A METHODOLOGY FOR THE DEVS SIMULATION GRAPH CONSTRUCTION

Adedoyin Adegoke[a], Ibrahima Amadou[b], Hamidou Togo[b], Mamadou K. Traoré[c]

[a] African University of Science and Technology, Abuja (Nigeria)
[b]Université de Bamako (Mali)
[c] LIMOS, CNRS UMR 6158, Université Blaise Pascal, Clermont-Ferrand 2 (France)

[a]aadegoke@aust.edu.ng, [b]temena2004@yahoo.fr, [c]traore@isima.fr

ABSTRACT
Various DEVS (Discrete Event Systems Specification) implementations exist but differ in the approaches considered for use. These approaches include how events are processed, the simulation architecture in use, the existing procedures (set of rules/algorithm), the organizational architecture (of the simulator) and so on. This work attempts to formalize a generic approach to Parallel and Distributed Simulation (PADS) with DEVS as well as embody these approaches by providing formal definitions that can be standardized for use during DEVS implementation. Therefore, we propose a DEVS Simulation Graph which gives basic information about the necessary elements that are useful for the analysis and construction of a DEVS simulator. The major aim is to help identify these elements before implementation and ease the development of a DEVS simulator.

Keywords: PADS, DEVS, Simulation Graph, Simulator

1. INTRODUCTION
DEVS (Discrete Event System Specification) can be called a universal simulation Turing machine as it offers a platform for the modeling and simulation of sophisticated systems in a variety of domains. It unifies various formalisms and provides a general description for the construction and execution of a model from an original system. Due to the separation of concerns in DEVS, the modeler needs to focus only on the models being created avoiding the details about how the simulator was built.

Parallel and Distributed Simulation (PADS) (Fujimoto 1990) has been a widely researched area in recent years. Its prominence offers increase in execution speed, reduction in execution time, execution of larger simulation models and increased processor fault tolerance to a possible failure. In addition, it provides a solution to the scientific need to federate existing and naturally dispersed simulation codes.

Although PADS is a matured field of study, its adaptability to existing modeling and simulation formalisms is an arduous task. PADS with DEVS implementation strategies differ from one another. Based on this heterogeneous factor the intrinsic elements used in developing DEVS simulators are not formally defined for these strategies. Hence, this work will attempt to identify and capture the elements commonly used in these strategies as well as propose a more generic approach and formal framework which is deemed necessary. These fundamental elements are in terms of the simulator's tree structure, the number of execution streams and the number of computing resources.

The rest of the paper is organized as follow: Section 2 presents a review of some existing works in the area of PADS with DEVS. Section 3 presents the foundations of DEVS simulation i.e. the Simulation Tree, the concept of the Simulation Graph and the fundamental elements. Section 4 presents a methodology and basic operations which are useful for the construction of the Simulation Graph. In section 5, we present a case study and a look at Simulation Graph approaches in existing works. Finally, we conclude in Section 6.

2. PADS WITH DEVS – IMPLEMENTATIONS
We take a look at some implementations which have attempted combining PADS with DEVS.

Himmelspach, Ewald, Leye, and Uhrmacher (2007) proposed a Parallel Sequential Simulator which was implemented to ease the distribution of DEVS models on several physical processors consequently introducing the need for partitioning and load balancing. Also, it proposes performing Sequential and/or Parallel execution.

Parallel variant of the CD++ (Wainer 2009) tool was designed to execute DEVS models on parallel memory architectures i.e. with the idea of distributing the simulating entities on different physical processors.

DEVS-Ada/TW (Christensen and Zeigler 1990) is the first attempt to combine DEVS and Time Warp mechanism for Optimistic Distributed simulation. The hierarchical DEVS model is partitioned at the highest level of the hierarchy and as a consequence, the flexibility of partitioning models is restricted.

The DOHS (Distributed Optimistic Hierarchical Simulation) scheme proposed by Kim, Seong, Kim and Park (1996) is a method of distributed simulation for hierarchical and modular DEVS models. It uses the

Time Warp mechanism for global synchronization. Each node of the simulation tree structure is revised to adapt to a simulation parallel/distributed environment.

There are also other variants that take the structure of non-hierarchical DEVS model to help reduce the cost of exchanged messages in the simulator. This is the case of optimistic simulation in P-CD++ (Qi and Wainer 2007) an optimistic version of the CD++ tool which was developed for the simulation of DEVS and Cell-DEVS models.

Contrary to optimistic approaches, few parallel DEVS simulators belong to the conservative class. In (Zeigler, Praehofer and Kim 2000), a distributed simulation framework (Conservative Parallel DEVS Simulator) is described for non-hierarchical DEVS models using conservative synchronization. In addition, the performance of a conservative approach depends strictly on a good look-ahead.

3. FROM DEVS SIMULATION TREE TO DEVS SIMULATION GRAPH

We interpret the building of a Parallel and Distributed Simulation (PADS) with DEVS as a move from the original Simulation Tree (ST) to a Simulation Graph (SG).

3.1. DEVS Sequential Simulation Tree

DEVS formalism (Zeigler, Praehofer and Kim 2000) specifies system behavior as well as system structure. System behavior in DEVS is described as input and output events as well as states while system structure is built from the composition of atomic or coupled models. A coupled model is composed of several atomic or coupled models and atomic model is a basic component that cannot be decomposed any further.

A DEVS model is built according to specification i.e. Classic DEVS or Parallel DEVS. CDEVS (Classic DEVS System Specification) was introduced in 1976 by Zeigler (Zeigler 1976) to simulate and execute models sequentially on single processor machine. As a solution to the rigidity in CDEVS, the appropriate execution of simultaneous events has led to the concept of process and to PDEVS (Parallel DEVS System Specification) (Chow and Zeigler 1994).

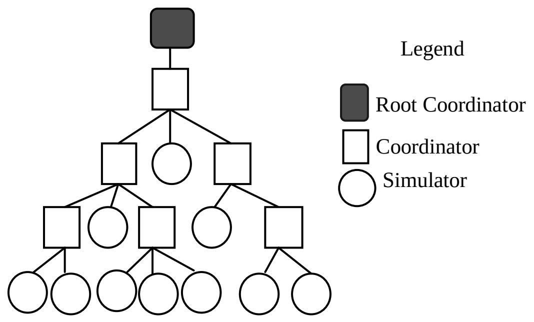


Figure 1: DEVS Simulation Tree

DEVS model execution is driven through the simulation tree which serves as vital element in construction of a DEVS simulator. The tree consists of simulating nodes which are used for executing DEVS models. These nodes are Coordinators, Simulators and Root Coordinator which are organized in a hierarchy that mimics the hierarchical structure of a DEVS model.

A DEVS atomic model is executed by assigning a simulator to it and to a DEVS coupled model a coordinator is assigned. Root Coordinator is a special coordinator that drives the global aspects of the simulation on a tree; it initializes and ends the simulation (when a termination condition is detected).

3.2. PADS with DEVS Simulation Graph

Due to the increasing number of complex model systems it is necessary to improve efficiencies and performances of DEVS simulators. A typical PADS with DEVS implementation will result in the Simulation Graph (SG). A SG is a representation of the relationship between a DEVS simulator's fundamental elements which are simulation tree, process and processor.

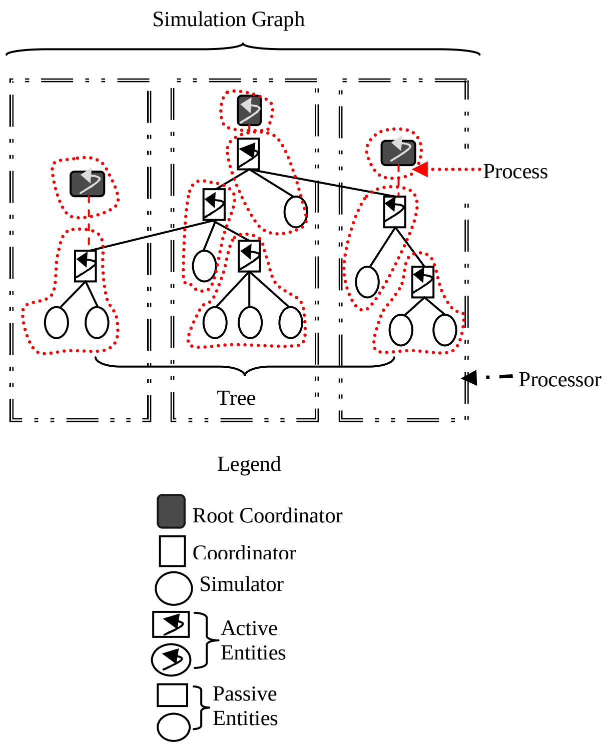


Figure 2: DEVS Simulation Graph

The structure of the DEVS simulator tree can be altered to improve performance of a simulator either by reducing (also known as flattening) the number of nodes on the tree as seen in (Jafer and Wainer 2009) or increasing the number of nodes (with specialized Coordinators and Simulators) as seen in (Troccoli and Wainer 2003).

Also, this tree can be split to form sub-trees based on the analysis of the model's structure. A simulator with single tree structure is designed with the use of a central scheduler called the Root Coordinator while in a multiple tree structure simulator each of the sub-trees has its own central scheduler/Root Coordinator and different simulation clocks. This is the preferable solution in distributed simulation.

We define a process as a stream of execution. It contains two types of entities during execution; they are active and passive entities. An active entity is an entity that is currently active in an execution stream (e.g. Java threads, ADA Tasks, etc.). While a passive entity is part of an execution stream but not actively involved until it is activated e.g. function calling in Object Oriented Paradigm. We consider that a process would have at most one active entity. If a process has more than one active entity, those entities are then regarded as being autonomous sub-processes. Also, there can be more than one passive entity in a process.

A processor is a computing resource that allows the execution of a program (a process, an entire tree, any other executable code) on itself.

4. METHODOLOGY FOR BUILDING THE SIMULATION GRAPH

The state chart provides an overview of the method and the trajectories describe the set of all possible paths that can be taken during the construction of the Simulation Graph (SG).

The SG construction is driven based on the analysis of the initial Simulation Tree, the available number of Processes and Processors. An overview of this methodological approach is given by the following state chart. It is worth noting that the methodology iterates on each state until some user-defined satisfaction criteria are reached (optimal splitting, optimal clustering, optimal mapping and optimal transformation).

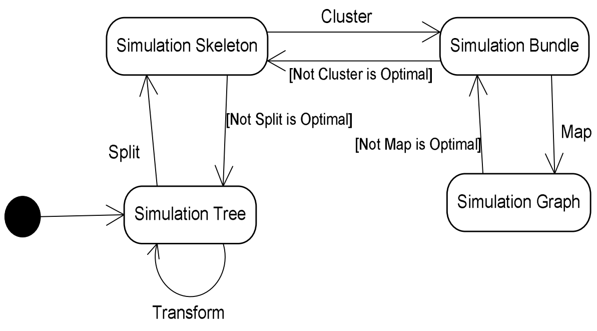


Figure 3: Simulation Graph Methodology

The process of Transformation, Splitting, Clustering and Mapping continues until it is certain that a good performance or speed will be gained during simulation from the new Simulation Graph.

4.1. Simulation Skeleton and Bundle

Informally presented, the Simulation Skeleton is the structure of the simulation protocol that can fit the PADS scheme. The Simulation Bundle is a collection of cluster of nodes. Examples are shown with Figures 4 and 5.

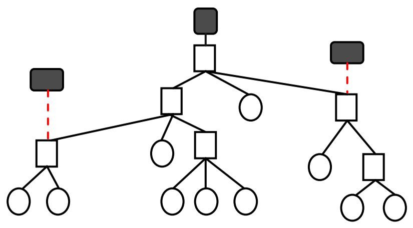


Figure 4: Simulation Skeleton

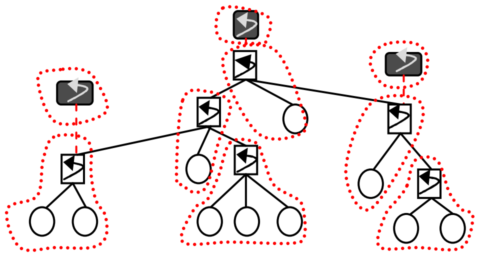


Figure 5: Simulation Bundle

4.2. Simulation Structures Formalized

Definition 1: A Simulation Tree T, can be defined as

$$T = < R, N, f >$$

With:

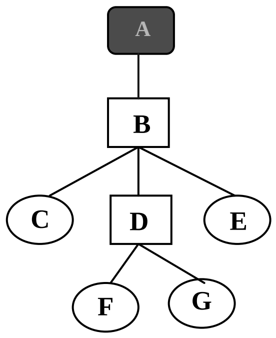
- $R \in N$
- $f: N \rightarrow \wp(N)$ where $\wp(N)$ is Power Set of N
- $f^{-1}(R) = \emptyset$
- $f^{-1}(J) \neq \emptyset,\ \forall J \in N - \{R\}$
- $Cardinal\ f(R) = 1$

Where:

R: the Root Coordinator of the tree
N: the set of nodes of the tree
f: a function that maps a child node to its parent

A node is a "parent" of another node (its child) if it is one step higher in the hierarchy.

For example:



Tree T will be defined as

$R = \{A\}$
$N = \{A, B, C, D, E, F, G\}$
$f(A) = \{B\}$
$f(B) = \{C, D, E\}$
$f(D) = \{F, G\}$
$f(C) = f(E) = f(F) = f(G) = \emptyset$

Definition 2: Simulation Tree T can also be defined as

$$T = < R, N, F >$$

With:

- $R \in N$
- $F \subset N \times (N - \{R\})$
- $(a, b) \in F \iff b \in f(a)$

Using Definition 2 for the example above, R and N will be defined as same while F will be

$$F = \{(\mathrm{A},\mathrm{B}), (\mathrm{B},\mathrm{C}), (\mathrm{B},\mathrm{D}), (\mathrm{B},\mathrm{E}), (\mathrm{D},\mathrm{F}), (\mathrm{D},\mathrm{G})\}$$

Definition 3: A Simulation Skeleton is defined by

$$S = < \{R_i\}, N, f >$$

With

- $R_i \in N \; \forall$ i
- $f: N \to \wp(N)$ where $\wp(N)$ is Power Set of N
- $f^{-1}(R_i) = \emptyset \; \forall i$
- $f^{-1}(J) \neq \emptyset, \; \forall J \in N - \bigcup\{R_i\}$

Definition 4: A Simulation Bundle is formally defined as

$$B = < \{R_i\}, N, f, Ps, Cluster >$$

With

- $< \{R_i\}, N, f >$ as a skeleton
- Ps is the set of Processes
- $Cluster: N \to Ps$

Definition 5: A Simulation Graph is defined by

$$G = < \{R_i\}, N, f, Ps, Pr, Cluster, Map >$$

With

- $< \{R_i\}, N, f, Ps, Cluster >$ is a Simulation Bundle
- Pr is a set of Processors
- Map: $Ps \to Pr$

4.3. Basic Operations Formalized

In this section we show that moving from a Simulation Tree (ST) to a Simulation Graph (SG) can be decomposed into basic operations and later we will show the methodology that drives this process.

4.3.1. Split

It is a function used for creating a partition of simulating entities/nodes from a simulation tree.

Definition 6: Split: $\tau \to \Sigma$

With
$T = < R, N, f >$
$Split(T) = < \{R_i\}, N', f' >$
Based on the following conditions:

- $R \in \bigcup\{R_i\}$
- $N' = N \bigcup \{R_i\}$
- $f'_{/N} = f$

Where τ is the set of all possible trees and Σ is the set of all possible skeletons.

4.3.2. Cluster

This function takes the available number of nodes and groups them into Processes.

Definition 7: Cluster: $N \to Ps$

Where
Ps is the set of Processes.
Based on the conditions that

- $Cluster^{-1}(p)$ is Connex $\forall p \in Ps$
- $\forall \, p_i, p_j \in Ps, \; p_i \neq p_j,$
  $Cluster(p_i) \cap Cluster(p_j) = \emptyset,$

4.3.3. Map

This function takes the set of available Processes and plots them onto the set of available Processors.

Definition 8: Map: $Ps \to Pr$

Where

- Ps is the set of Processes
- Pr is the set of Processors

Based on the following condition
$\forall \, p_i, p_j \in Ps, \; p_i \neq p_j, \; Map(p_i) \cap Map(p_j) = \emptyset$

4.3.4. Transform

Transform is a function used for altering the Simulation Tree structure either by expansion or reduction. This altering is done on the number of available nodes (not including the Root Coordinators) on the Tree and their relationships.
Definition 9: $Transf[Na, Nr, Fa, Fr]: \tau \to \tau$

$$Transf[Na, Nr, Fa, Fr](< R, N, F >) = < R, N', F' >$$

With

- $N' = N \bigcup Nr - Na$
- $F' = F \bigcup Fr - Fa$

Where

- Na is the set of nodes to be added to N
- Nr is the set of nodes to be removed from N
- Fa is the set of relationships to be added to F
- Fr is the set of relationships to be removed from F

Based on the following conditions:

- $Na \cap N = \emptyset$
- $Nr \subset N - \{R\}$
- $Fa \subset (N \times Na) \bigcup Na^2 \bigcup (Na \times N - \{R\})$
- $Fr \subseteq F$

5. APPLICATION

5.1. Case Study

In this study we describe a possible path in the application of the Simulation Graph construction methodology. This application starts with the original DEVS tree. At the Transform stage the tree structure is modified by reduction after which the tree is Split to create a partition of nodes also increasing the number of Root Coordinators on the entire tree. These partitions of nodes are grouped into available number of Processes thereby creating a Simulation Bundle (Figure 6c). At the final stage (Map), a Simulation Graph was created by mapping the each Process on the Simulation Bundle to an available number of Processors. See Figure 6.

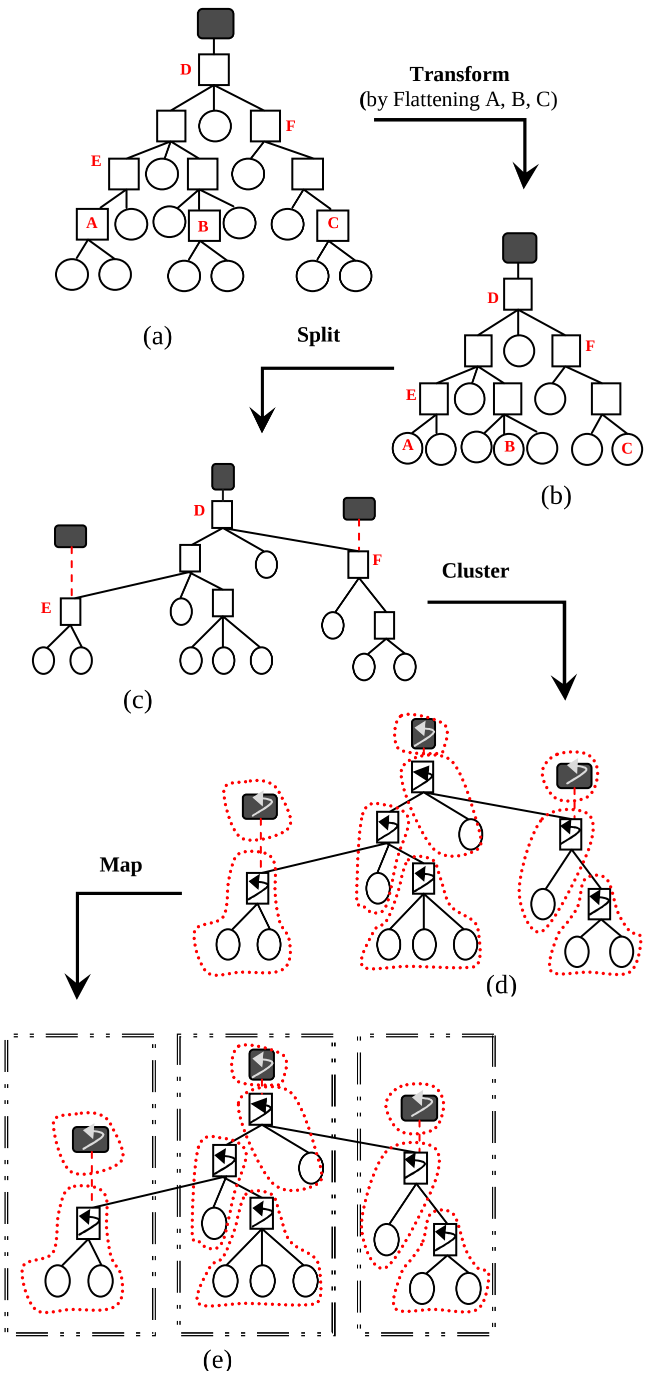


Figure 6: Application of Simulation Graph Methodology

5.2. Works Revisited

In a more general overview, most DEVS implementation decisions have been observed to be based on the fundamental elements i.e. simulation tree, process and processor. We present some of them here and their Simulation Graph.

To explain the strategies in the literature in a more formal way we suggest the Tree-Process-Processor notation. It consists in defining the number of elements for each aspect of PADS. We use N for "many elements". For example, a 1-1-1 scheme is a Simulation Graph with 1 tree, 1 process and 1 processor while N-N-1 is a Simulation Graph with many trees, many processes and 1 processor.

PythonDEVS (Bolduc and Vangheluwe 2002) is a 1-1-1 strategy. It uses the CDEVS formalism in specifying models and as a consequence it performs sequential simulation.

The Abstract Threaded Simulator of the James II (Himmelspach and Uhrmacher 2006) package uses a 1-N-1 strategy with its processes created using Java threads. Depending on the memory size of the processor and the model size, the cost of creating threads gets expensive as the number of models increases. This is a critical factor to be considered when using many processes.

In (Troccoli and Wainer 2003), the Parallel CD++ Simulator is a 1-N-N strategy. Also, new simulation nodes were used on the tree thereby expanding it. This methodology better suits distributed simulation but increases the cost of communication between the nodes. Also, Parallel Sequential Simulator by Himmelspach, Ewald, Leye and Uhrmacher (2007) uses this 1-N-N strategy.

The Conservative CD++ (Jafer and Wainer 2010) is an N-N-N strategy. In order to reduce communication costs between the nodes the simulator was flattened. The flattening involves a reduction in the number of nodes on the simulation tree. Some other works that uses the N-N-N strategy include DEVS-Ada/TW (Christensen and Zeigler 1990), DOHS scheme by Kim, Seong, Kim and Park (1996) and Optimistic Parallel CD++ (Qi and Wainer 2007).

Also, we observed that having an implementation which involves the use of N-1-1 strategy or N-1-N strategy is not realistic. The reason for this is execution of each tree is asynchronous and can be done simultaneously using many processes instead.

6. CONCLUSION

This paper is part of a more general research direction in that we investigate various approaches commonly used in PADS with DEVS to build simulators. We presented the Simulation Graph concept to help ease the process of building DEVS simulators and provide a common platform for different implementation strategies. This was achieved through the identification of the fundamental elements used in DEVS simulators and the relationship between them. Thus, with this we were able to provide definitions for a formal framework as opposed to the traditional intuitive way of constructing a DEVS simulator. Also, we presented a possible path in the application of the Simulation Graph and revisited works by identifying their Simulation Graph approach. Further works include automating this process by implementing the methodology in SimStudio package (Traoré 2008).

ACKNOWLEDGMENTS
The work in this paper is partly funded by grants from RAMSES (*Réseau Africain pour la Mutualisation et le Soutien des poles d'Excellence Scientifique*).

REFERENCES

Bolduc, J., Vangheluwe, H., 2002. *A Modeling and Simulation Package for Classic Hierarchical DEVS.* Technical Report, McGill University, School of Computer Science.

Chow, A. C., Zeigler, B. P., 1994.Revised DEVS: A Parallel, Hierarchical, Modular Modeling Formalism. *Proceedings of the Winter Simulation Conference*

Christensen, E.R., Zeigler, B. P., 1990. Distributed Discrete Event Simulation: Combining DEVS and Time Warp. *In Proceedings of the SCS Eastern Multiconference on AI and Simulation Theory and Applications*

Fujimoto, R. M., 1990. Parallel Discrete Event Simulation. *Communications of the ACM,* 33(10), 30-53.

Himmelspach, J., Ewald, R., Leye, S., Uhrmacher, A., 2007. Parallel and Distributed Simulation of Parallel DEVS Models. *Proceedings of the 2007 Spring Simulation Multiconference*.

Himmelspach, J., Uhrmacher, A., 2006, Sequential Processing of PDEVS Models. *Proceedings of the 3rd EMSS,* 239-244.

Jafer, S., Wainer. G. A., 2009. Flattened Conservative Parallel Simulator for DEVS and CELL-DEVS. *Proceedings of CSE* (1), 443-448.

Jafer, S., Wainer. G. A., 2010. Conservative DEVS – A Novel Protocol for Parallel Conservative Simulation of DEVS and Cell-DEVS Models. *Proceedings of 2010 Spring Simulation Conference (SpringSim10), DEVS symposium 168-175.*

Kim, K. H., Seong, Y. R., Kim, T. G., Park, K. H., 1996. Distributed Simulation of Hierarchical DEVS Models: Hierarchical Scheduling Locally and Time Warp Globally. *TRANSACTIONS of the SCS International 13, no. 3, 135-154.*

Qi, L., Wainer, G. A., 2007. Parallel Environment for DEVS and Cell-DEVS Models. *SIMULATION 83(6), 449-471.*

Traoré, M. K., 2008. SimStudio. A Next Generation Modeling and Simulation Framework, *SIMUTools'08, ISBN 978-963-9799-20-21.* March 3-7, Marseille, France

Troccoli, A., Wainer, G., 2003. Implementing Parallel Cell-DEVS. *Proceedings of the 36th Annual Symposium on Simulation*, 273-277. March 30-April 02.

Wainer, G. A., 2009. *Discrete-Event Modeling and Simulation: A practitioner's Approach.* New York: CRC Press.

Zeigler, B. P., 1976. *Theory of Modeling and Simulation.* New York; Wiley-Interscience.

Zeigler, B. P., Praehofer, H., Kim, T. G., 2000.*Theory of Modeling and Simulation. Integrating Discrete Event and Continuous Complex Dynamic Systems.* London; Academic Press.