVS Framework

In this framework, there are four kinds of nodes: Master, Logger, Model Server, Root Server. Figure 2 illustrates the structure of framework and connections between nodes. Master and Logger are responsible for Fault

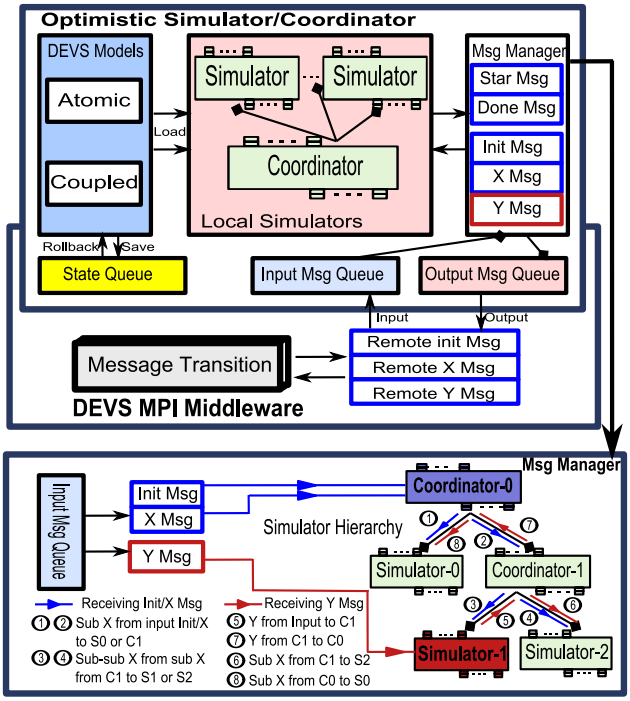

Figure 3: Distribute Simulator

Table 1: Message Channels

Channel Type	Messages in Channel
Command	Init, Start, Pause, Continue, Stop
Detection	Detect, Response, LVT, GVT
Model	Remote Messages
Log	ST_{as}, ST_{cc}, Request

Tolerant while the servers are models' simulation platforms. In addition, State Pool subordinated to Logger stores states at intervals. Master is seemed as the control center in the framework, it controls simulation's life circle and decides when and which node is fault.

Master and Logger are connected with all the Model Servers including Root Server by message channels. The channels are constructed on MPI infrastructure. The channels are divided into Command channel, Detection channel, Model channel and Log channel. Tab 1 describes the channels and the messages in them. Command Message is used to control simulation experiments, Partition and Repartition are attached to the Init and Continue messages. Detection detects the fault nodes, GVT is attached to Detect message while LVT is attached to Response message. Model Message is transmitted among model servers, including X-Y and anti messages. Log Message is the package within the simulator states and the request for reloading fault models. Fault Detection, States Storage and Fault Recovery are the basic elements in Fault Tolerant. We describe the algorithms and detail how they work in the framework in next sections.

Table 2: States for Storage

Entity Type	State Definition
Atomic Model	$ST_a = S$
Atomic Simulator	$ST_{as}\{ST_a, t_l, t_n, LVT\}$
Coupled Model	$ST_c = \{S_i\}, S_i \in \{ST_a, ST_c\}$
Coupled Coordinator	$ST_{cc}\{ST_c, t_l, t_n, eventList, LVT\}$
Node	$ST_n = \{S_i\}, S_i \in \{ST_{as}, ST_{cc}\}$

4.2 Fault Detection

We consider the faults in three categories:(1)Hardware faults to halt simulation in some node.(2)Operation System shut down for some reason.(3)The simulator crashes. The logical faults in model and MPI-based Communication faults between nodes are not covered here. And we assume that master and logger are robust enough to maintain the thousands times of simulation.

These faults cause a common phenomenon: The models on the node do not respond to any queries no matter what they are. For this reason we use detection messages to inquiry Timeout. The threshold value of Timeout is given according to specified simulation at first. Master sends detect messages to model servers periodically during simulation. If the model servers do not respond in time, the node is seemed to have been in fault.

It is important to determine the appropriate threshold value in relative simulation. If the value is larger than need, the recovery of simulation can not be activated timely. If the value is less, the false fault report causes the unnecessary recovery. Timeout differs from simulation to simulation. It is not reasonable to keep the fixed value in all simulations. We do the Timeout computation by way of modeling simulation platform including simulation entities and Fault Tolerant entities.

These entities are built in **DEVS**. Simulation entities simulate the protocol of Atomic Solver(simulator) model, Coordinator model and Root Coordinator model while Fault Tolerant entities are composed of Master and Logger Models. These **DEVS** models are used to load the **DEVS** models for concrete simulation applications. Once the entities and models are ready, the sequential simulation can be executed to compute the threshold value of Timeout. Regarded as the reference value, sequential result helps find the accurate optimal Timeout by the experiments on real system.

4.3 States Storage and Fault Recovery

Fault Recovery is actually the rollback to last correct state. So the States Storage is the basis of Fault Recovery like the State Queue in Time Warp. The definition of state from atomic model to node is listed in Tab 2. It is noted that simulators report states to Logger immediately after each time advance operation. The states in the phase of receiving input messages are abandoned because the Input and Output Queues have to be stored

in case of recovering this kind of state. Logger organizes the log messages into groups by nodes and stores them into States Pool in time order. The State Pool updates the model states as soon as the log messages are coming. Supported by Logger and State Pool, The algorithm of Fault Recovery is given below:

1. N_* is found to having been fault by Fault Detection.
2. `re_partition` returns N_r for reloading the fault models.
3. $\forall m_{*,j}$ in N_*, $s_{r',j}$ is created in N_r, N_r requests $ST_{as}(*,j)$ or $ST_{cc}(*,j)$ from Logger to update $s_{r,j}$. t_* is acquired from $s_{r',j}$.
4. $\forall s_{i,j}, i \neq r', \forall msg \in Q_i$, if $t(msg) \geq t_*$ and msg received from $m_{*,j}, s_{i,j}$ receives $\neg msg$.
5. $\forall s_{i,j}, i \neq r', \forall msg \in Q_o$, if $t(msg) \geq t_*$ and msg sent to $m_{*,j}, s_{i,j}$ sends msg to $s_{r',j}$.

The `re_paritition` is to allocate models again to the healthy nodes. The scheme is designed in two respects:(1) The models on the fault node are all reloaded on the chosen node. (2) The new recovering node is chosen according to the number of atomic models. On one hand, compared to the original partition, the new one does not distribute the reloading models. So the communication load can remain the same level as before. On another hand, atomic models are in charge of most computation in **DEVS** simulation. The number of atomic models indicates the computation burden on the node. `re_partition` chooses the node with the least atomic models to recover fault models.

5 Case Study and Analyze

5.1 Model Description

We build the City Map to model the traffic in city. The entities such as District, Home, Office, TrafficLightController, RoadSegment and Bridge constitute the city. Car movement between entities is seemed as the message in implementation. Home, Office, TrafficLightController, RoadSegment and Bridge are the atomic models, they are used to couple District. City Map is composed of an amount of Districts and the RoadSegments or Bridges connecting the Districts. At the beginning of the simulation, cars are generated in each Homes. Soon after they are transmitted to Office through RoadSegment, TrafficLightController or Bridge. Finally, when all the cars go back Home, the simulation is terminated.
The complexity of City Map is determined by the number of Districts. In order to adjust the complexity in experiments, we give the number in initialization.

5.2 Experiments

Supported by **Python DEVS**, simulators, Fault Tolerant servers and models are implemented in **Pyhon**. MPI is the infrastructure of distribution of simulation. Therefore MPI4Python is used to develop the distributed simulators. Conservative and optimistic simulators are both implemented to show the improvement from sequential to distributed, from distributed to parallel. Our testbed is a 32 nodes cluster installed the Linux Fedora 9.0, Python 2.5, openMPI 1.2.7 and MPI4Python 1.1.0.
The experiments are designed in two sets: Efficiency centered and Fault Tolerant centered. The former focuses on the speedup of simulators using Time Warp. The latter obtains Timeout's optimal value for Fault Tolerant.
The top picture in Fig 4 gives the consumed time curves of different number of Districts. The performance data is collected when the Districts number is 1, 2, 4, 8 and 16. Moreover, the curve in blue indicates the simulation executed on sequential simulator while the red one is the result from optimistic simulator on 2 nodes.
In another aspect, we select Timeout to do the optimization for Fault Tolerant. The Accuracy of Fault Detection is defined to help find the optimal value. The number of Districts is 4 and the samples of Timeout are ranged from 0.0 to ∞. The experiments are performed 10 times at each point. Accuracy is the correct detection proportion in experiments.

5.3 Analyze

From the Efficiency centered experiments, it is shown that the optimistic simulator accomplishes a great deal in improving performance. The top graph shows that the consumed time approaches the half of the sequential by increasing Districts number. Obviously, the performance will be elevated significantly by increasing nodes number.
In the Fault Tolerant centered experiments, the Accuracy curve indicates that the Accuracy increases from 0 to 100% along with the Timeout from 0 to ∞. 100% Accuracy is the goal of Fault Tolerant. Simultaneously, the less the Timeout is, the better for the performance. If the Timeout is larger than necessary, lots of time is wasted in detecting faults. Therefore, the fault can not be corrected immediately, the chain faults will happen soon after. It costs much more than need for recovering the faults. In the result, the simulation performance is lowered greatly. So the value of Timeout just touches the 100% Accuracy is the most optimal one. In our experiments, the best Timeout is 0.3 second as shown in Fig 4.

6 Conclusions and Future Work

We have introduced how to do the **DEVS** simulation distribution with the support of MPI. The algorithms to realize the conservative and optimistic simulators are presented in detail. Based on the distribution, the

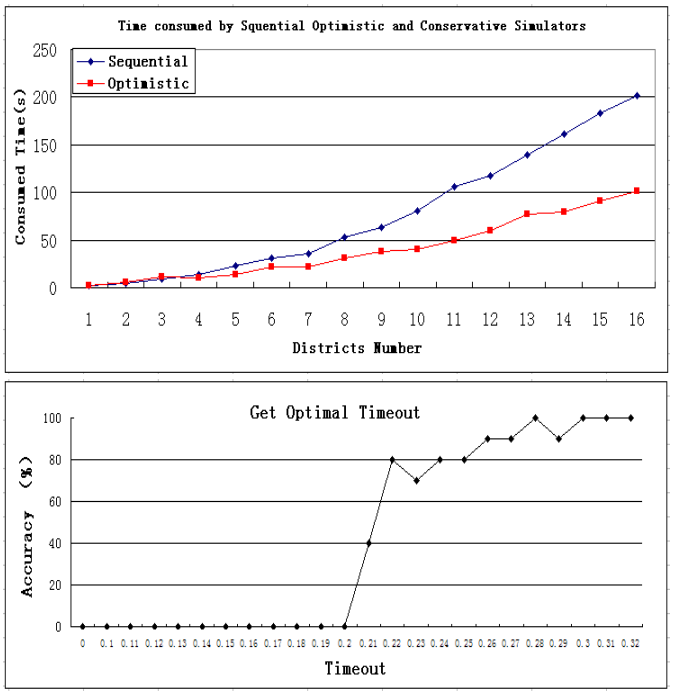

Figure 4: Experiments Results for Distribution and Timeout

framework for supporting the Fault Tolerant is designed. Fault Detection, States Storage and Fault Recovery are described respectively in advance, Master, Logger and Model Server are implemented to fulfill the requirements from Fault Tolerant. Two sets of experiments are performed to show the improvement of distributed **DEVS** simulation and the optimization for Fault Tolerant. The results testify that optimistic simulators speedup the simulation a lot. And the most optimal value for Timeout is acquired at the moment Accuracy just reaches 100%. For future work, it is necessary to further the optimization in distributed simulators. The algorithms such as machine-learning have to be applied on the settings in Time Warp. They can adapt to the features in different models, hence the optimistic simulators can achieve the highest speed in all kinds of simulation. Moreover, the calibration of modeling simulation platforms is not limited to the Timeout. The other parameters like time delay of messages can be modeled and in consequence calculated to get the optimal value in Fault Tolerant framework.

References

[1] Fujimoto, R. M. "Parallel and distributed simulation systems," *Simulation Conference, 2001. Proceedings of the Winter*, Arlington, VA, USA, pp. 147-157.

[2] Bernard. P. Zeigler, H. Praehofer, and T. G. Kim. *Theory of Modeling and Simulation*, Second Edition, Academic Press, 2000.

[3] O.P. Damani and V.K. Garg, "Fault-Tolerant Distributed Simulation," *Parallel and Distributed Simulation, Workshop on Parallel and Distributed Simulation*, Los Alamitos, CA, USA, V1, pp. 3-8.

[4] Aguilar, Jose and Hernández, Marisela, "Fault Tolerance Protocols for Parallel Programs Based on Tasks Replication," *MASCOTS '00: Proceedings of the 8th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, Washington. DC, USA, pp. 397-404.

[5] Dan Chen, Stephen J. Turner, Wentong Cai, "Towards Fault-tolerant HLA-based Distributed Simulations," *SIMULATION*, V84, N10-11, pp. 493-509.

[6] Min M., Shiyao J., Chaoqun Y., Xiaojian L., "Dynamic Fault Tolerance in Distributed Simulation System," *Proc. International Conference on Computational Science (1), 2006*, Reading, UK, pp.769-776.

[7] Qi L., Wainer G., "Parallel Environment for DEVS and Cell-DEVS Models," *Simulation* V83, N6 pp. 449-471.

[8] Feng, B., Liu, Q., Wainer, G., "Parallel simulation of DEVS and Cell-DEVS models on Windows-based PC cluster systems," *SpringSim '08: Proceedings of the 2008 Spring simulation multiconference*, Ottawa, Canada, pp. 439-446.