A unifying framework for specifying DEVS parallel and distributed simulation architectures

Adedoyin Adegoke, Hamidou Togo and Mamadou K Traoré

SIMULATION 2013 89: 1293 originally published online 15 October 2013
DOI: 10.1177/0037549713504983

Simulation

Simulation: Transactions of the Society for Modeling and Simulation International
89(11) 1293–1309

DOI: 10.1177/0037549713504983
sim.sagepub.com
SAGE SCS

A unifying framework for specifying DEVS parallel and distributed simulation architectures

Adedoyin Adegoke[1], Hamidou Togo[2] and Mamadou K Traoré[3]

Abstract

DEVS (Discrete Event System Specification) is an approach in the area of modeling and simulation that provides a means of specifying dynamic systems. A variety of DEVS tools have been implemented without a standard developmental guideline across the board, consequently revealing a lack of central frameworks for integrating heterogeneous DEVS simulators. When implementing a DEVS Simulator there are salient concepts that are intuitively defined, such as how events should be processed, what simulation architecture to use, what existing procedures (set of rules/algorithm) can be used, what should be the organizational architecture and so on. The aim of this paper is to propose a theoretical guide in building a DEVS distributed simulation as well as a formalization of underlying concepts to allow symbolic reasoning and automated code synthesis. From a review of existing implementation approaches, we propose a taxonomy of the identified concepts, including some formal definitions as they constitute the essential building blocks of performing Parallel Discrete-Event Simulation by utilizing DEVS. The contribution of this taxonomy and its impact as a unifying framework is that it provides a more systematic understanding of the process of constructing a DEVS simulator. Also, it offers an abstract way for integrating different and heterogeneous DEVS implementation strategies and thus can serve as a contribution to the on-going DEVS standardization efforts.

Keywords

DEVS, Parallel Discrete-Event Simulation, conceptual framework, Simulation Tree, Simulation Graph

1. Introduction

DEVS (Discrete Events System Specification) offers a platform for the modeling and simulation (M&S) of sophisticated systems in a variety of domains. It provides a mechanism to mix different formalisms as well as a generic mechanism for M&S. A DEVS simulator is capable of reproducing behaviors that are identical to that of the system under observation. In doing so, the modeler is provided with some level of abstraction by being able to build models without having knowledge of how the simulator was built.

Due to the growing complexity of systems to be modeled, efficient simulation of such systems cannot be performed on a single physical processor. One way out of this is to make use of distributed strategies by exploiting the computing power of current technologies (grid, cloud, web services, etc.). Some benefits of this include reduction in execution time, improved simulation performance, real-time execution and integration of simulators.[1] Parallel Discrete-Event Simulation (PDES)[1] is a widely researched area with some potential benefits. Firstly, the use of parallel processors promises an increase in execution speed and a reduction in execution time. Secondly, the potentially larger amount of available memory on parallel processors will enable the execution of larger simulation models. Thirdly, with the use of multiple processors comes an increased tolerance to a possible processor failure. In addition, it provides a solution to the scientific need to federate existing and naturally dispersed simulation codes. Thus, simulation architecture can be called parallel if its main design goal is to reduce execution time, while the term distributed simulation could be referred to as

[1]African University of Science and Technology, Nigeria
[2]Université des Sciences, Techniques et Technologies de Bamako, Mali
[3]LIMOS, Université Blaise Pascal, France

Corresponding author:
Adedoyin Adegoke, African University of Science and Technology, Km 10, Airport Road, Galadimawa, Abuja, Nigeria.
Email: aadegoke@aust.edu.ng

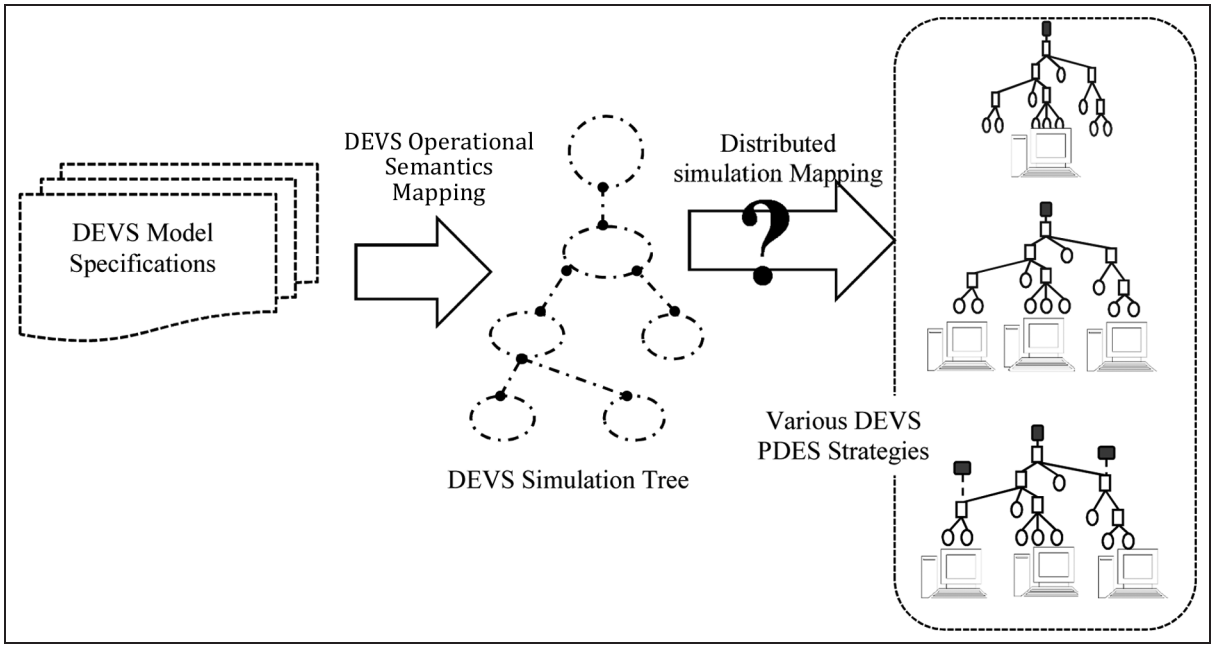


Figure 1. Process of mapping DEVS to Parallel Discrete-Event Simulation.

interoperating geographically dispersed simulators.[1–3] Building a simulation model on a particular world view significantly reduces implementation complexity.[4] However, the distribution of the DEVS simulation protocol (which unifies the three classic simulation strategies also known as the world views[2]) is a challenging issue.

PDES is a matured field of study but its adaptability to some existing M&S formalisms (e.g., DEVS or Petri nets) is an arduous task. Two main reasons concur with this point.

- Various concerns are involved in the process of building a DEVS PDES. A good building strategy would be based on a clear separation of concerns. The formal specification of the key concepts and transformation that are found in these concerns would remove ambiguity and reduce accidental complexity (i.e., wrong implementation due to misunderstanding of concepts).
- There is a lack of systematic and quantifiable approach that can guide this process.

DEVS simulation principle takes a DEVS model specification and maps it to DEVS Simulation Tree (ST) using well-defined DEVS operational semantics.[2] From studied literature it is seen that most works are moving from sequential to parallel/distributed infrastructures due to the benefits involved in so doing. Also from this study we see the existence of various DEVS PDES strategies (as shown in Figure 1). However, this raises the question of how we can achieve the mapping of a DEVS ST onto parallel/distributed infrastructures. The objective of this work is as follows.

- Propose a conceptual framework that models the process of mapping a DEVS ST to a graph of simulation components distributed over a network. Then, each builder of DEVS distributed simulation can instantiate this generic model to get his own mapping strategy which, therefore, is a guideline for any user to apply this strategy.
- Formalize the concepts and operations of such a process so that:
  1. there can be the partial or full automation of such a process;
  2. one can symbolically reason and derive properties (for evaluation or verification).

The rest of the paper is organized as follows. Section 2 presents the foundations of DEVS simulation, that is, the ST, from which all the distributed strategies are built. Section 3 presents the key concepts in use in this paper. Identified aspects in DEVS PDES as well as classification of research contributions in this area are presented in Section 4. In Section 5 we present the generic approach for building a DEVS PDES implementation and we show how it instantiates in a case study. In Section 6, we give a discussion on the framework and then conclude in Section 7.

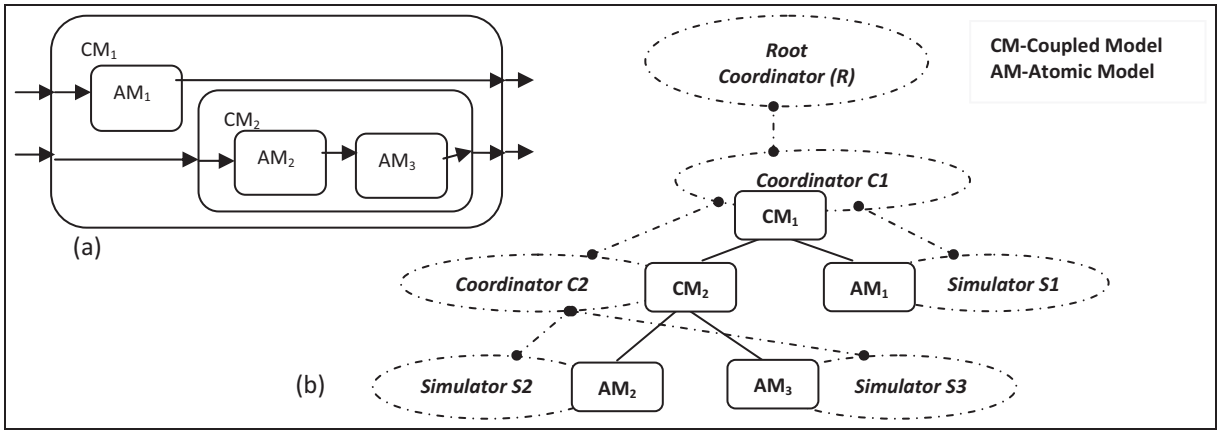


Figure 2. (a) DEVS model. (b) Hierarchical mapping of DEVS model to abstract simulator.

2. DEVS simulation protocol

The DEVS formalism[2] provides a comprehensive M&S framework for modeling and analysis of Discrete Event Systems. It specifies system behavior as well as system structure. System behavior in DEVS is described through its DEVS dynamic functions, while system structure is built from the composition of atomic and coupled models. A coupled model is composed of several atomic or coupled models and the atomic model is a basic component that cannot be decomposed any further. They are hierarchically organized as shown in Figure 2(a).

A DEVS model is built according to a specification, that is, Classic DEVS (CDEVS) or Parallel DEVS (PDEVS). CDEVS was introduced in 1976 by Zeigler[5] to simulate and execute models sequentially on single processor machine. PDEVS was later introduced to increase the potential of parallelism in simulating DEVS models.[6]

Due to the separation of concerns in DEVS, the modeler needs to focus only on the models being created, avoiding the details about the abstract simulator (algorithms). The operational semantics of DEVS models has been defined by abstract algorithms.[2] These algorithms consist of different nodes (Coordinator, Simulator) organized in a hierarchy that mimics the hierarchical structure of a model. In these algorithms, a DEVS atomic model is executed by assigning a simulator to it and to a DEVS coupled model a coordinator is assigned. From its original definition, the DEVS abstract simulator structure is hierarchical in nature and the hierarchy of models is mapped onto it (Figure 2(b)). The distinctiveness of the DEVS framework is in its hierarchical compositional structures, which help in complexity reduction. During simulation, the interaction/communication between different model components is achieved through event messages exchanged between the Simulators and Coordinators, each representing an event to be processed.

Two key pieces of information carried by these messages are their category and time stamp. The category of a message is associated to a certain form of treatment the receiver component (Coordinator or Simulator) must perform. The time stamp indicates the simulation time this message has been generated.

In CDEVS,[5] categories are *, i, x and y. In the first version of PDEVS,[6] categories are *, i, q, done, @ and y. The next versions propose various sets of categories (with associated sets of treatments). However, all DEVS-based algorithms adhere to the same simulation principle (i.e., generalized or specialized Coordinators and Simulators exchanging messages and performing specific actions on receipt of specific categories of message).

3. Key concepts

In this section we briefly introduce key concepts of the framework. They are as follows.

- Root Coordinator (RC): the simulating element that manages the time of a ST.
- Nodes: the simulation entities used for executing DEVS models. These nodes are Coordinators, Simulators and RCs. The RC has an event loop that sends event messages and controls the simulation cycles while the Coordinator and Simulator are capable of receiving, treating and sending event messages.
- ST: a tree is made up of nodes. The RC is always at the top of the tree's hierarchy and has a Coordinator as its descendant. Also, the Coordinator has either a Coordinator or Simulator as its descendant but the Simulator has none.
- Process: we define this as a stream of execution. It contains two types of nodes during execution: they are active and passive nodes. An active node is a

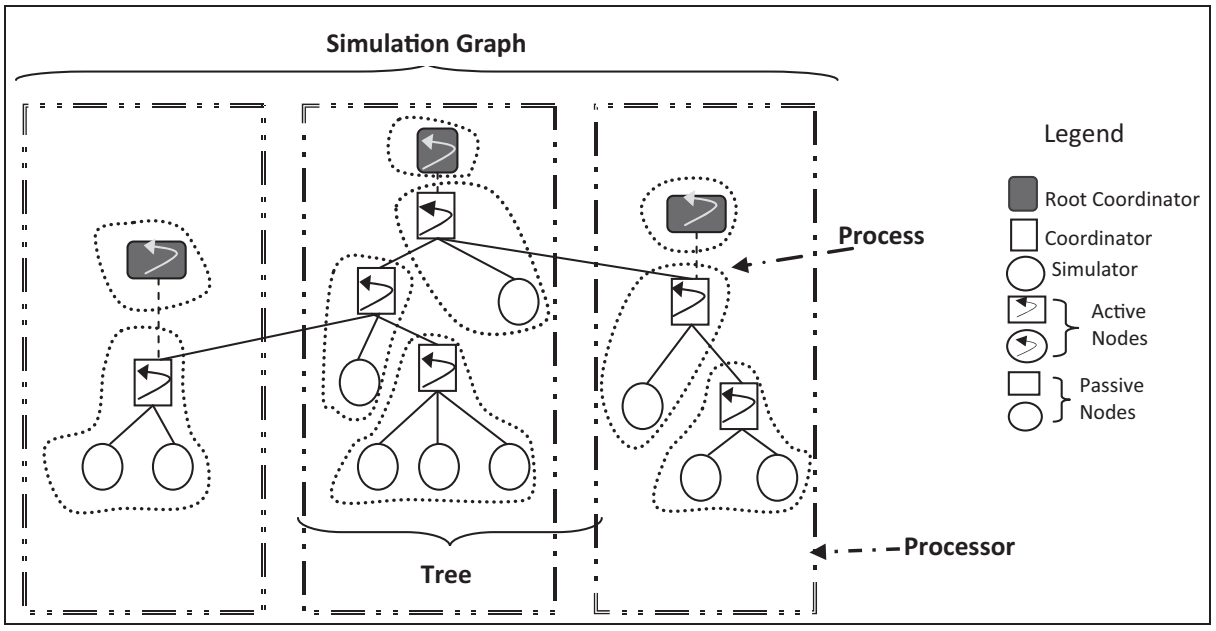


Figure 3. Relationship between Trees, Processes and Processors in a graph.

node that is currently active in an execution stream (e.g., Java threads or Ada tasks). While a passive node is part of an execution stream, it is not actively involved until it is triggered (e.g., function calling in Object Oriented Paradigm). We consider that a process would have at most one active node. If a process has more than one active node, those nodes are then regarded as being autonomous sub-processes. Also, there can be more than one passive node in a process.

- Activity: a set of actions that is performed at the receipt of an event.
- Processor: computing resource that allows the execution of a program (a process, an entire tree, any other executable code) on itself.
- Simulation Graph (SG): a representation of the relationship between the identified aspects in DEVS simulation. An example of a SG can be seen in Figure 3. Details about its components are discussed in the following sections.

We also formalize these concepts. Numerous formal definitions are given throughout the paper for the following reasons.

- They provide a clear understanding (by reducing ambiguity) of simulation structures and operations we introduce. We use definitions given in earlier sections to formalize concepts presented in later sections.
- They provide mathematical objects that one can use for consistency checking or validity checking.
- They can ease the automation of processes defined with them.

Definitions 1 and 2 given below will be used as building blocks to formalize simulation structures we will introduce later, as well as the generic operations that make up the framework. Although they are equivalent, one or the other definition is more convenient to use in specifying other concepts.

Definition 1: We formally define ST as $T = < R, N, f >$ *with*:

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

where:

R: The RC of the tree
N: The set of nodes of the tree
f: A function that maps a child node to its parent (the one at one step higher in the hierarchy).

For example, the tree given in Figure 2(b) is defined as $T = < R, \{R, C1, C2, S1, S2, S3\}, f >$, where $f(R) = \{C1\}$, $f(C1) = \{C2, S1\}$, $f(C2) = \{S2, S3\}$ and $f(S1) = f(S2) = f(S3) = \varnothing$.

Definition 2: ST can also be defined as $T = < R, N, F >$ *with*:

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

Using Definition 2 for the example of Figure 2(b), R and N will be defined as same while F will be $\{(R, C1), (C1, C2), (C1, S1), (C2, S2), (C2, S3)\}$.

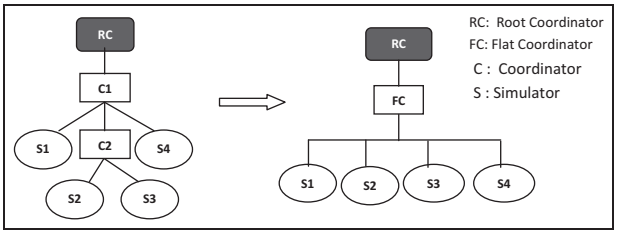


Figure 4. Tree Transformation by reduction.

4. Taxonomy in DEVS parallel and distributed simulation

There are different practices behind the concept of exploiting DEVS with PDES. Due to this, the concept becomes burdened with variances in opinions on how to build a DEVS simulator. We were able to identify four major factors in use in these practices. Firstly, some approaches alter the tree structure.[3,7–13] We call this ''Tree Transformation''. Secondly, most approaches split the model tree into many trees.[14–17] We call this ''Tree-Splitting''. Also, some approaches differ on the number of executions/processes that can be performed per simulation run.[18,19] We call this ''Node-Clustering''. Lastly, some approaches vary the number of computing resources to be used during simulation.[2,11] We call this ''Process-Distribution''. In the following sections we present and formalize these concepts. In doing so, we also offer a review of major strategies proposed in the literature.

4.1 Tree Transformation

It has been observed that altering a ST structure can improve simulation performance and also enhance distribution. This transformation is usually achieved either by reducing or increasing the number of nodes of a tree.

4.1.1 Reduction. As presented by Kim et al.,[7] the hierarchical structure of the simulator (which has a one-to-one correspondence with the DEVS model architecture) can increase the communication overhead between nodes. The process of reducing the number of these nodes on a tree is also known as flattening. A flattened simulator[8] simplifies the hierarchical simulator while keeping a hierarchical model structure (see Figure 4). Various studies[9,10] have shown that a flattened simulator reduces these costs. CD++ uses a flat simulation approach that eliminates the need for intermediate coordinators to improve the performance of simulation.[3,12] Some other approaches prefer to alter the compositional structure of a DEVS model. Kim and Wang[11] proposed transforming a hierarchical DEVS model into a non-hierarchical structure to ease synchronization in a distributed simulation. Zeigler et al.[2] also considered building Conservative DEVS simulator for non-hierarchical models.

4.1.2 Expansion. Himmelspach et al.[13] achieved the expansion by introducing new simulation nodes into the ST structure, as presented in Figure 5. This is to enable the distribution of nodes on different processors. The introduction of extra components on the tree introduces more concerns as to what type of information each of these new components should contain. Also, communication between these nodes constitutes an increasing overhead cost as the structure of the messages being passed is altered to accommodate extra information. For example a new sub-coordinator has no coupled model associated with it and therefore contains no coupling information. Also, it has to correctly identify imminent models and influences. One way to deal with this is through the composition of messages, that is, by including more information in a message's construct, as seen in Himmelspach et al.[13] In PCD++ [20,21] the inclusion of Node Coordinators (NCs) in the ST was to enable synchronization and communication between processes in a distributed environment.

4.1.3 Formal Specification for Tree Transformation.

Definition 3: Formally we define Tree Transformation as

Transf[Na,Nr,Fa,Fr]: $\tau \rightarrow \tau$ *where* τ *is the set of all possible STs*

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

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

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

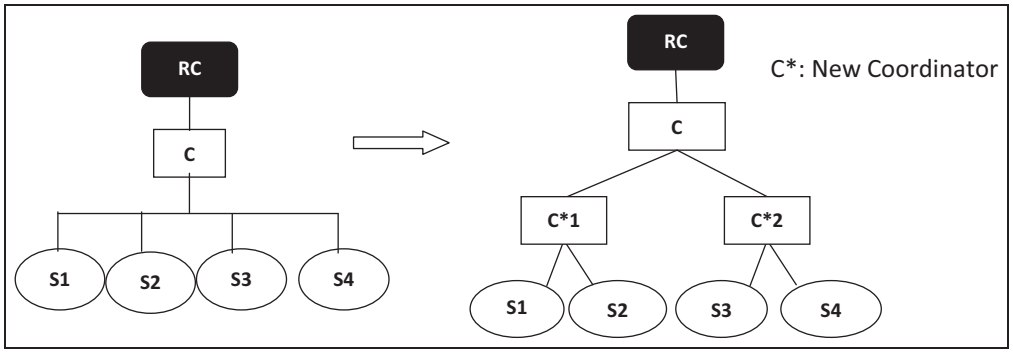


Figure 5. Tree Transformation by expansion.

Based on the following conditions:

$Na \cap N = \varnothing$ (new nodes should not belong to the old tree)
$Nr \subset N - \{R\}$ (only nodes of the old tree can be removed excluding Root)
$Fa \subset (N \times Na) \cup Na^2 \cup Na \times (N - \{R\})$ (a new parenthood must exist either between an old and a new node or between two new nodes or between a new and an old node)
$Fr \subseteq F$ (only parenthood of the old tree can be removed)

4.2 Tree Splitting

Tree Splitting can be referred to as the decomposition of a simulator tree to form sub-trees based on the analysis of the model's structure. We identified two types, namely the single tree structure and the multiple tree structure. It is necessary to state here that this section does not deal with how the tree structure can be split or executed, or how they can be mapped to the available number of processors.

4.2.1 Single tree structure. In describing this structure, executing a model with a single tree structure can be expressed as having an entire model tree simulated with the use of a central scheduler called the RC.

Single tree structures are mostly implemented using CDEVS and PDEVS algorithms. In CDEVS,[2] events are processed in a sequence. This approach is the simplest form of simulation but it does not properly reflect the simultaneous occurrence of events in the system being modeled. Indeed, serialization reduces possible utilization of parallelism during the occurrence of events. On the other hand, Chow and Zeigler[6] introduced PDEVS as a possible solution to the problem of serialization. According to Chow and Zeigler, one desirable property provided by PDEVS is the degree of parallelism that can be exploited in parallel and distributed simulation. It beats the restrictions in CDEVS in both execution time and memory usage.

4.2.2 Multiple trees. We look at the multiple tree structure as when a tree can be split into different sub-trees with each having its own central scheduler/RC and different simulation clocks. This is the preferable solution in distributed simulation. Based on this structure, all events with the same time stamp are scheduled to be processed simultaneously. Distributed simulation algorithms are used to synchronize trees.

The two basic distributed simulation algorithms in use are the Optimistic[17] and Conservative (Pessimistic)[14–16] algorithms. Optimistic algorithms, in contrast to Conservative algorithms, enable increased degrees of parallelism. However, they also result in more complex algorithms. Communication between these trees is usually between:

a. RC and RC (see Figure 6(a)), for example, DEVS Time Warp (TW);[2]
b. Coordinators and Simulators (Parents and their Children in the initial tree), for example, the Distributed Optimistic Hierarchical Simulation (DOHS) scheme[7] (see Figure 6(b)).

4.2.3 Formal specification for Tree Splitting.
Definition 4: Formally we define Tree Splitting as
$Split: \tau \rightarrow \Sigma$
$Split(< R, N, f >) = < \{R_i\}, N', f' >$
where

Σ is the set of all simulation skeletons (see the definition of this structure in section 5)
$R \in \{R_i\}$
$N' = N \cup \{R_i\}$
$f'_{/N} = f$

4.3 Node clustering

Node clustering is the association of one or many nodes to various processes. Events execution is driven through the

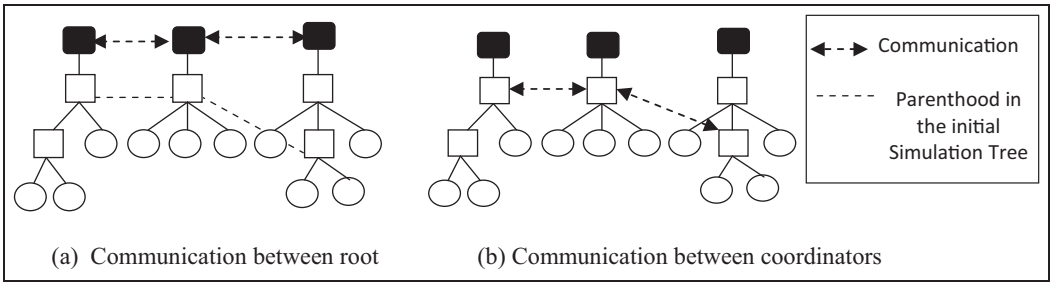


Figure 6. Communication schemes between tree structures.

use of processes. We take a look at the concept of process as an execution stream. A process can be seen as a mechanism that is able to execute events. We categorize based on the number of processes: as ''one process'' execution or ''many processes'' execution. However, we will not be dealing with how execution takes place on processors.

4.3.1 One process. A one-process execution denotes having events processed in a serially and orderly manner, that is, one after another. This restricts concurrent execution streams. Himmelspach and Uhrmacher[18] denote this form of execution as ''sequentialization''. In this sense, for example, the ''main'' program is a process. A desired speed up may not be achieved when using a one-process execution stream. On the other hand, it is easier and faster to implement. During the one-process execution, interaction between the nodes is called intra-process communication. Most implementations based on CDEVS make use of the one-process type of execution stream.

4.3.2 Multiple processes. In the case of many processes, execution of events can be split into several logical processes (autonomous tasks) for concurrent processing. Examples of such processes are Java threads, POSIX threads, Ada tasks and so on.

Using many processes could speed up execution, as each could execute events without interrupting other processes. However, this is balanced by the increase of memory consumption and the burden of communication between processors. This type of communication is called inter-process communication. It is possible that during a simulation run only one process, out of many, is scheduled for execution. This situation is called pseudo-parallelism; otherwise it is pure-parallelism. During implementation, it is essential to manage how processes access resources that are common to all of them (e.g., shared data type). Locks, Semaphores, Monitors and other synchronizing mechanisms can be used to coordinate these processes. The CCD++ implementation utilizes many processes for model execution.[19]

4.3.3 Formal specification for Node Clustering.
Definition 5: Formally we define Node Clustering as

$$Cluster: N \rightarrow Ps$$

where

Ps is the set of Processes.
$(Cluster)^{-1}(p)$ *is Connex* $\forall p \in Ps$
$\forall p_i, p_j \in Ps, p_i \neq p_j, Cluster^{-1}(p_i) \bigcap Cluster^{-1}(p_j) = \varnothing$

4.4. Process distribution

Process distribution can be referred to as the allocation of one or many processes to the available number of processors. We considered that the number of processors play a major role in speed, performance and efficiency that can be achieved during simulation. We therefore categorize this into two distinct classes: ''one-processor'' or ''many-processors''.

4.4.1 One processor. On a uniprocessor system, the entire simulation runs on one processor so there is no overhead cost but it is limited to the size of the memory in use. Thus, it is not completely suitable for executing complex models. The type of communication that takes place in this case is called an intra-processor communication.

4.4.2 Multiple processors. In order to coordinate simulation on many networked processors, some form of inter-processor communications is required to convey data between processors and synchronize each processor's activities. When utilizing multiple processors for simulation, the memory architecture type could either be shared memory (processors have direct access to common physical memory), or distributed memory. Meanwhile, in shared memory only one processor can access the shared memory, hereby introducing the need to control access to the memory through synchronization. Distributed memory refers to the fact that the memory is physically distributed as well. Memory access in shared memory is faster but it is limited to the size of the memory. Therefore, increasing the number of processors without increasing memory size can cause severe bottlenecks. Inter-processor communications

is usually achieved through interoperability mechanisms (e.g., CORBA,[22] or more recently Web Services[23,24]).

As a consequence of using more than one processor, the nodes can be split into a set of partition blocks based on certain decision criteria and mapped unto the available number of processors. This is called partitioning. In the case of no partition, simulation is performed on a single processor machine. The partitioning problem is one of the most important issues in parallel and distributed simulation as it directly affects the performance of the simulation. Different partitioning algorithms have been proposed. An example is the Generic Model Partitioning (GMP) algorithm proposed by Park et al.[25] It uses cost analysis methodology to construct partition blocks, although it makes an effort to guarantee an incremental quality of partitioning. However, it is restricted only to models from which cost analysis can be extracted and processed.

4.4.3 Formal specification for Process Distribution.

Definition 6: Formally we define Process Distribution as

$Distrib: Ps \rightarrow Pr$

where

Pr *is the set of Processors*

$\forall p_i, p_j \in Pr, p_i \neq p_j, Distrib^{-1}(p_i) \bigcap Distrib^{-1}(p_j) = \varnothing$

4.5 Simulation graph strategies

Due to the increasing complexity and size of models, various studies have been conducted to improve efficiencies and performances of DEVS simulators,[2,9,10,12,13,20,21] thus giving rise to various graph strategies. In a general overview, most implementation decisions have been observed to be based on the presented aspects in the previous sections. In this section we use figures to illustrate how these aspects are interrelated with one another using a three-categorized view: the Processor-Tree-Process view. Since the Tree Transformation and Tree Splitting aspects both focus on the tree, they will be represented as the same category, that is, the Tree. In each of these categories, the number of elements, that is, Trees, Processes or Processors is put into consideration. This therefore forms the basis for development of any SG Strategy presented.

4.5.1 Single processor – single tree – single process. This is the simplest form of mapping strategy that has one RC (one tree) controlling the simulation on one processor (see Figure 7(a)). Execution of events is purely sequential with no need to synchronize communication between the nodes. In this case, when simultaneous events occur, one event is selected and others are ignored thereby introducing rigidity during execution. PythonDEVS[26] uses this mapping strategy as an implementation of the CDEVS formalism and, as a consequence, it performs sequential simulation.

4.5.2 Single processor – single tree – multiple processes. The entire simulation depends on one RC while execution is through the use of many processes, as shown in Figure 7(b). These processes run concurrently and are mostly used to increase execution speed, but as the number of processes increase the rate of memory consumption increases, thereby slowing down execution and time.

This strategy was proposed for use in Abstract Threaded Simulator.[18] However, depending on the memory size of the processor and the model size, the cost of creating threads becomes expensive as the number of models increases. This is a critical factor to be considered when using many processes.

4.5.3 Single processor – multiple trees – single process. Several trees or RCs exist on one processor with each performing sequential execution one at a time. An example is shown in Figure 7(c). This scenario is not realistic because execution is asynchronous and can be done simultaneously using many processes instead.

4.5.4 Single processor – multiple trees – multiple processes. Several RCs or trees exist on a partition (as seen in Figure 7(d)). This makes it easy to implement a synchronization mechanism (optimistic or pessimistic) for dealing with causality errors. Causality errors usually occur when messages are not processed in a time-stamp order.[1] Communication between different trees can be made via the RCs or coordinators and simulators. This strategy brings about the idea of federating existing abstract simulators. Since all the trees are on one processor there is intra-processor communication thereby eliminating the need for an interoperability technology.

4.5.5 Multiple processors – single tree – single process. A RC controls the entire simulation on multiple partitions, while execution is sequential. An example of this is shown in Figure 7(e). The sequential execution can only take place locally, that is, when a simulating node receives a message from its parent on the same processor. This strategy is not realistic, since nodes on different processors require different execution streams. Therefore, a single process is not enough to cover the entire simulation execution.

4.5.6 Multiple processors – single tree – multiple processes. One RC controls the entire simulation on multiple partitions but with multiple processes. There are two types of communication between the nodes, that is, locally (intra-processor) and remotely (inter-processor). At the local level, communication is between nodes on the same

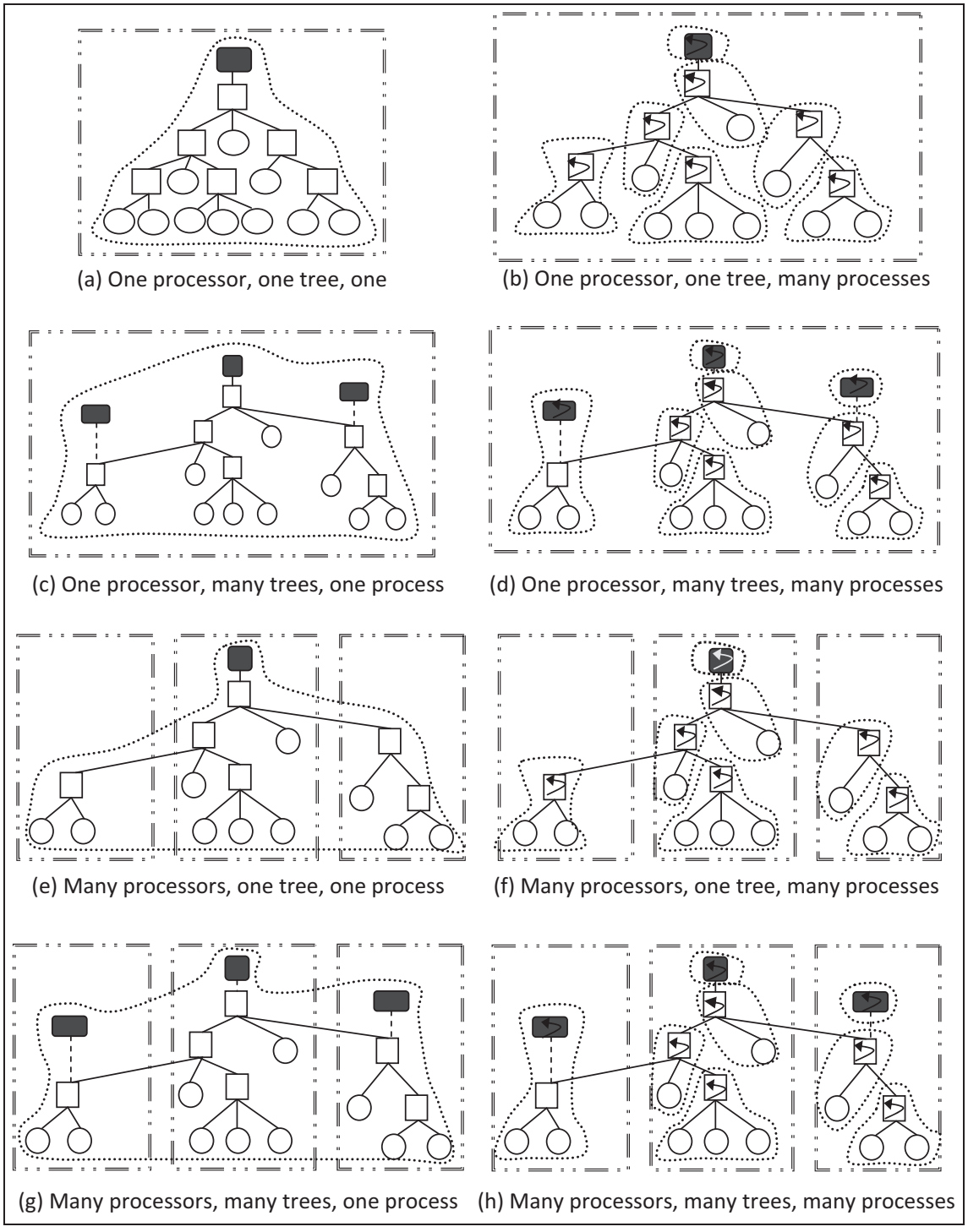


Figure 7. DEVS Parallel Discrete-Event Simulation strategies.

processor. Remote communication is achieved through the use of interoperability technologies. This is the case of the abstract simulator proposed by Himmelspach et al.[13] Also, as seen in Triccoli and Wainer,[27] when running parallel and distributed simulations, the entire ST is divided among a set of processes, each of which will execute on a different processor. In general terms, each process will host one or more simulation nodes, as shown in Figure 7(f).

4.5.7 Multiple processors – multiple trees – single process. This case, which concerns multiple partitions

(each containing at least a tree running independently, as shown in Figure 7(g)), is also not feasible for the same reasons mentioned in Section 4.5.3.

4.5.8 Multiple processors – multiple trees – multiple processes. As shown in Figure 7(h), each partition contains at least one tree and several executions at the same time. Each tree implements a synchronization mechanism for causal errors (because each tree has its own clock). Communications between partitions are either between the RCs (as in Zeigler et al.[2]) or between the coordinators and simulators (between ascendants and descendants as in Kim and Wang[11]) via distributed simulation middleware such as high-level architecture (HLA).[28,29]

Optimistic and Conservative strategies are synchronization techniques used for PDES in general. The conservative approach is the first synchronization algorithm that was proposed in the late 1970s by Bryant,[14] and Chandy and Misra.[15] It is also known as the Chandy–Misra–Bryant (CMB) algorithm and strictly avoids the possibility of processing events out of time-stamp order. In contrast, the optimistic approaches, introduced by Jefferson's TW protocol,[17] allow causality errors to happen temporarily but provide mechanisms to recover from them during execution. The first attempt to combine DEVS and TW mechanisms for optimistic distributed simulation is DEVS-Ada/TW.[30] It is an asynchronous approach that uses the TW mechanism for global synchronization; it treats all nodes on one processor as one process. In DEVS-Ada/TW, the hierarchical DEVS model can be partitioned at the highest level of the hierarchy for distributed simulation. As a consequence, the flexibility of partitioning models is restricted.

The DOHS scheme[7] is a method of distributed simulation for hierarchical and modular DEVS models that uses the TW mechanism for global synchronization.

A proposal was made by Zeigler et al.[2] to combine TW with the CDEVS hierarchical simulator as the TW DEVS Simulator. In this approach, the overall model is distributed so that each sub-model is a single coupled model. Then, hierarchical execution is done locally on each processor using the classic abstract simulator with extensions for state saving and restores (rollback). In addition, each processor has a RC that realizes the mechanism for TW. On each processor, the RC performs optimistic TW synchronization. For this, it stores the input and output messages of the processor and takes care of anti-messages.

The ''Risk-Free'' Optimistic Simulator[2] is another version of optimistic DEVS simulator. The operational semantics of the ''risk-free'' version of optimistic DEVS is based on optimistic DEVS TW. The TW optimistic simulator and coordinator are used without change but the synchronization mechanism in the optimistic RC changes. Local events on each compute node (tree) are processed sequentially but optimistically. That is to say if a straggler event is received by a node, a rollback occurs but is local to that node. This is less costly when compared to TW.

In parallel versions of CD++ (PCD++[20,21] and CCD++[19,31]), the authors proposed a distributed simulation architecture for DEVS and Cell-DEVS models. These variants use the flattening structure of simulators with four kinds of simulation nodes on each tree: Simulators, the Flat Coordinator (FC), NC and RC. The RC is created on one of the processors to start/end the simulation process and perform input/output (I/O) operations. The FC and NC are created on each processor. The FC is in charge of intra-processor communication between its children Simulators. The NC is the local central controller on each processor. The Simulator executes the DEVS functions defined in its atomic model. In Section 5, we will use this specific strategy to illustrate how our conceptual framework can apply.

4.5.9 Tree – process – processor notation. To briefly explain some of the strategies in the literature in a more formal way we suggest the tree – process –processor notation. It consists of defining the number of elements for each aspect of PDES. We use *N* for ''many elements''. For example, a 1-1-1 scheme is a DEVS SG strategy with 1 Tree, 1 Process and 1 Processor, while *N*-*N*-1 is a DEVS tool with many Trees, many Processes and 1 Processor. For example, PythonDEVS[26] is a 1-1-1 strategy, using CDEVS. The Abstract Threaded Simulator of the James II[18] package uses a 1-*N*-1 strategy with its processes created using Java threads. The Parallel CD++ Simulator[27] and the Parallel Sequential Simulator,[13] which implements the PDEVS formalism, use the 1-*N*-*N* strategy. The Conservative CD++[31] is an *N*-*N*-*N* strategy. Some other approaches that use the *N*-*N*-*N* strategy include DEVS-Ada/TW,[30] DOHS scheme[7] and Optimistic Parallel CD++.[20] A DEVS tool that uses the *N*-*N*-*N* strategy fully supports distributed simulation. A state of the art is given in Table 1.

We identify that a SG strategy for an implementation differs from another. However, we state here that our definitions of the components used in SG construction are at an abstract level. The links and nodes in a SG strategy are seen as abstract, hence they can be implemented in different ways either by using the language in which it was implemented or by using interoperability technologies (if simulators are implemented in different languages). The only constraint we define is that the simulators implement the DEVS simulation algorithm. Thus the issue of DEVS implementation differences can be overcome.

5. Unifying the DEVS Parallel Discrete-Event Simulation framework

A unifying framework is needed to harness the identified components and their operations (found in the SG) in a

Table 1. An overview of major Simulation Graph strategies.

Approaches	Algorithm	Strategy
PythonDEVS[26]	CDEVS	1-1-1
Abstract Threaded Simulator[18]	PDEVS	1-*N*-1
Parallel Sequential Simulator[18]	PDEVS	1-*N*-*N*
PCD ++ [27]	PDEVS	1-*N*-*N*
Optimistic PCD ++ [20]	Optimistic DEVS	*N*-*N*-*N*
CCD ++ [19]	Conservative DEVS	*N*-*N*-*N*
Risk-Free Optimistic DEVS Simulator[2]	Optimistic DEVS	*N*-*N*-*N*
Time Warp DEVS Simulator[2]	Optimistic DEVS	*N*-*N*-*N*
DEVS-Ada/TW[30]	Optimistic DEVS	*N*-*N*-*N*
DOHS[7]	Optimistic DEVS	*N*-*N*-*N*

DEVS: Discrete Event System Specification; DOHS: Distributed Optimistic Hierarchical Simulation; CDEVS: Classic DEVS; PDEVS: Parallel DEVS.

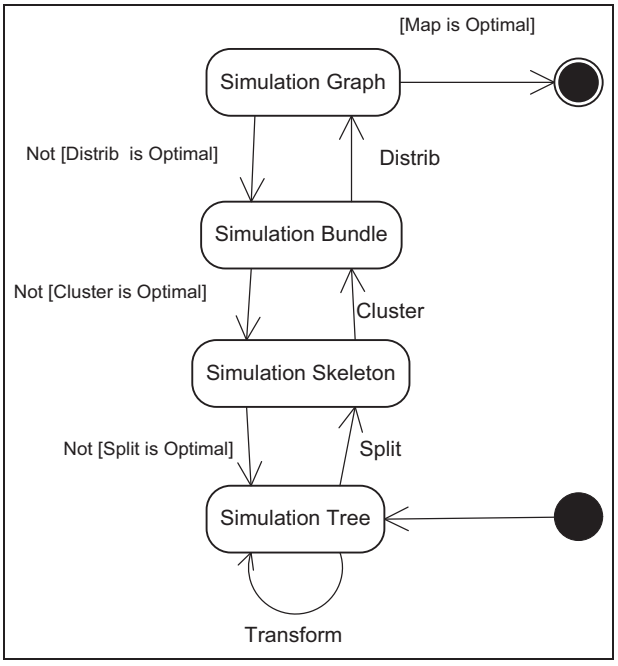


Figure 8. Simulation graph methodology.

bid to automate the process of building a DEVS simulator and performing simulation by using DEVS with PDES. We interpret the building of a DEVS PDES Simulator as a move from the original ST to a SG. Having taken a look at the possible SG strategies, in this section we propose a methodology for building a SG. A SG is obtained through the depictions of the relationship and the components of DEVS PDES, that is, Trees, Processes and Processors. This methodology thus describes a structural and behavioral view in exploiting DEVS PDES.

5.1 From Simulation Tree to Simulation Graph

We propose using a layered approach to present the simulation structures involved in DEVS PDES to avoid the pitfalls inherent to the building of a DEVS simulation system suitable for Parallel and Distributed execution. By using a state chart, we present the trajectories that describe the set of all possible paths that can be taken during the construction of the SG. Such state chart can be seen as a metamodel of the process of mapping the ST onto a SG.

Any user-defined SG construction process can be depicted as an instantiation of the state chart given in Figure 8, driven by the analysis of the initial ST and the available number of Processes and Processors. The methodology allows iterating on each state until some user-defined satisfaction criteria are reached (optimal splitting, optimal clustering and optimal distribution, which we specify in Figure 8 respectively as [Split is Optimal], [Cluster is Optimal] and [Distrib is Optimal]).

As previously introduced and formally specified in Definitions 3–6, *Split* is a generic function that is used for creating a partition of nodes from a *ST*. The *Cluster* generic function takes the available number of nodes and associates them with *Processes*. While the *Distrib* generic function takes the set of available *Processes* and plots them onto the set of available *Processors*. Also, the *Transform* generic function alters the *ST* structure either by expansion or reduction. This altering is done on the number of available nodes (not including the RCs) on the tree and their relationships. The process of *Transformation, Splitting, Clustering* and *Distribution* can iterate until it is sure that a good performance will be gained during simulation from the new *SG*.

The *Simulation Skeleton* is the structure obtained from splitting a ST. The *Simulation Bundle* is a collection of clusters of nodes. Examples are shown in Figures 9(a) and (b), respectively.

We formally define each of the Simulation Structures that are found in this methodology (the definitions for the ST structure are found in Definitions 1 and 2). To have a complete definition we make reference to Clustering and Distribution functions that have been formally defined in Definitions 5 and 6, respectively, while definitions for the Transform and Split functions are given in Definitions 3 and 4.

Definition 7: A Simulation Skeleton is formally defined as

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

where

$R_i \in N \ \ \forall i$

$f: N \rightarrow \wp(N)$, *where* $\wp(N)$ *is Power Set of* N

$f^{-1}(R_i) = \varnothing, \ \forall i$

$f^{-1}(J) \neq \varnothing, \ \forall J \in N - \{R_i\}$

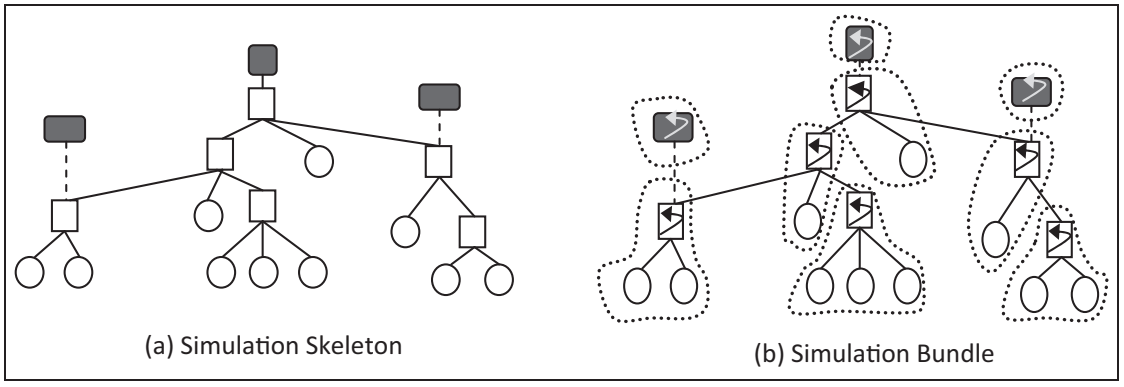


Figure 9. Intermediary simulation structures from Simulation Tree to Simulation Graph.

Definition 8: A Simulation Bundle is formally defined as
$B = < \{R_{ij}\}, N, f, Ps, Cluster >$
where

$< \{R_{ij}\}, N, f >$ *is a skeleton*
Ps is the set of Processes
Cluster: $N \rightarrow Ps$

Definition 9: A SG is formally defined by
$SG = < \{R_{ij}\}, N, f, Ps, Pr, Cluster, Distrib >$
where

$< \{R_{ij}\}, N, f, Ps, Cluster >$ *is a Simulation Bundle*
Pr is a set of Processors
Distrib: $Ps \rightarrow Pr$

5.2 Case study

To illustrate the use of our methodology, we consider an existing work from literature as an example. PCD++ [20,21] consists of simulation nodes that are concrete implementations of abstract DEVS simulators. We describe how an instantiation of our generic approach can provide a formal guideline for the process of mapping a DEVS simulation protocol onto the PCD++ strategy.

We consider a fire propagation model ($n \times n$ cells) as the one presented by Liu and Wainer[20] and Glinsky and Wainer[21] and the distributed simulation strategy introduced as well. We assume the initial DEVS model (without the couplings) and DEVS tree structure in use are as shown in Figure 10.

In PCD++, we see that the first operation is to reduce (transform) the number of simulation nodes on the tree, hence the introduction of FCs. Each FC synchronizes its child simulators, routes messages among them and is in charge of intra-process communication between its children. The approach of reducing the simulation nodes was used to achieve improved simulation performance. At the end of this operation the resultant simulation structure is also a ST. The formal definition of the transformation done includes the formal algorithms of the nodes involved in both sides of the transformation (i.e., initial DEVS algorithms for coordinators at one side, and new algorithms for FCs at the other side). Figure 11 presents the resulting tree.

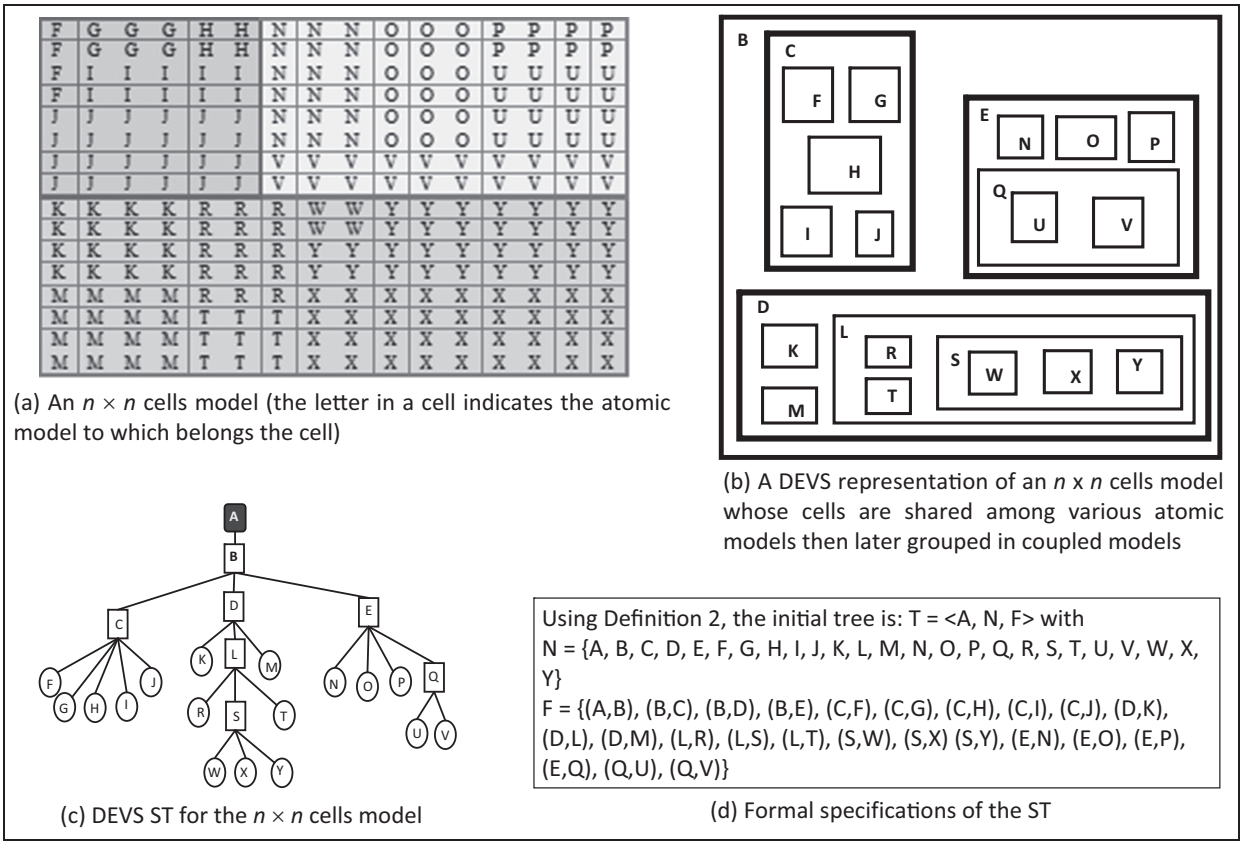
The next operation to be performed is a *Split* operation. It takes the transformed tree and partitions it into several sub-trees with a node coordinating the timing on each sub-tree. In PCD++ the NC has been introduced to be the local central scheduler on each sub-tree. The RC shares its time scheduling capabilities with a NC. While the Root starts the simulation and performs I/O operations, the NC schedules simulation time and manages inter-process communication. The resultant structure is a Simulation Skeleton, as shown in Figure 12.

In PCD++, there is at most one of each of NCs and FCs clustered into a process called the Logical Process. We see that the authors varied the number of processes/processors for the purpose of checking the performance of PCD++. We, however, assume the use of three processes in this example. Figure 13 shows the Simulation Bundle resulting from the *Cluster* operation performed.

In PCD++, the constraint used is to permit only one Logical Process on a Processor, thereby allowing intra-Logical Process communication via the NC. The FC is in charge of communication between its child Simulators. The final SG is obtained by mapping each Logical Process onto a Processor, as shown by Figure 14 and suggested by Liu and Wainer[20] and Glinsky and Wainer.[21]

6. Discussion

The most targeted benefit of the framework proposed is to give a straight and clear guideline for realizing DEVS PDES by avoiding some common pitfalls. One is in trying to distribute the model instead of the simulation (this confusion is due to the fact that most simulations do not separate clearly the concerns). The taxonomy makes it clear what should be distributed, that is, the simulation protocol



(a) An $n \times n$ cells model (the letter in a cell indicates the atomic model to which belongs the cell)

(b) A DEVS representation of an n x n cells model whose cells are shared among various atomic models then later grouped in coupled models

(c) DEVS ST for the $n \times n$ cells model

(d) Formal specifications of the ST

Figure 10. Initial DEVS $n \times n$ cells model, its Simulation Tree (ST) structure and formal definition.

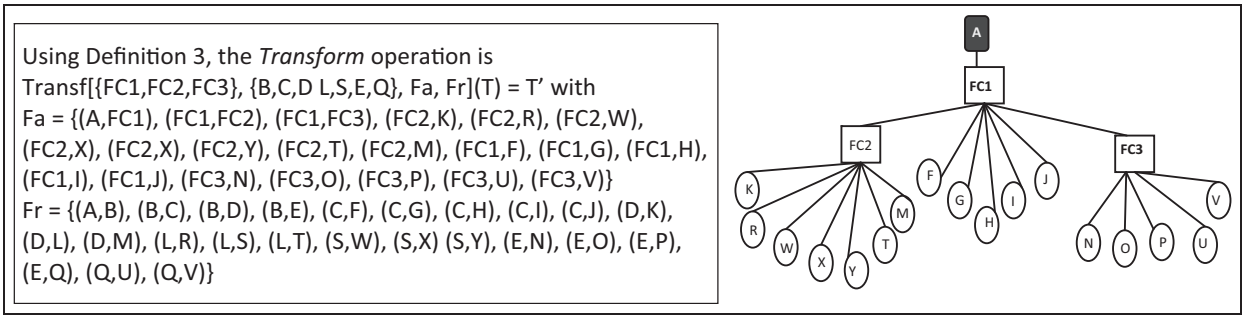


Figure 11. Simulation Tree transformation by *reduction*.

and not the simulation model. However, a priori knowledge about a model's internal behavior can be tremendously helpful for efficiently building the PDES strategy. Such knowledge should be carried by the strategy builders into their initial ST such that it can influence the definition of the Transform, Split, Cluster and Distrib operations involved in producing their final SG.

When defining the SG, another common pitfall is to understand the partitioning as a one-to-one mapping between simulation nodes and processes. The taxonomy makes clear the difference between the nodes that represent the simulation components and the way they can be aggregated at the implementation level as part of a single process (e.g., the same thread can implement a coordinator and its children simulators). For example, trying to build a SG by adopting a 1-1-1 strategy will lead to implementing the whole ST as the process (which implies that all nodes of the ST are implemented as passive nodes except the RC, which will be the main program).

From an existing distributed code, it is difficult to build a new distributed solution that must implement a different SG strategy. An example is the building of a new *N-N-N*

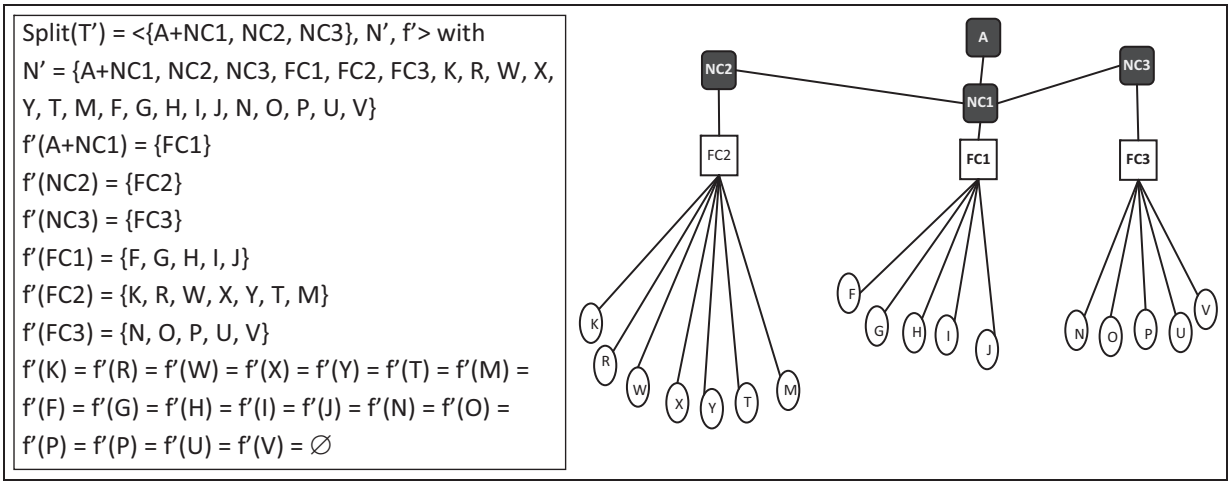


Figure 12. Intermediary simulation skeleton.

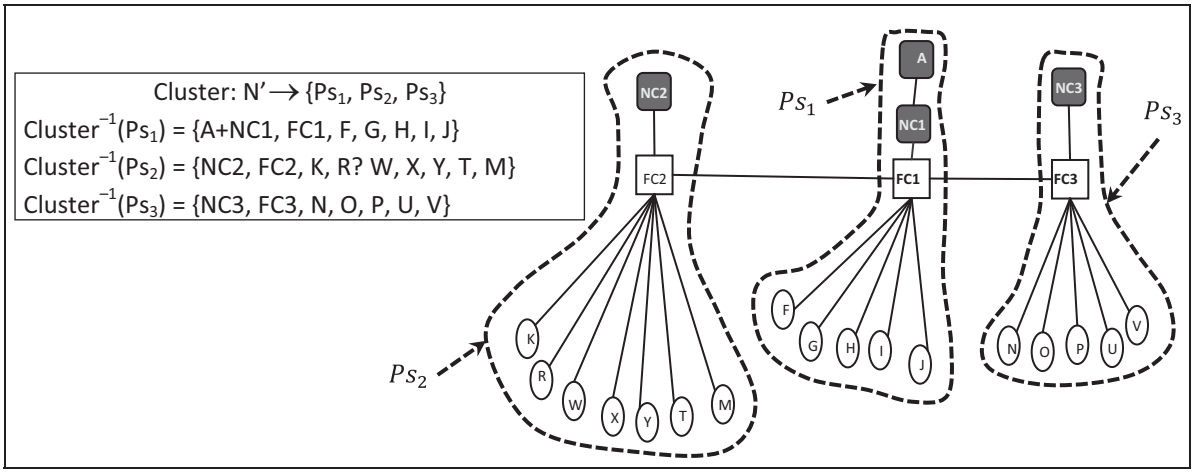


Figure 13. Intermediary simulation bundle.

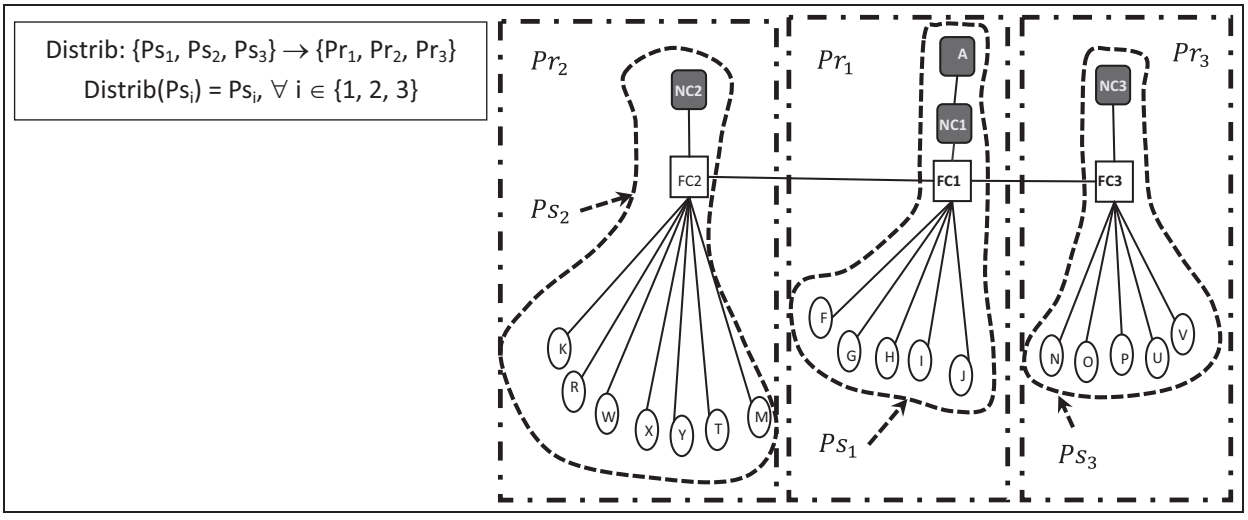


Figure 14. Simulation Graph obtained applying the PCD ++ strategy.

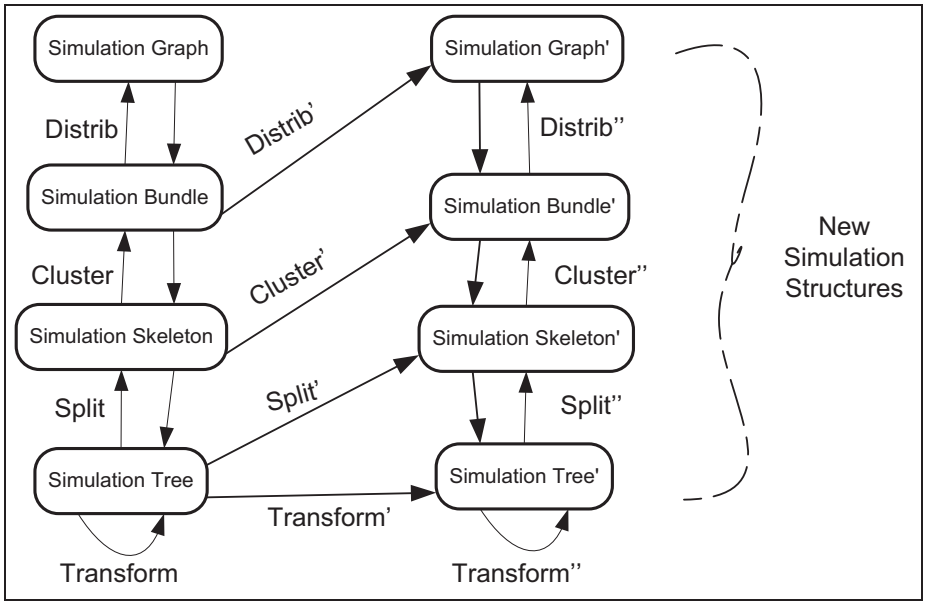


Figure 15. Flexibility of the layered approach.

strategy from an existing *N-N-N* strategy by increasing or decreasing the number of Trees/Processes/Processors or by reallocating differently Processes (respectively nodes) to Processors (respectively Processes). Our framework provides the way for specifying, comparing and contrasting various implementations and offers a standardized platform for all DEVS PDES methodologies. To modify the SG for adaptation to a new computing architecture, starting from the SG layer, there would be a need to move to the layer below it or back to the ST before making the necessary modification in the Simulation Structure (see Figure 15). Such disciplined approach makes it easy to avoid mistakes.

The lack of a unified approach for DEVS PDES development can hamper the easy integration of DEVS PDES simulators. For example, combining two existing trees (or combining two existing SGs or an existing ST with an existing SG) to get a final SG is not obvious. Also, with the various practices involved in building DEVS PDES simulators, it is difficult to evaluate each design and implementation on the same basis. The proposed approach offers a common frame of reference for all DEVS PDES simulators. Failure or inability to facilitate consistency while specifying a Simulation Structure indicates lack of focus on the essential elements of the taxonomic framework and lack of understanding of the value that each structure contributes within the framework.

7. Conclusion

This paper has presented a conceptual framework for DEVS PDES implementation strategies. It aims at contributing to a better understanding of the process to drive and the simulation structures involved. It introduces a taxonomy, formalized concepts and a methodology consisting of a meta-model of the process of mapping a DEVS ST onto a DEVS SG. As such, it proffered an abstract way for integrating heterogeneous DEVS implementation strategies by providing a systematic and quantifiable generic approach that can be instantiated to fill the gap identified from taking a DEVS simulation model specification to mapping it onto a parallel/distributed infrastructure. Each instantiation provides a guideline to applying a specific strategy.

There has been an on-going research effort in providing standard representation of DEVS to support common understanding, sharing and interoperability.[29] Decisions that are to be taken while implementing a DEVS PDES simulator include how events should be processed, what simulation architecture to use, what should be the organizational architecture, and so on. As these decisions may often be quite challenging, we therefore proposed a solution by going through a review of major existing approaches, categorized them and came up with an integrative view producing a formally specified unified taxonomic framework. All practitioners dealing with building DEVS PDES simulators can now choose to use this new layered approach as a reference framework for specifying their approaches unambiguously, then fostering knowledge reuse across the community. As a consequence, we envision that this framework contributes to the DEVS standardization efforts.[32,33]

Another benefit we see with the conceptual framework is that it also opens the way to automated code synthesis

of the simulation structures involved. We are developing a software tool called SimStudio[34] to support it.

Lastly, all guidelines produced by instantiating the proposed meta-model may be amenable to formal checking, because of the formalization of concepts and operations.

Funding

This research received no specific grant from any funding agency in the public, commercial or not-for-profit sectors.

References

1. Fujimoto RM. *Parallel and distributed simulation systems.* New York: Wiley, 2000.
2. Zeigler BP, Kim TG and Praehofer H. *Theory of modeling and simulation.* 2nd ed New York: Academic Press, 2000.
3. Wainer G. *Discrete-event modeling and simulation: a practitioner's approach.* New York: CRC Press, 2009.
4. Balci O. The implementation of four conceptual frameworks for simulation modeling in high-level languages. In: *proceedings of the 20th conference on winter simulation (WSC)*, San Diego, CA, 1988, pp.287–295.
5. Zeigler BP. *Theory of modeling and simulation.* New York: Wiley-Interscience, 1976.
6. Chow AC and Zeigler BP. Revised DEVS: a parallel, hierarchical, modular modeling formalism. In: *proceedings of the 26th conference on winter simulation (WSC)*, Lake Buena Vista, FL, 1994, pp.716–722.
7. Kim KH, Seong YR, Kim TG, et al. Distributed simulation of hierarchical DEVS models: hierarchical scheduling locally and time warp globally. *Trans Soc Comput Simulat Int* 1996; 13: 135–154.
8. Jafer S and Wainer G. Flattened conservative parallel simulator for DEVS and Cell-DEVS. In: *proceedings of the international conference on computational science and engineering (CSE)*, Vancouver, Canada, 2009, pp.443–448.
9. Glinsky E and Wainer G. Performance analysis of real-time DEVS models. In: *Proceedings of the 34th conference on winter simulation (WSC)*, San Diego, CA, 2002, pp.588–594.
10. Kim K, Wang K, Sagong B, et al. Efficient distributed simulation of hierarchical models: transforming model structure into a non-hierarchical one. In: *proceedings of the 33rd annual simulation symposium*, 2000, p.227.
11. Kim K and Wang W. CORBA-based, multi-threaded distributed simulation of hierarchical DEVS models: transforming model structure into a non-hierarchical one. In: *proceedings of international conference on computational science and its applications*, Assisi, Italy, 2004, pp.167–176.
12. Wainer G. CD++: a toolkit to DEVS. *Software, Practice and Experience* 2002; 32: 1261–1306.
13. Himmelspach J, Ewald R, Leye S, et al. Parallel and distributed simulation of parallel DEVS models. In: *proceedings of spring simulation multi-conference*, San Diego, CA, 2007, pp.249–256.
14. Bryant RE. *Simulation of packet communication architecture computer systems.* Technical Report MIT-LCS-TR-188. 1997. Cambridge, MA: Massachusetts Institute of Technology.
15. Chandy KM and Misra J. Distributed simulation: a case study in design and verification of distributed programs. *IEEE Trans Software Eng* 1979; SE-5: 440–452.
16. Misra J. Distributed discrete-event simulation. *ACM Comput Surv (CSUR)* 1986; 18: 39–65.
17. Jefferson DR. Virtual time. *ACM Trans Program Lang Syst* 1985; 7: 405–425.
18. Himmelspach J and Uhrmacher AM. Sequential processing of PDEVS models. In: *proceedings of the 3rd EMSS*, Barcelona, Spain, 2006, pp.239–244.
19. Jafer S and Wainer G. Conservative DEVS – a novel protocol for parallel conservative simulation of DEVS and Cell-DEVS models. In: *proceedings of spring simulation multi-conference (DEVS symposium)*, Orlando, FL, 2010, pp. 168–175.
20. Liu Q and Wainer G. Parallel environment for DEVS and Cell-DEVS models. *Simulation* 2007; 83: 449–471.
21. Glinsky E and Wainer G. New parallel simulation techniques of DEVS and Cell-DEVS in CD++. In: *proceedings of the 39th annual symposium on simulation*, Huntsville, AL, 2006, pp.244–251.
22. Chow YW, Hu X and Zeigler BP. The RTDEVS/CORBA environment for simulation-based design of distributed real-time systems. *Simulation* 2003; 79: 197–210.
23. Mittal S, Risco JL and Zeigler BP. DEVS-based simulation web services for net-centric T&E. In: *proceedings of the 2007 summer computer simulation conference*, San Diego, CA, 2007, pp.357–366.
24. Seo C and Zeigler BP. Automating the DEVS modeling and simulation interface to web services. In: *proceedings of the spring simulation multi-conference*, San Diego, CA, 2009.
25. Park S, Hunt CA and Zeigler BP. Cost-based partitioning for distributed and parallel simulation od decomposable multi-scale constructive models. *Simulation* 2006; 82: 809–826.
26. Bolduc, J. -S., and H. Vangheluwe. *A Modeling and Simulation Package for Classic Hierarchical DEVS, Technical Report.* McGill University, School of Computer Science, 2002.
27. Triccoli A and Wainer G. Implementing parallel Cell-DEVS. In: *proceedings of the 36th annual symposium on simulation*, Washington, DC, 2003, pp.273–277.
28. Sarjoughian HS and Zeigler BP. DEVS and HLA: complementary paradigms for M&S. *Trans Soc Comput Simulat* 2000; 17: 187–197.
29. Kim JH and Kim TG. DEVS Framework and toolkits for simulators interoperation using HLA/RTI. In: *proceedings of asia simulation conference*, Beijing, China, 2005, pp.16–21.
30. Christensen ER and Zeigler BP. *Hierarchical optimistic distributed simulation: combining DEVS and time warp.* Doctoral Dissertation, University of Arizona, University of Arizona, 1990.
31. Jafer S and Wainer G. Conservative vs optimistic parallel simulation of DEVS and Cell-DEVS: a comparative study. In: *proceedings of summer simulation conference*, Ottawa, ON, 2010, pp.342–349.
32. Wainer G, et al. DEVS standardization: ideas, trends and future. In: Wainer G and Mosterman PJ (eds) *Discrete-event modeling and simulation: theory and applications*. New York, New York: Taylor and Francis Group, 2010, pp.389–392.

33. Wainer G. *DEVS Standardization Study Group*. Interim *Final Report*. 2005. Seattle, WA: The SISO Standards Activity Committee (SAC).
34. Traoré MK. SimStudio: a next generation modeling and simulation framework. In: *proceedings of the 1st international conference on simulation tools and techniques for communications, networks and systems & workshops*, Marseille, France, 2008.

Author biographies

Adedoyin Adegoke is a PhD student at the African University of Science and Technology under the supervision of Prof. Traoré. She is currently working on the specification of DEVS implementation strategies. Her research interests are modeling methodologies and parallel/distributed simulation.

Hamidou Togo is a PhD student at the Blaise Pascal University (Clermont-Ferrand – France) and is a research assistant at the University of Bamako. His research interests focuses on DEVS formalism and simulation virtual machines.

Mamadou K Traoré received his MSc (1989) and PhD (1992) from Blaise Pascal University (Clermont-Ferrand – France) where he is a Professor of Computer Science. His current research is on formal specifications, symbolic manipulation and automated code synthesis of simulation models.