Proceedings of the 2005 Winter Simulation Conference
M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines, eds.

BIAS IN PARALLEL AND DISTRIBUTED SIMULATION SYSTEMS

Tobias Kiesling

Fakultät für Informatik
Universität der Bundeswehr München
85577 Neubiberg, GERMANY

Johannes Lüthi

Studiengang Wirtschaftsinformatik
FHS KufsteinTirol
6330 Kufstein, AUSTRIA

Rachid El Abdouni Khayari

Fakultät für Informatik
Universität der Bundeswehr München
85577 Neubiberg, GERMANY

ABSTRACT

Even after several decades of research, modeling is considered an art, with a high liability to produce incorrect abstractions of real world systems. Therefore, validation and verification of simulation models is considered an indispensable method to establish the credibility of developed models. In the process of parallelizing or distributing a given credible simulation model, a bias is introduced, possibly leading to serious errors in simulation results. Depending on the mechanisms used for parallelization or distribution, a separate validation of the parallel or distributed model is required. A necessary first step for such a validation is an understanding of the sources of bias that might occur through parallelization or distribution of a simulation model. The intention of this paper is to give an overview of the various types of bias and to give a formal definition of the bias and its quantification.

1 INTRODUCTION

When developing a parallel or distributed simulation model, several choices have to be made regarding the partitioning of the simulation tasks in multiple subtasks and the synchronisation between the parallel processes. Depending on the nature of the simulation, several different approaches exist. For example, in real-time training simulations, a federation of sequential simulations is created using Distributed Interactive Simulation (DIS) (IEEE Std 1278.1–1995). A constructive (as-fast-as-possible) discrete-event simulation is parallelized using conservative or optimistic synchronization methods (Fujimoto 2000). Simulations from both areas can be used together in a High-Level Architecture (HLA) federation (Dahmann, Fujimoto, and Weatherly 1997). In each of these cases, the newly developed distributed model is distinct from the model that was parallelized (the models that were composed to a distributed simulation, resp.).

Classic synchronisation methods for parallel simulation of discrete-event models were devised to ensure correctness of the parallel simulation, i.e. to produce results identical to the results of a corresponding sequential simulation. Formal proofs of this property have been provided for conservative (Misra 1986) and optimistic (Leivent and Watro 1993) synchronization. However, in many cases, the parallelization methods seriously restrict the degree of parallelism of simulation systems. Therefore, variants of these methods have been developed where the correctness property does not hold (Kiesling and Pohl 2004, Martini, Rümekasten, and Tölle 1997, Rao, Thondugulam, Radhakrishnan, and Wilsey 1998, Wang and Abrams 1992). In the latter case, credibility of the parallel simulation models has to be established by the use of validation and/or verification methods in addition to the validation of the sequential model.

The situation is similar with distributed simulation systems where no strict time-management is utilized, as is the case with DIS, or HLA federations where at least one federate is not time-constrained. The necessity of performing validation and verification activities for the distributed models has been recognized before (Brade 2003, Section 6.8), but no decisive steps in this direction have been taken.

Figure 1 shows an abstract view of the development process of a parallel or distributed simulation. Starting from a problem in the real world, a model of the real system is created. It is supposed here, that the initial model is of a sequential nature, i.e. no special precautions for the parallelization of the model have been taken. The abstraction that is involved in the modeling step always leads to a discrepancy between the behavior of the real system

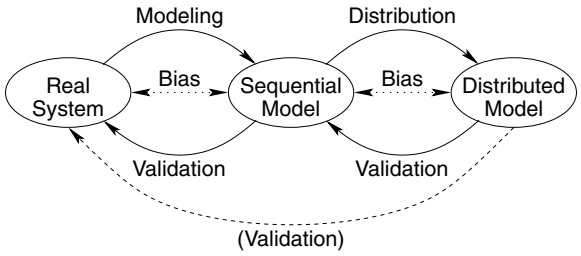

Figure 1: Distributed Simulation Development

and the behavior specified in the model. This discrepancy can be interpreted as a *bias* of the model in respect to the real system (the term bias is used in a non-statistical sense in this paper). The model is considered credible if the bias is negligible according to the problem definition. To ensure this credibility, validation of the model against the real system is performed according to pre-defined criteria.

To create a parallel or distributed simulation, the sequential model is enhanced with details about the distribution (or parallelization), resulting in a distributed (or parallelized) model. As discussed above, this introduces another level of bias, which is caused by the utilized distribution (or parallelization) method. It is dangerous to always consider the distributed model valid without having performed another validation step, either against the sequential model, or against the real system, or both.

An important aspect to be considered for every validation activity of the distributed model is the bias introduced in the distribution/parallelization step. Without a more detailed knowledge of the bias caused by the distribution, validation might be hard to perform. This paper gives an overview about the specific types of bias that might occur in a model (Section 2) and gives a formal definition of the bias and its quantification (Section 3).

2 CLASSIFICATION

Prior to formalization of different types of uncertainty, vagueness, and incorrectness in Section 3, we summarize with the term *bias* that there is a difference of an aspect of a simulation run from some reference. Different aspects of interest as well as different reference systems may be considered. For example, the following aspects in a (distributed) simulation may be of interest:

- The trajectory of the state vector in the state space,
- an aggregated performance measure (e.g. the average utilization of a service center in a queueing network model),
- events occurring in an event-driven simulation,
- the exact ordering of events in a discrete-event simulation,
- the state of the simulation system at an arbitrary point in wall clock time during the simulation run.

Typical reference systems are:

- The real/physical system (keyword *validation of simulation models*),
- the formal and/or conceptual model the executable model is based on (keyword *verification of simulation models*),
- another simulation run with the same simulator (keyword *repeatability of simulation experiments*),
- a sequential version of a parallel or distributed simulation system (keyword *synchronization and coordination in parallel and distributed simulation*),
- an idealized distributed simulation system, i.e. distributed simulation using a reliable communication system without latencies.

There are various sources for bias in the above sense. Below, we roughly classify reasons for uncertainties and incorrectness in three categories:

1. Bias that is caused independently from parallelization or distribution of the simulation model,
2. bias in parallel and distributed simulation that is caused by the technical framework applied in parallel and distributed simulation, and
3. bias that is deliberately accepted by the modeler (simulation developer, resp.) in a controlled fashion in order to increase the performance or fault tolerance of a parallel or distributed simulation run.

2.1 Bias not Depending on Parallelization and Distribution

Even in sequential simulation on a single processor, aspects of vagueness, inexactness, and uncertainty may occur. In order to obtain a clearer understanding for the effects of parallel and distributed simulation, we briefly discuss aspects that occur independently from parallel or distributed simulation.

2.1.1 Abstraction and Implementation

While constructing a simulation model, the required abstractions and simplifications always cause a divergence of the models behavior from the behavior of the real system. Controlling this type of inexactness in a systematic way is dealt with by model validation (Balci 1998, Brade 2003). The degree of inexactness tolerated here is implicitly or explicitly defined via the primary objective of the simulation model.

Incorrect behavior and associated inexactness of a simulation model as opposed to the corresponding formal or conceptual model may also be caused by incorrect trans-

lation and implementation. Such aspects should be dealt with by methods of model verification (Balci 1998).

2.1.2 Stochastic Modeling

Using random variables is a popular instrument to represent details of the real system that are not available to the modeler or that are too fine grained for the intended purpose of the simulation model. Consequently, in stochastic simulation uncertainty and vagueness occur at least twofold: Firstly, repeated simulation runs differ in their behavior due to the introduced randomness. Secondly, the behavior of the model clearly differs from the behavior of the real system, since in the real system many aspects modeled via random variables are in fact deterministic.

Since random behavior in simulation models is actually realized via pseudo random sequences (which causes an additional difference between the formal and the executable model), uncertainty in the first sense may be eliminated by controlling the seeds of the pseudo random number generators. This way, also (pseudo) stochastic simulation experiments can be made repeatable. Uncertainties in the second sense are an integral property of stochastic simulation. They can be dealt with by appropriate statistical methods (such as confidence intervals or the computation of higher moments of simulation results) if the intended purpose of the simulation model is system analysis. If the intended purpose is training and education, the trainee and trainer should be aware of random aspects of the model.

2.1.3 Discretization and Rounding Errors

Usually, systems of differential equations are used for the representation of continuous aspects of system behavior. Since such mathematical representations often do not allow for a closed form solution, numerical methods are applied that typically rely on the discretization of continuous model aspects. This usually implies a difference between numerically computed model solutions and the theoretical exact solutions of the formal model. However, many methods of computational mathematics provide an estimation or bound for the error introduced by discretization. Moreover, discretization errors may be controlled by using techniques of adaptive steplength control.

Also, rounding errors introduced by the application of numerical methods may be controlled by using appropriate mathematical tools (e.g., interval arithmetic (Moore 1988, Neumaier 1990). However, to the best of our knowledge, such methods are rarely used in simulation models.

2.1.4 Interactivity

The influence of human interaction on simulation behavior has a direct negative influence on the repeatability of simulation runs by its very nature. However, repeatability w.r.t. human interaction is not a realistic requirement in interactive simulation.

2.2 Uncertainties Caused by the Principles and Technical Frameworks of Distributed Simulation

Parallel and distributed simulation is based on the principle of simulating partitions of a simulation model on a number of processing nodes. Depending on the type of synchronization method, the most relevant aspects of the technical framework for parallel and distributed simulation w.r.t. bias and uncertainty are communication latencies, differing hardware clocks and the necessity to order simultaneous events.

2.2.1 Communication Latencies

Communication between processors or computing nodes is an integral part of parallel and distributed simulation. Depending on the time management of the distributed simulation model, communication latencies can be a source of additional uncertainties and inexactness.

In uncoordinated real-time simulation (e.g., DIS – Distributed Interactive Simulation) communication latencies cause a delayed arrival and thus a delayed consideration of messages. This yields inexact behavior in at least two respects: Firstly, such delays cause a difference of the simulation behavior from the formal model and a corresponding sequential implementation. This can lead to time anomalies and causality violations (consider for example the situation where an observer sees a target destroyed before he/she has seen another tank firing). Secondly, communication latencies contribute to differences between repeated simulation runs, because many communication systems involve non-deterministic behavior (e.g., Ethernet-based local area networks or the Internet). When using coordinated real-time simulation, uncertainties of the latter kind may be eliminated. However, latencies may still cause a violation of real-time requirements.

The well-known mechanisms for time coordinated as-fast-as-possible simulations, such as conservative (Chandy and Misra 1979) or optimistic simulation (Jefferson 1985), guarantee that parallel simulation produces the same event sequences as sequential simulation. Thus, uncertainties described in this section do not appear in time-coordinated simulation. Especially, repeatability is guaranteed despite the presence of indeterministic communication latencies.

2.2.2 Asynchronous Hardware Clocks

Even without the consideration of communication latencies, uncoordinated distributed real-time simulation assumes synchronized hardware clocks in order to obtain correct behavior. However, exact synchronization of hardware clocks

cannot be guaranteed. This may contribute to various uncertainty aspects: a divergence from the formal model and from a sequential implementation as well as non-repeatability of simulation runs. In analogy to communication delays, also unsynchronized hardware clocks do not contribute to uncertainties in the case of coordinated as-fast-as-possible simulations.

2.2.3 Simultaneous Events

In order to obtain repeatable simulation runs, special care has to be taken on simultaneous events (i.e., events with identical simulation time stamp) even in sequential simulation. Repeatable ordering of simultaneous events is an even more critical challenge in parallel and distributed simulation.

If no measures are taken in order to guarantee a repeatable ordering of simultaneous events, uncontrollable factors such as different event rates or different communication latencies in repeated simulation runs can cause a different ordering of simultaneous events.

Various approaches can be used in order to enable repeatability of distributed simulations also in the presence of simultaneous events. These techniques work by making identical time stamps unique. Among other approaches, this can be achieved by associating additional bits with each time stamp that allow the consideration of causal interdependencies. Similar results can be obtained by associating different priorities to events with equal time stamps. In (Fujimoto 2000), a combination of using an age field, priority, node-ID and a sequence number is suggested. An overview of different event ordering schemes can also be found in (Rönngren and Liljenstam 1999).

2.3 Exploiting Uncertainty and Tolerating Inexactness in Order to Achieve Higher Efficiency

In this section, we discuss existing approaches and techniques for parallel and distributed simulation that deliberately exploit uncertainties in the simulated real system or tolerate (slightly) incorrect behavior of the executable model in order to achieve higher efficiency of the simulation execution.

2.3.1 Temporal Uncertainty

In many models, sufficiently high lookahead values are hard to determine, preventing high efficiency in conservative simulation of time coordinated models. In previous work, future lists have been proposed to increase lookahead capabilities (Nicol 1988).

In (Martini, Rümekasten, and Tölle 1997), within the context of a hybrid optimistic/conservative synchronization scheme, the conservative blocking behavior is relaxed by introducing a tolerance ϵ. Consider a synchronization point at simulation time t_0. Then, the tolerant synchronization approach allows simulation until time $t_0 + \epsilon$. Events with a time stamp $t_0 + x < t_0 + \epsilon$ are scheduled at simulation time t_0. This introduces a controllable error ϵ in the exact time stamps of simulated events that allows to increase the efficiency of the simulation execution significantly. The authors argue that the introduced error corresponds to unknown factors in the real system. Furthermore, it is reported that the effect of the error in time stamps of some events on the error of the overall simulation result is usually smaller than the confidence interval of the result.

Fujimoto addresses a similar yet different solution to the low/zero-lookahead-problem: In (Fujimoto 1999), a partial order called *approximate time order* is introduced: in approximate time order, intervals are used to capture temporal uncertainty of the simulated real system. Fujimoto argues that temporal uncertainty can be found in basically every model because simulation is always only an approximation of the real world. Given two time intervals, an is-before relation holds if the intervals do not overlap. For overlapping intervals no ordering relationship exists between the two events. In the sequel, this relaxed order for overlapping time intervals is exploited in order to increase efficiency in conservative simulation. However, causal interdependence between events may be lost with that approach. Thus, another partial order is defined that considers causality as well: *approximate time causal order*. In experiments based on the PHOLD algorithm (Fujimoto 1990), Fujimoto showed that significant speedup could be achieved by using time intervals and approximate time or approximate time causal order with moderate effects on the accuracy of the simulation results.

A disadvantage of the approach proposed in (Fujimoto 1999) is the fact that in order to use the proposed algorithms for the Runtime Infrastructure (RTI) in the High Level Architecture (HLA), message delivery mechanisms and the HLA interface specification would have to be modified (Loper and Fujimoto 2000). To achieve compliance with the HLA specification, Loper and Fujimoto propose to use a combination of time intervals and pre-sampling of a precise time stamp within time intervals according to some specified probability distribution (Loper and Fujimoto 2000). The effect of this algorithm is an increase of lookahead values that can directly be used in HLA-based simulation.

2.3.2 Spatial Uncertainty

In analogy to temporal uncertainty, Quaglia argues that in many real systems also spatial uncertainty exists (Quaglia and Beraldi 2004). In parallel and distributed simulation based on the logical process (LP) paradigm, usually a uniquely specified LP is responsible for the simulation of a given event. Quaglia proposes to exploit spatial uncertainty in the model in order to relax this mapping by associating a

set of LPs to every event. The event can be passed to any of the LPs in this set. This freedom is subsequently exploited as follows: In optimistic simulation, an event that would cause a rollback at the receiver LP may be passed on to other LPs in the set associated to the event. This way, the number of rollbacks can be reduced and the efficiency of the optimistic simulation run is increased. Similarly, spatial uncertainty can be used to increase lookahead in conservative simulation. In (Quaglia and Beraldi 2004), this technique is applied to the simulation of mobile communication systems and a moderate bias of the simulation results is reported.

2.3.3 Approximate Time-Parallel Simulation

Kiesling et al. use a technique based on tolerating small errors in time-parallel simulation (Kiesling 2004, Kiesling and Pohl 2004, Kiesling and Lüthi 2005). In time-parallel simulation, the simulation time interval is partitioned into subintervals and every node simulates the whole model for one of these subintervals. The main problem in time-parallel simulation is the so-called state-match problem: Generally, the simulated state at the end time of one node does not match the initial state of the successor node. Various methods have been proposed to solve the state-match problem, e.g. the method of fix-up computations (Heidelberger and Stone 1990). However, the requirement of exact state matching often prohibits efficient use of time parallel simulation. In (Kiesling and Pohl 2004) it is proposed to relax exact state matching and tolerate some difference in neighboring states. For some applications, it is possible to simultaneously control the error in the simulation results (e.g. for cache simulations reported in (Kiesling 2004)). Another application area where this approach appears to be promising is traffic simulation, as reported in (Kiesling and Lüthi 2005).

2.3.4 Relaxed Ordering Mechanisms

Most synchronization schemes for time coordinated distributed simulation rely on using time stamps for events and apply a time stamp ordered event simulation and/or message delivery. However, strict time stamp ordering often decreases the efficiency of distributed simulation systems, sometimes to a level that real-time requirements are no longer met. On the other hand, pure timestamp ordering is sometimes overpessimistic, because not every message that is delivered out of order necessarily produces a causality violation. There exists a couple of approaches to overcome this problem by relaxation of the strict time stamp ordering.

The most radical concept is that of *unsynchronized parallel discrete-event simulation*, proposed by Rao et al. (Rao, Thondugulam, Radhakrishnan, and Wilsey 1998). In this work, the effect of dispensing with synchronization in time-warp-based simulation of queueing models is studied. For this special case, significant efficiency improvements were reported in terms of both processor time and memory usage, with surprisingly little effect in the overall simulation results. However, the general applicability of such a radical approach seems questionable.

A concept originally developed in the context of multimedia systems with real-time requirements is the notion of *delta causality* (Yavatkar 1992, Adelstein and Singhal 1995). In delta causality, an expiration time is associated with every time-stamped message. Messages that are received before the expiration time is elapsed are delivered and processed in time-stamp order. Messages that are delivered after expiration, may be processed out of order, eventually causing causality violations. A variant of this concept is also applied in DIS.

Time management in the high level architecture (HLA) is based on two options for message delivery: receive order and time-stamp order. In (Lee, Cai, and Zhou 2001), the notion of causal order in the context of HLA time management is introduced. Causal order focuses on the most important aspect of time-stamp order, namely preservation of causality. However, the costly ordering of all time-stamp-order messages is reduced such that only messages with causal dependencies are ordered. One major challenge in causal ordering is to provide less costly meta information about causality constraints than event time stamps. In (Zhou, Cai, Turner, and Lee 2002), the authors propose an enhanced version of causal ordering, namely *critical causal order*, being an additional relaxation of causal order. With the critical causal order mechanism, it is possible to preserve causality even with a relatively large number of federates, while still preserving important real-time properties of the simulation system.

2.3.5 Dead Reckoning

Communication overhead is a significant performance limitation in distributed real-time simulation. An important technique that is applied to reduce communication effort is *dead reckoning*. Dead reckoning is based on extrapolation models of objects simulated by other nodes. These so-called dead-reckoning models are used as local representations of external objects. Such extrapolations usually provide only approximations of the correct state of these objects. The simulation node that is responsible for the simulated object computes the associated dead-reckoning model as well. This way, updates need only be sent when a predefined accuracy threshold is exceeded. Among other uses, dead reckoning is an essential method in the DIS standard (IEEE Std 1278.1–1995).

3 FORMAL DEFINITION

In the previous section, some basic properties of the bias introduced in the modeling process are discussed in an informal way. To be able to devise a general approach to the examination of the bias in a given distributed simulation model, a formal definition of bias caused by the distribution is necessary. As introduced in Section 2, there are various different aspects of a simulation run, where a bias can be measured. A formal definition of the bias highly depends on the specific aspect to be measured. In this section, we will only cope with state deviations at a specific simulated time and show exemplarily, how a bias can be defined for this aspect.

The formalization should be applicable to constructive (as-fast-as-possible), as well as interactive simulation applications. At first sight, these two areas appear as quite distinct types of distributed simulations. Fortunately, there are commonalities between constructive and interactive simulation in the way the overall simulation state is distributed among processing nodes. If we neglect more uncommon simulation parallelization methods, like time-parallel simulation, the basic way of distributing a simulation task among multiple processing nodes is the decomposition of the overall system state into a number of sub states.

Before discussing details about state decomposition, some notions have to be introduced. The term *logical process* will be used to denote executing entities in the parallel or distributed simulation system. Logical processes perform an assigned simulation task in cooperation with other logical processes. Typically, one logical process is associated with every processing node. In general, however, multiple logical processes might execute on the same node. The term logical process is well established in the field of parallel discrete-event simulation (Fujimoto 2000). Furthermore, it will be used here to denote federates in a simulation federation, as well. The state of a simulation system is composed of a number of state variables, each having a scalar value. However, when looking at the semantics of state variables, often a grouping of the atomic variables is obvious. For example, in an interactive flight training simulator, the state of an airplane is typically represented by several state variables for its position, velocity, heading, among others. The overall state of the flight simulator consists of the composition of the state variables of all airplanes in the simulation. Here, the state variables can (and frequently will) be grouped by correspondence to a specific airplane. When speaking of such a group of state variables, the term *object* will be used in the rest of the paper.

Regarding the decomposition of the overall system state in parallel and distributed simulation, there are two extreme cases of this decomposition, shown in Figures 2 and 3.

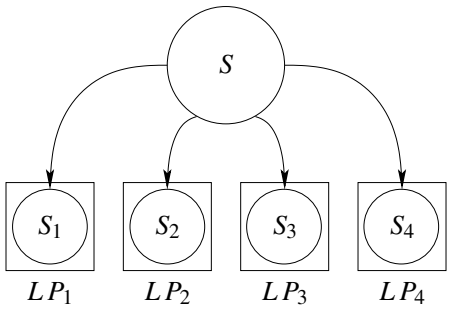

Figure 2: Replication of States

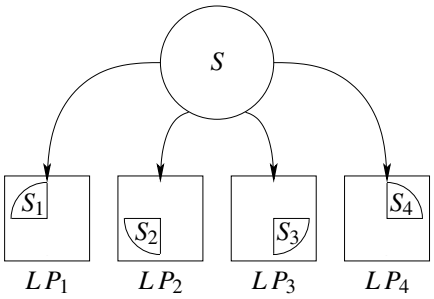

Figure 3: Partitioning of States

Figure 2 illustrates the case where the overall system state S is completely replicated on all logical processes. In that case, every logical process LP_i has a representation of the overall system state and is responsible for its consistency and timeliness. The representations of the system state of different logical processes can be identical at a given time, but in general, this will not be the case, due to various influencing factors (e.g. message latencies, update mechanisms). An example of this case is a DIS training simulation. In a DIS simulation, every simulator has knowledge about the state of all objects in the simulation. To reduce the number of messages transmitted in a DIS training simulation, the dead reckoning technique is used. This frequently results in a deviation of the states of two different logical processes, which can be interpreted as a bias in the distributed model.

The other extreme case is presented in Figure 3, where the overall system state is partitioned into multiple non-overlapping substates. Each of these substates is assigned to a separate logical process, which is responsible for performing the calculations associated with the corresponding objects. To support interrelationships between objects in different partitions, communication between logical processes is performed. This approach is often followed in classical parallel discrete-event simulation models. If synchronization between processes is relaxed, as discussed in Section 1, the composition of all the sub states at a given simulation time is not necessarily the same as in a corresponding sequential simulation, introducing a bias in the distributed model.

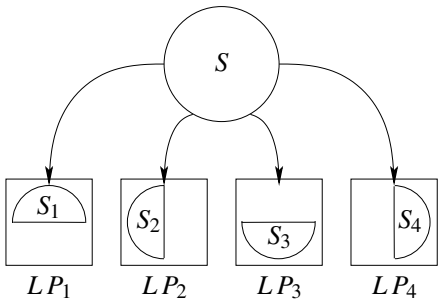

Figure 4: General State Decomposition

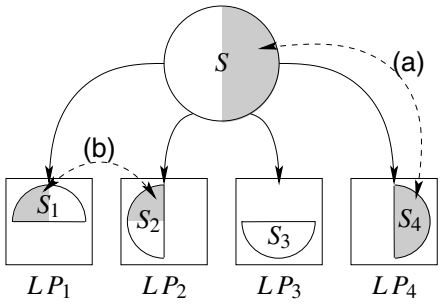

Figure 5: Comparisons of States

In most practical cases, however, these two extremes of pure replication and pure partitioning rarely can be encountered. Frequently, as illustrated by Figure 4, the overall system state is decomposed into substates, with a possible overlap of substates assigned to different logical processes. This is in fact a general view on distributed simulation, as it can represent both of the extreme cases shown in Figures 2 and 3. Many classical parallel discrete-event simulation models use this decomposition with overlapping substates, as well as federates in an HLA federation using declaration management to subscribe only to certain object classes.

When using the view on state decomposition presented in Figure 4, the measurement of a bias can be grouped in two different classes (cf. Figure 5):

(a) The bias can be measured between the sub state of a specific logical process and the *correct* state of the corresponding sequential or idealized simulation system (cf. Section 2), and

(b) the bias can be measured between common parts of sub states of different logical processes.

Now, the preliminaries for a formal definition of the bias have been introduced. The state of a sequential simulation at a time t is represented by a vector $s(t) = (s_1(t), \ldots, s_J(t))$, where $s_j(t)$ for $j = 1, \ldots, J$ is the state of the object j in the simulation. $s_j(t)$ might be a scalar value, but in general it is considered to be a vector. For example, in an interactive vehicle training simulation, $s_j(t) = (x_j(t), y_j(t), v_j(t))$, i.e. the state of a vehicle (object) consists of its position in x- and y-coordinates and its velocity.

Note, that the measures that are to be introduced here, are only reasonable with deterministic simulation executions. However, this does not restrict the types of models that can be supported. For every stochastic model, deterministic simulation executions can be ensured by a pre-sampling of random numbers and executing the simulation with these numbers. This can be implemented efficiently without the need of large storage space by the utilization of pseudo random number generators with pre-determined seeds. Using this technique, deterministic and repeatable simulation exeutions are possible.

Furthermore, in a distributed simulation, there are a number I of logical processes, each having a state $s^i(t) = (s_1^i(t), \ldots, s_J^i(t))$. In every logical process i, the state of each object is composed of real numbers, i.e. $s_j^i(t) \in \mathbb{R}^n$ for some integer n depending on the type of object. As the logical processes may have only a restricted knowledge of object states, an additional value of object states is introduced: It may be the case that $s_j^i(t) = *$. The asterisk symbol $*$ indicates, that logical process i has no knowledge about object j.

The **bias of an object** is defined as the distance between two different states of the object. A real-valued distance measure d_j for object j is utilized to determine the object bias. A uniform definition of d_j for all objects in the simulation is not possible, as it depends on the type of object to compare. For example, in a queuing network simulation, a single queue would be an object in the simulation. As the state of the queue can be represented by a scalar value, the number of jobs in the queue at a time, d_j can simply be defined as the distance between the number of jobs in two different states. In the vehicle simulation mentioned above, however, the definition of d_j is more complex, as a distance in the position of a vehicle has to be aggregated with the difference in the velocity. Based on the distance d_j, we define the bias of an object j at time t as

$$b_j^i(t) := \begin{cases} d_j(s_j(t), s_j^i(t)), & \text{if } s_j^i(t) \neq * \\ 0, & \text{otherwise} \end{cases}$$

and

$$b_j^{i,k}(t) := \begin{cases} d_j(s_j^i(t), s_j^k(t)), & \text{if } s_j^i(t) \neq * \wedge s_j^k(t) \neq * \\ 0, & \text{otherwise.} \end{cases}$$

The term $b_j^i(t)$ is used for the comparison between the state of an object j at a logical process i and the state of the same object in the sequential simulation (marked with (a) in Figure 5). $b_j^{i,k}(t)$ is used for the comparison of the states of an object j at two different logical processes i and k (marked with (b) in Figure 5). Note, that for multiple objects in the simulation belonging to a common class of objects (i.e. having the same attributes), the same distance measure d would be used, resulting in a number of different distance

measures that is equivalent to the number of different types of objects in the simulation.

Now, it is possible to define the **bias of a logical process** i at time t as an aggregation of the bias of every object. For example this could be a weighted sum:

$$\text{(a)} \quad B_i(t) = \sum_{j=1}^{J} w_j b_j^i(t) \quad \text{(b)} \quad B_{i,k}(t) = \sum_{j=1}^{J} w_j b_j^{i,k}(t)$$

for Cases (a) and (b) of Figure 5. In the latter case, this represents the bias of a pair i,k of logical processes at time t. In both cases some arbitrary weights $w_1, \ldots, w_J$ with $\sum_{j=1}^{J} w_j = 1$ are utilized. This is a measure of the deviation of the state of a logical process from the state of the sequential simulation, or another logical process, resp. The weights w_j are model dependent and have to be chosen carefully, so as to reflect the proper impact of the object bias of every object j.

To calculate the **bias of a simulation model** at time t of a simultion run, the logical-process biases $B_i(t)$, or $B_{i,k}(t)$, resp., have to be aggregated. This can be done in various ways, for example as a weighted average:

$$\text{(a)} \quad B^{(a)} = \sum_{i=1}^{I} c_i B_i(t) \quad \text{(b)} \quad B^{(b)} = \sum_{i=1}^{I} \sum_{k=i}^{I} c_{i,k} B_{i,k}(t)$$

with $\sum_{i=1}^{I} c_i = 1$ for Case (a), and $\sum_{i=1}^{I} \sum_{k=i}^{I} c_{i,k} = 1$ for Case (b). Again, the weights c_i ($c_{i,k}$, resp.) have to be chosen according to the simulation model, although it might be possible to give more general definitions.

The bias measures $B_i(t)$ and $B_{i,k}(t)$ can now be used to determine the accuracy of a distributed model, either after a simulation run in comparison to a sequential model for a validation of the distributed model, or during the simulation execution for a determination of the result accuracy at a given simulation time. In this respect, an aggregation of the model bias over the simulated time might be of interest, as well. One possibility for the definition of such a measure is to integrate the simulation model bias over the simulated time. However, the details of the utilization of the defined bias measures is out of the scope of this paper.

4 CONCLUSIONS

In the process of building a simulation model, a bias is introduced, which is a discrepancy between the behavior of the real system and the behavior of the model. Likewise, a bias is also introduced with the parallelization or distribution of a previously developed sequential model. The former bias is caused by the abstraction that is an important feature of modeling. This is an issue that has been recognized long ago, and validation and verification techniques have been developed to ensure the credibility of developed models. However, the consequences of a bias introduced with parallelization or distribution of a model have been neglected in many cases. This paper tries to stimulate fundamental research of these consequences.

The bias introduced in the process of building a model (sequential or parallel/distributed) can be classified by various criteria: Firstly, there are several aspects of a simulation which have to be considered, e.g. the development of the simulation state over time (wall clock or simulation time), the order of events in a discrete-event simulation, or a statistical measure calculated by the simulation. Secondly, it has to be identified, what has to be compared when measuring a bias, e.g. the overall state of the simulation system against the state of a corresponding sequential simulation or the overlapping parts of the states of two different logical processes. Thirdly, the bias can be classified by its causes, e.g. bias caused by technical constraints of a distributed simulation or introduction of a bias to increase the performance of a simulation.

Based on these observations, a formal definition of the measurement of a bias introduced by the parallelization or distribution of a sequential model has been given. The definition has been restricted to the aspect of comparing states at specific simulation times, as this is deemed the most important aspect. Two fundamentally different ways of measuring bias have been introduced: as a distance between the sub state managed by a logical process to the corresponding sub state in a sequential or idealized simulation execution, or as a distance between the common parts of the sub states of two different logical processes. It is not possible to give a general definition of the distance measure to utilize, as it highly depends on the specific model.

It is an important first step for the validation of parallel or distributed simulation models to have an intricate knowledge of the bias introduced with the parallelization or distribution. Furthermore, to apply this knowledge, a measurement of the bias must be possible. These two aspects have been addressed in this paper. The utilization of bias measurement for the accreditation of a parallel or distributed model is still an open issue. Further research in this direction is necessary.

REFERENCES

Adelstein, F., and M. Singhal. 1995. Real-time causal message ordering in multi-media systems. In *Proceedings of the 15th International Conference on Distributed Computing Systems*, 36–43.

Balci, O. 1998. Verification, validation and testing. In *Handbook of Simulation*, ed. J. Banks, 335–393. John Wiley & Sons.

Brade, D. 2003. *A generalized process for the verification and validation of models and simulation results*. Ph. D. thesis, Universität der Bundeswehr München, Fakultät für Informatik.

Chandy, K. M., and J. Misra. 1979. Distributed simulation: A case study in design and verification of distributed programs. *IEEE Transactions on Software Engineering* SE-5 (5): 440–452.

Dahmann, J. S., R. M. Fujimoto, and R. M. Weatherly. 1997. The department of defense high level architecture. In *Proceedings of 29th Winter Simulation Conference*.

Fujimoto, R. M. 1990. Performance of time warp under synthetic workloads. In *Proceedings of the SCS Multiconference on Distributed Simulation*, 23–28.

Fujimoto, R. M. 1999. Exploiting temporal uncertainty in parallel and distributed simulations. In *Proceedings of the 13th Workshop on Parallel and Distributed Simulation*, 46–53.

Fujimoto, R. M. 2000. *Parallel and distributed simulation systems*. New York: John Wiley & Sons.

Heidelberger, P., and H. S. Stone. 1990. Parallel trace-driven cache simulation by time partitioning. In *Proceedings of the 1990 Winter Simulation Conference*, 734–737.

IEEE Std 1278.1–1995. IEEE standard for distributed interactive simulation — application protocols.

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

Kiesling, T. 2004. Approximate time-parallel cache simulation. In *Proceedings of the 2004 Winter Simulation Conference*, 345–354.

Kiesling, T., and J. Lüthi. 2005. Towards approximate time-parallel road traffic simulation. In *Proceedings of the 19th Workshop on Principles of Advanced and Distributed Simulation*.

Kiesling, T., and S. Pohl. 2004. Time-parallel simulation with approximative state matching. In *Proceedings of the 18th Workshop on Parallel and Distributed Simulation*, 195–202.

Lee, B.-S., W. Cai, and J. Zhou. 2001. A causality based time management mechanism for federated simulation. In *Proceedings of the 15th Workshop on Parallel and Distributed Simulation*, 83–90.

Leivent, J. I., and R. J. Watro. 1993. Mathematical foundations for time warp systems. *ACM Transactions on Programming Languages and Systems* 15 (5): 771–794.

Loper, M. L., and R. M. Fujimoto. 2000. Pre-sampling as an approach for exploiting temporal uncertainty. In *Proceedings of the 14th Workshop on Parallel and Distributed Simulation*, 157–164. Los Alamitos.

Martini, P., M. Rümekasten, and J. Tölle. 1997. Tolerant synchronization for distributed simulations of interconnected computer networks. In *Proceedings of the 11th Workshop on Parallel and Distributed Simulation*, 138–141.

Misra, J. 1986. Distributed discrete-event simulation. *Computing Surveys* 18 (1): 39–65.

Moore, R. E. (Ed.) 1988. *Reliability in computing: The role of interval methods in scientific computing*, Volume 19 of *Perspectives in Computing*. San Diego, CA: Academic Press, Inc.

Neumaier, A. 1990. *Interval methods for systems of equations*, Volume 37 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press.

Nicol, D. 1988. Parallel discrete-event simulation of FCFS stochastic queueing networks. *SIGPLAN Notices* 23 (9): 124–137.

Quaglia, F., and R. Beraldi. 2004. Space uncertain simulation events: some concepts and an application to optimistic simulation. In *Proceedings of the 18th Workshop on Parallel and Distributed Simulation*, 181–188.

Rao, D. M., N. V. Thondugulam, R. Radhakrishnan, and P. A. Wilsey. 1998. Unsynchronized parallel discrete event simulation. In *Proceedings of the 1998 Winter Simulation Conference*, 1563–1570.

Rönngren, R., and M. Liljenstam. 1999. On event ordering in parallel discrete event simulation. In *Proceedings of the 13th Workshop on Parallel and Distributed Simulation*, 38–45. Los Alamitos.

Wang, J. J., and M. Abrams. 1992. Approximate time-parallel simulation of queueing systems with losses. In *Proceedings of the 1992 Winter Simulation Conference*, 700–708.

Yavatkar, R. 1992. MCP: A protocol for coordination and temporal synchronization in multimedia collaborative applications. In *Proceedings of the 12th International Conference on Distributed Computing Systems*, 606–613.

Zhou, S., W. Cai, S. J. Turner, and F. B. S. Lee. 2002. Critical causality in distributed virtual environments. In *Proceedings of the 16th Workshop on Parallel and Distributed Simulation*, 53–59.

ACKNOWLEDGMENTS

Funding for this research was provided by the Institut für Technik Intelligenter Systeme (ITIS) and Tiroler Wissenschaftsfonds, contract UNI-404/66.

AUTHOR BIOGRAPHIES

TOBIAS KIESLING is a Research Assistant at the University of the Federal Armed Forces in Munich, Germany. He received his B.S. in Statistics in 1997 and his M.S. in Computer Science in 2002, both from the Ludwig-Maximilians-University in Munich, Germany. He is currently working on his Ph.D. degree in the field of

parallel and distributed simulation. His research interests include time-parallel simulation and the utilization of approximation techniques in parallel and distributed simulation. His e-mail address is <kiesling@informatik.unibw-muenchen.de>, his personal homepage can be found at <http://fakinf.informatik.unibw-muenchen.de/~tkiesling/>.

JOHANNES LÜTHI is a lecturer at the University of Applied Sciences in Kufstein, Austria. He received his M.Sc. and Ph.D. degrees in applied mathematics from the University of Vienna, Austria, in 1995 and 1997, respectively. From 1998 until 2002, he was an Assistant Professor at the Faculty of Computer Science of the University of the Federal Armed Forces in Munich, Germany, where he is still also lecturing. His primary research interests include performance and dependability models of computer and communication systems and distributed discrete event simulation. He is a member of the IEEE Computer Society, the ACM, the Austrian Computer Society (OCG), and the German Gesellschaft für Informatik. His e-mail address is <Johannes.Luethi@fh-kufstein.ac.at>, his personal homepage can be found at <http://www.fh-kufstein.ac.at/wi/jluethi/>.

RACHID EL ABDOUNI KHAYARI is working as Assistant Professor at the University of the Federal Armed Forces in Munich, Germany. From 1998 to 2002 he worked as a research assistant at the Department of Computer Science, Institute for Performance Evaluation and Distributed Systems at the Technical University Aachen (RWTH) where he received his Ph.D. Now Dr. El Abdouni Khayari is working on the design and performability evaluation of distributed computer and communications systems, focusing on the characterization on network traffic, caching and scheduling in world-wide web based systems. His e-mail address is <rabdouni@informatik.unibw-muenchen.de>, his personal homepage can be found at <http://fakinf.informatik.unibw-muenchen.de/~rabdouni/>.