Complete Event Ordering for Time-Warp Simulation of Stochastic Discrete Event Systems

Armin Zimmermann Michael Knoke Günter Hommel
Technische Universität Berlin
Institute of Computer Engineering and Microelectronics
Einsteinufer 17, D-10587 Berlin, Germany
{azi, mknoke, hommel}@cs.tu-berlin.de
Phone: +49 (30) 314 73 112, Fax: +49 (30) 314 21 116

Keywords: Distributed simulation, logical time, event ordering, immediate events

Abstract

Distributed simulation of stochastic discrete event systems is a well-known technique to speed up computationally expensive simulation runs. Synchronization between logical processes requires a *logical time*. Such a time scheme has been introduced recently to allow models to be decomposed into almost arbitrary logical processes, even in the presence of zero delays and global priorities. This paper discusses the background of this *compound simulation time*, and proves some important properties.

1 INTRODUCTION

A popular approach to speed up a simulation experiment is the use of multiple processing nodes. One possible technique of distributed simulation divides the model into parts, which are simulated by communicating *logical processes* LP. Just like in a standard sequential simulation, discrete event occurrences are observed in each logical process over a virtual simulation time scale. An event list is managed with events scheduled in the future, ordered by their occurrence time.

A distributed simulation has no centralized control to synchronize the different logical processes. Synchronization is realized by exchanging messages between the nodes; a fast communication subsystem is thus important for efficient execution. The underlying idea is that enabling of actions and event execution is often performed locally in stochastic discrete-event system (SDES) models. Many events occur thus at different points in time, but do not affect each other.

Distribution at the event level divides a global simulation task such that each logical process simulates a part of the global model. The model (state variables and actions) is partitioned into a set of regions that are associated to each process. *Internal events* do not affect state variables of other processes, while *external events* may do so. Non-local dependencies and results of events need to be propagated to the corresponding processes. Messages for state variable changes (remote events) and other notifications for management issues are exchanged via a communication system in a distributed simulation. The main problem to be solved is to guarantee causal correctness of a distributed simulation run.

A distributed simulation is obviously said to be correct w.r.t. the local event processing if the partial event ordering created by it is consistent with the total event order of a sequential simulation. This leads to the question which notion of time is necessary to achieve such an ordering. The *logical clock problem* [1] aims at generating clock values in a distributed system in a way that all events are ordered in a *logical time*. It was shown in [2], that this is the inverse of the problem in a distributed simulation run. Causality errors are impossible if all LP execute the events ordered by their time stamps. This is called the *local causality constraint* [3], and has been shown in [4].

If a model has only a few events that require messages to be exchanged between logical processes, it is safe in most cases to proceed inside such a process. The idea of optimistic logical process simulation or Time Warp [5, 2] is to temporarily accept the possibility of local causality violations. A violation occurs if a logical process LP_i receives a message from another one, notifying it about a past remote event affecting the local state of the process. Such a message is called a *straggler message*, and the causality violation is overcome by a *rollback* of the logical process to the time before the time stamp of the remote event, i.e. a state which is consistent with the received message.

A distributed simulation of SDES models thus requires time stamps for events that allow their unique and correct ordering. With the existing approaches it is however impossible to order events that are due to immediate action executions (with zero delay), or have priorities. Standard distributed simulations therefore require a model to be decomposed into regions of logical processes in a way that there are no zero-delay events to be sent, i.e. only at timed actions. It is then (practically) impossible that two events are scheduled for the same point in time.

Global enabling functions (guards) and condition functions (capacities) are impossible without global state access, which in turn requires a complete ordering between all event times in a distribute simulation as well. Standard simulation times lack this feature and can thus not be used.

We proposed a new logical time scheme for stochastic colored Petri net models recently [6, 7]. This *compound simulation time* is applicable to SDES models including immediate actions, priorities, and global functions. It allows partitioning with less structural restrictions than e.g. the approaches covered in [8]. The contribution of this paper is a discussion of the proposed logical time, expecially proving its soundness. Petri nets are used for examples; the logical time is, however, usable for distributed SDES simulation in general.

The remainder of the paper is organized as follows. Fine-grained model partitioning for dynamic load balancing [6, 7] is briefly introduced in the following section. Section 3 gives some background on logical time schemes. Problems arising with zero delays and priorities as well as a possible solution are shown in Section 4. The subsequent section defines the proposed *compound simulation time*, while Section 6 discusses and proves some of its properties. Global states and the use of the logical time scheme for this issue are covered in Section 7.

2 FINE-GRAINED PARTITIONING FOR DYNAMIC LOAD BALANCING

A fine-grained partitioning of an SDES model has the advantage of almost arbitrary associations of model parts to computing nodes. It is thus a prerequisite for dynamic load balancing. Different to standard time-warp simulations we propose to run one logical process per host, which manages several *atomic units* that are running quasi-parallel in that machine. An atomic unit is responsible for the optimistic simulation of a smallest possible model part. Each atomic unit has its own local simulation time, event and state list. This allows to migrate it during runtime without affecting other atomic units. An atomic unit can restore its local state accurately for a given simulation time and send rollback messages to other atomic units that might be affected. Rollbacks are thus more precise and unnecessary ones are avoided or canceled whenever possible. The way of scheduling the operations of atomic units inside a logical process avoids causality violations between them, reducing the number of rollbacks further.

We denote by au one atomic unit, and by AU the set of all of them.

$$AU = \{au_1, au_2, \ldots, au_{|AU|}\}$$

The mapping of atomic units to logical processes has been covered in [6, 7] already, and is not required for the presentation in this paper. Informally, an action together with its input state variables (as well as other actions that share them) forms one au. In the case of Petri nets, each *extended conflict set* of transitions [9] together with their input places form an atomic unit. The discussion in this paper can be easily transferred to standard time-warp simulations by understanding an atomic unit as one logical process.

We require the simulated model to be *confusion-free*, i.e. if the model evolution depends on the ordering of events, this ordering must be specified by the model. The compound simulation time presented in this paper actually allows to detect confusions on-the-fly; this issue is however not detailed here. If the SDES model to be simulated is confusion-free as required, conflicts can always be solved locally inside an extended conflict set, and thus inside one atomic unit. Events of two or more actions belonging to different atomic units and having the same priority may then be executed in any order, without changing the behavior of the stochastic process (compare e.g. [9]). The order of execution of these events can then be fixed arbitrarily. Different priority values must however be valid across the borders of atomic units — the corresponding events need to be ordered by the priority of the underlying action. We associate priorities to SDES model actions such that actions in different atomic units never have equal values, without disturbing the global priority ordering. This is possible without changing the behavior of a model if there are no confusions.

Another restriction is the absence of *immediate* or *vanishing loops*. We slightly restrict immediate paths requiring that an extended conflict set containing only immediate actions must not be visited more than once in one immediate path. This restriction is however of little practical significance.

3 LOGICAL TIME

Causal correctness of a distributed simulation algorithm for SDES models is guaranteed if the events are processed in the same sequence that a sequential method would follow. A sequential simulation processes events in the order given by the simulation time T, remaining delays, and by taking priorities of events into account which are scheduled for the same time. Following [3, 4], every atomic unit must execute events in a non-decreasing time stamp order. However, time stamps must allow a complete ordering.

A global simulation clock alone is not sufficient, because there are actions with zero delay to be executed at the same simulation time, and there is a non-zero probability that several timed actions are scheduled at the same time as well. The priority $\Pi(a)$ of an action a specifies the order of execution. Events to be executed at the same simulation time must be uniquely ordered at every place (atomic unit) where their ordering matters. Priorities and causal relations between these events must be taken into account for such a decision.

Lamport's algorithm [1] allows a time ordering among events [10]. A single number is associated to every event as its logical time, and increases with subsequent events. However, a mapping from Lamport time to real (simulation) time is not possible. This is no problem if the correct ordering is sufficient; in a simulation we however do need the actual simulation time T e.g. to compute performance measures. Lamport time is furthermore not sufficient to detect causal relationships between events, which is a prerequisite for models with action priorities. It is impossible to sort concurrent and independently executed events whose occurrence is based on a different priority. Lamport time would impose an artificial ordering and neglect the priorities. Moreover, it is forbidden for neighboring regions of a distributed model to exchange events that have a zero delay. In a colored Petri net, this would prevent models to be decomposed at immediate transitions, and thus restrict the formation of atomic units significantly.

A logical time that characterizes causality and thus overcomes some of the mentioned problems of Lamport time is *vector time* (or *vector clocks*) proposed by Mattern [11], Fidge [12] and others independently in different contexts. In our proposed setting, a vector time V value is a vector of natural numbers, which contains one entry for every atomic unit au.

$$V : AU \to \mathbb{N} \quad \text{or} \quad V \in \mathbb{N}^{|AU|}$$

Whenever an atomic unit executes an event or rollback, it increases the vector time entry of itself by one when the next logical simulation time is obtained from the current simulation time. The local entry of the V entry thus always increases, even when a rollback is processed. The element-wise maximum is taken for every non-local entry of V to update the local time, whenever a remote event is processed.

Vector time is a notion of causality, and can thus be used to differ between events that depend on each other. If event e_2 is causally dependent on e_1, it must naturally be

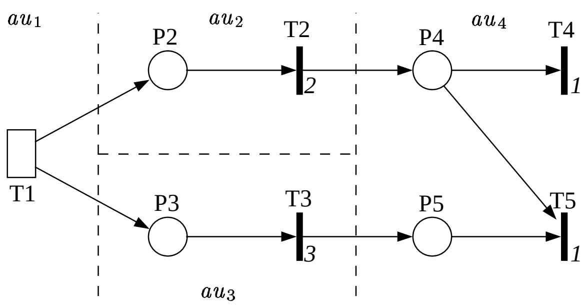

Figure 1. Simple Petri net example with immediate transition priorities

scheduled after it. In that case we write $V_1 < V_2$.

$$\begin{aligned} V_1 < V_2 &\iff \forall au \in AU : V_1(au) \le V_2(au) \\ V_1 = V_2 &\iff \forall au \in AU : V_1(au) = V_2(au) \end{aligned}$$

The case of all elements of two vector times being equal occurs only if two events are compared that result from actually conflicting actions in one atomic unit. Their execution sequence is then decided based on a probabilistic choice. This can however only happen inside one atomic unit and for events that are in the future of the local simulation time. It will never happen in a distributed simulation that a remote event has the same vector time as any other locally known one, because the same event is only sent once to another atomic unit. In algorithms where this cannot be guaranteed, all equally timed events must be executed together in one step.

Two events e_1 and e_2 are said to be concurrent with respect to their vector times, if there is no causal dependency found. This case is denoted by $V_1 \parallel V_2$.

$$V_1 \parallel V_2 \iff (V_1 \not< V_2) \wedge (V_2 \not< V_1)$$

Vector time is however still not sufficient for models with priorities and immediate delays. Two or more events can be scheduled for execution at the same (simulation) time in a SDES model with priorities, but the one with a higher priority must always be executed first. It may disable events with lower priorities by doing so.

A small Petri net example is shown in Figure 1. Priorities of transitions are annotated in italics; timed transitions (empty boxes) always have a priority of zero. The correct sequence of events would be the firing of transitions `T1`, `T3`, `T2`, and then `T4` or `T5`, depending on the probabilistic solution of the conflict between them.

In a distributed simulation of the model, transitions and places are associated to atomic units $au_1 \ldots au_4$ as shown. It might happen during distributed simulation that `T2` fires after `T1`, and the associated event is received and processed in au_4. `T4` then fires locally, which is not correct. This is detected later on, when T3 has fired and the corresponding event is received in au_4. The events of firing transitions T2 and T3 must be ordered correctly in au_4. Otherwise it cannot be detected that T3 had to fire before T2, and that the previous local firing of T4 must be rolled back.

An ordering of events T2 and T3 in au_4 is however impossible based on the simulation times $T_{\mathtt{T2}}$ and $T_{\mathtt{T3}}$ (which are equal) or the vector times of the events, which are concurrent.

The simulation clock, (measured in model time units), has to be incorporated in a time scheme for SDES to make it applicable for performance evaluation. Vector time is added to the simulation time to cover causal dependencies between events at the same time, and extended by a priority vector as described below.

4 IMMEDIATE EXECUTION PATHS AND THE PRIORITY VECTOR

Priorities (especially of actions with a zero delay) play a significant role in many subclasses of SDES. Event serialization may only take place when different events are sorted w.r.t. their logical times in atomic units that process them. Thus the decision about which event has (or had) to be executed first must be taken in a distributed way, i.e. in each atomic unit that receives and sorts events by their time.

We introduce a *priority vector* as a supplement to the logical vector time in order to sort events correctly. A priority vector P maps each atomic unit au to a natural number.

$$P : AU \to \mathbb{N} \quad \text{or} \quad P = \mathbb{N}^{|AU|}$$

This number stores the minimum priority of any event belonging to au, which has been executed in the current path of events that immediately followed each other. It should be noted that only paths of event executions between two tangible states are significant in this case, because otherwise the simple simulation time T is sufficient for the ordering. Such a path of immediate state transitions starts with the execution of an action with non-zero delay (like transition T1 in Figure 1), and ends again in a state in which some simulation time passes.

All event executions of one path must be considered in the priority vector, because every single one can influence the correct sequence. If two events are compared, the one with the smaller minimum priority entry must be ordered last. This is because the event with the lowest priority delays the propagation of an event until no other action with a higher priority on other paths can be executed. Entries are initialized with infinity as a neutral value of the minimum, and are again set to infinity when some simulation time T passes (i.e. in a *tangible state*).

When an immediate action becomes executable in a local state, it is scheduled with a logical time in which vector time and priority vector are copied from the current local logical time. The priority vector entry corresponding to the local atomic unit is set to the event priority of the scheduled action. This scheduled time is used to sort a new event into the local event list.

When an event is executed in an atomic unit, the local simulation time is advanced to a new value. The local priority vector of the atomic unit is then updated such that the entry related to the atomic unit of the event (it could be a remote or local action) becomes the minimum of the previous value and the entry in the event's priority. The remaining entries are not changed. All entries are set to the default value infinity if some time passed between the previous simulation time and the event execution. The only exception is the entry corresponding to the executed timed event, which is set to its priority.

The priority vectors are used to compare event times. However, it is obviously not adequate to compare the minimal priority vector entry. Priorities of actions that were visited in both paths have to be ignored, based on the vector time of the events. Equal entries in the two vector times denote identical dependencies.

We thus define a *minimal path priority* of a priority vector P_a to which a vector time V_a belongs, with respect to another vector time V_b as

$$P_{\min}^{a,b} = \begin{cases} \min\limits_{\substack{\forall au \in AU: \\ V_a(au) \neq V_b(au)}} P(au) & \text{if } V_a \neq V_b \\ \infty & \text{otherwise} \end{cases} \quad (1)$$

Note that we define this path priority to be infinity for completeness in the case of identical vector times; this case is however only of theoretical interest. For the example shown in Figure 1, it is then possible to order incoming events at au_4 correctly: $P_{\min}^{T2,T3} = 2 < P_{\min}^{T3,T2} = 3$, and thus the event corresponding to the firing of transition T3 is executed first.

One special case remains. It is possible that the minimal path priorities of two events a and b are equal with the formula introduced above, i.e. $P_{\min}^{a,b} = P_{\min}^{b,a}$. This may happen if the paths share a common atomic unit where conflicting immediate actions were executed, which by chance have the smallest priority on the path. The reason is that the events have the same atomic unit as their common predecessor, in which two conflicting transitions with the smallest priority initiated them. We only need to use the vector time entries that correspond to the atomic unit in which the minimal path priorities were set to detect the causal ordering (see Equation 2 below).

Example models covering the elements of the compound simulation time have been given in previous papers [6, 7], and are omitted due to space limitations.

5 COMPOUND SIMULATION TIME

We define a *compound simulation time ct* containing the actual simulation time T, vector time V, and priority vector P of an event or any other specific point in time. The set of all possible compound simulation times is denoted by CT.

$$CT = \Big\{ (T, V, P) \mid T \in \mathbb{R}^{0+}, V \in \mathbb{N}^{|AU|}, P \in \mathbb{N}^{|AU|} \Big\}$$

Events e mark state changes that are either executed or are scheduled in the future of an atomic unit. Furthermore they are exchanged in messages. An event $e = (ct, a)$ comprises the executed action a and the corresponding compound simulation time ct. The set of all possible events is denoted by E.

The compound simulation time is intended for an ordering between different events. For any two events e_1 and e_2 it must be clear which one has to be scheduled first. Another application is the comparison of a remote event time with the local simulation time, which is important to decide whether an event is scheduled for the future or past of the local time of an atomic unit.

The comparison is performed using the compound simulation times ct_1 and ct_2. While the comparison of the actual simulation times is obvious, things are more complicated when vector times and priority vectors are taken into account. This has been informally explained already above, and is now defined formally. Elements of compound simulation times are denoted by assuming that $ct_i = (T_i, V_i, P_i)$.

$$\begin{aligned}
&\forall ct_1, ct_2 \in CT : ct_1 < ct_2 \iff (T_1 < T_2) \vee \\
&(T_1 = T_2) \wedge \Big((V_1 < V_2) \vee \\
&\quad (V_1 \parallel V_2) \wedge \big[(P^{1,2}_{\min} > P^{2,1}_{\min}) \vee \\
&\qquad \exists au \in AU : \big(P^{1,2}_{\min} = P^{2,1}_{\min} = P_1(au)\big) \wedge \\
&\qquad\quad \big(V_1(au) < V_2(au)\big) \big] \Big)
\end{aligned} \tag{2}$$

There are four cases: If simulation time or vector time allow a decision about which time is smaller, it is taken accordingly (cases one and two). If $V_1 \parallel V_2$, the minimal path priorities are compared. The time with the greater value is then smaller (case three). If they are equal as well, the decision is based on the vector time entry of the significant atomic unit (i.e. the one in which the minimal path priority occurred). The fourth case covers equal minimal path priorities.

Formally is is also possible that two compound simulation times are equal, meaning that all elements are completely identical. This is however only possible in the case of two events that belong to the same atomic unit, have zero delay and equal priority, and are being scheduled for execution in a given local state. This means that they are in conflict, and the order of execution will be decided locally in the atomic unit by a probabilistic choice. A comparison between such events is thus never necessary to decide the correct order. If it is technically possible that the results of one event are propagated to other logical units seperately in an actual implementation, they can be identified by equal compound simulation times and must be executed together.

Another issue is the selection of the compound simulation time ct' for an event that is newly scheduled and inserted into a local event list. The event is scheduled for the actual simulation time plus a randomly drawn delay, corresponding to the probability distribution of the action's delay. If it is different from zero, all priority vector entries are reset to infinity except for the local value, which is set to the priority of the scheduled action. It would be possible to reset the entries of the V vector every time a non-zero delay passes, just as it is done for the priority vector P. This is however avoided to keep the full causal information for all messages, including rollbacks, for an improved message cancellation mechanism. The other entries are copied. The vector time is not changed at this point; it is updated upon actual execution of the event.

The scheduled compound simulation time of future local events always has to correspond to the current local compound simulation time. This means that if a future event in the local event list stays executable after an event execution, its compound simulation time has to be updated, while the scheduled simulation time T is kept. This can be efficiently implemented without recalculation by transparently mapping the current local vector time and priority vector to local future events.

When an event e, which is caused in an atomic unit au_e, is executed in the atomic unit au, the local compound simulation time ct^{au} is updated as follows. The new simulation time is set to the time of the event first. The local vector time entry is incremented by one, and the maximum of current and event vector time entries is taken for all "remote" entries. $P(au_e)$ is set to the priority of the executed local action, while the other entries of P assume the minimum of the previous value and the entry of e. A more detailed description is contained in in [6, 7].

6 DISCUSSION OF COMPOUND SIMULATION TIME

In our attempt to simulate a timed synchronous system on distributed computing hardware, a compound simulation time has been introduced for event ordering. The goal of this section is to show that (1) our time scheme associates a reasonable compound simulation time to each event, (2) the introduced time scheme complies to the nature of time, and (3) that it is possible to derive global state information effectively using it.

6.1 Relation of Events and Clock Values

Do the algorithms associate a "meaningful" compound simulation time $ct(e_i)$ to every event e_i? This issue is considered as *clock condition* in literature (see e.g. [1, 11, 13]). If an event e_1 *may affect* e_2 (which is denoted by $e_1 \to e_2$), it is mapped to an "earlier" logical time similar to the "*happens before*" relation in [1]. This ensures that the future can not influence the past in the logical time scheme, as it is natural in our understanding of the passage of time.

$$\forall e_1, e_2 \in E : \big(e_1 \to e_2\big) \longrightarrow \big(ct(e_1) < ct(e_2)\big) \qquad (3)$$

We consider events e_i of the distributed simulation here; they are related to the simulated SDES model thus, and not to the distributed way of computation as usually understood in the literature about distributed systems. An alternative interpretation of $e_1 \to e_2$ is thus that e_1 *is executed before* e_2 in a sequential simulation.

We examine the conditions under which e_1 affects (or is executed before) e_2 in a sequential simulation. The following cases are distinguished for a complete proof of the proposition in Equation 3. Here and in all following proofs we denote the compound simulation time associated to an event e_i by ct_i, and its elements simply as $ct_i = (T_i, V_i, P_i)$ for notational convenience.

If an event e_2 is executed later in the actual simulation (model) time T, it may be affected by an earlier event e_1.

$$\forall e_1, e_2 \in E : \big(T_1 < T_2\big) \longrightarrow \big(e_1 \to e_2\big)$$

Then obviously also $ct(e_1) < ct(e_2)$ because of the definition of '$<$' for compound simulation times in Equation 2.

If an event has to be executed at the same simulation time, but is causally dependent on another event, it should be executed later. Causal dependency is fully captured by vector time [11]. Thus

$$\forall e_1, e_2 \in E : \big((T_1 = T_2) \wedge (V_1 < V_2)\big) \longrightarrow \big(e_1 \to e_2\big)$$

which obviously leads to $ct(e_1) < ct(e_2)$ due to the second case in Equation 2.

There are cases in which two events are executed at the same simulation time, but are not directly causally dependent. Both belong to individual paths of immediate executions then, which started at the same tangible state. The two paths may share some prior immediate event executions, which can be obtained from the entries of the vector time that are equal and have an associated priority vector entry smaller than infinity. The decision on which event has to be executed first must be based then on the minimal priorities of action executions that have taken place since the paths split up.

$$\begin{aligned}&\forall e_1, e_2 \in E :\\ &\quad \big((T_1 = T_2) \wedge (V_1 \parallel V_2) \wedge (P^{1,2}_{\min} > P^{2,1}_{\min})\big)\\ &\qquad\qquad\qquad\qquad \longrightarrow \big(e_1 \to e_2\big)\end{aligned}$$

This is captured exactly in the computation of the minimal path priority: the event that took the path with a smaller lowest priority will always be ordered after another event. The '$<$' relation for compound simulation times covers this case accordingly, leading to $ct(e_1) < ct(e_2)$.

The final case occurs, if in a setting as described above the smallest priorities of the paths occurred in their last common atomic unit. The two event executions in this common atomic unit are different ones for the two paths, because their vector times would otherwise be equal, and thus their priority would not have been taken into account for the minimal path priority. Obviously there must have been a unique ordering of these two previous events, that started the different paths. This ordering is simply given by the sequence of executions in the common atomic unit, which can be directly deduced from the corresponding entry in the vector time.

$$\begin{aligned}\forall e_1, e_2 \in E : \big((T_1 = T_2) \wedge (V_1 \parallel V_2) \wedge&\\ \big[\exists au \in AU : P^{1,2}_{\min} = P^{2,1}_{\min} = P(au)\big] \wedge&\\ \big[V_1(au) < V_2(au)\big]\big) \qquad \longrightarrow \big(e_1 \to e_2\big)&\end{aligned}$$

This case is covered in the bottom part of Equation 2, and ensures that $ct(e_1) < ct(e_2)$.

There are no other cases in which two events of a confusion-free model can be in the '$\to$' relation of a sequential simulation.

With our choice of compound simulation time even the converse condition of Equation 3 is true:

$$\forall e_1, e_2 \in E : \big(ct(e_1) < ct(e_2)\big) \longrightarrow \big(e_1 \to e_2\big) \qquad (4)$$

Indirect proof: Assume we find $e_1, e_2 \in E$ such that $ct(e_1) < ct(e_2)$ and not $e_1 \to e_2$. With our assumption of unique event priorities for different atomic units, it is always clear which event has to be executed first in a sequential simulation. Relation '$\to$' over events is thus trichotomous, and thus $\neg(e_1 \to e_2) \longrightarrow \big((e_2 \to e_1) \vee (e_1 = e_2)\big)$. Obviously $ct(e_1) = ct(e_2)$ if $e_1 = e_2$, which contradicts our assumption and leaves the case $e_2 \to e_1$. From the clock condition in Equation 3 it follows that $ct(e_2) < ct(e_1)$, which contradicts the assumption as well (asymmetry of '$<$' is shown in Section 6.2 below). □

Ordering based on compound simulation times of our distributed algorithm thus ensures that events are processed in exactly the same way as in a sequential simulation.

The mapping of events to compound simulation times is obviously a function: Every individual event is generated at an atomic unit, which increases its local vector time (and possibly the simulation time) during the process. The local entry thus reaches a new maximum value, which becomes a part of the new event's time stamp. There are no two events with the same vector time for the same reason. It follows that the mapping of events to compound simulation times is *bijective*, i.e.

$$\forall e_1, e_2 \in E : \big(e_1 = e_2\big) \iff \big(ct(e_1) = ct(e_2)\big)$$

From the bijectivity and the corollaries given with Equations 3 and 4 it follows that the event set E with the '$\rightarrow$' relation is isomorphic to the compound simulation times CT with the '$<$' relation, and

$$\forall e_1, e_2 \in E : \big(e_1 \rightarrow e_2\big) \iff \big(ct(e_1) < ct(e_2)\big) \quad (5)$$

The compound simulation times can thus be used for a correct and unique decision about the ordering of events.

6.2 Compound Simulation Time Properties

There are some conditions that any model of time should adhere to (compare e.g. [11]), which we will check in the following for the compound simulation time. We will thus show that the '$<$' relation defined in Equation 2 satisfies irreflexivity, asymmetry, transitivity, linearity (more exactly trichotomy), eternity, and density.

It should be noted that it makes no sense to analyze compound simulation time entities with arbitrarily set values; we restrict ourselves to time stamps of events that could possibly be obtained during a distributed simulation of a real SDES model as described above.

Irreflexivity of the '$<$' relation:

$$\forall ct_1 \in CT : \neg(ct_1 < ct_1)$$

Assume that $ct_1 \in CT$ can be found such that $ct_1 < ct_1$ for an indirect proof. The elements of identical compound simulation times are of course equal; thus neither $T_1 < T_1$ nor $V_1 < V_1$ will ever be true. In addition to that, $P^{1,1}_{\min} = P^{1,1}_{\min} = \infty$ because $V_1 = V_1$. It is thus impossible to find an atomic unit satisfying the two final lines of Equation 2, which is a contradiction to the assumption. □

Asymmetry of '$<$'.

$$\forall ct_1, ct_2 \in CT : (ct_1 < ct_2) \longrightarrow \neg(ct_2 < ct_1)$$

Indirect proof: assume we find $ct_1, ct_2 \in CT$ satisfying $(ct_1 < ct_2) \wedge (ct_2 < ct_1)$. If $T_1 \neq T_2$, the decision about which time is smaller would be based on the simulation times and unique (asymmetry of '$<$' for real numbers), thus $T_1 = T_2$. With similar arguments it follows that $V_1 \parallel V_2$, and $V_1 \neq V_2$ because otherwise $ct_1 = ct_2$ and neither one would be smaller.

Due to the differing vector times it is always possible to obtain unique minimal path priorities for ct_1 and ct_2. If we would have $P^{1,2}_{\min} \neq P^{1,2}_{\min}$, the '$<$'-relation would be true only for one comparison. It thus follows that $P^{1,2}_{\min} = P^{1,2}_{\min}$. Because we adopted unique event priorities, there is exactly one atomic unit au for which $P^{1,2}_{\min} = P^{1,2}_{\min} = P_1(au) = P_2(au)$ holds. However, we know that $V_1(au) \neq V_2(au)$, because this entry would otherwise have been ignored for the derivation of the minimal path priority. Thus either $V_1(au) < V_2(au)$ or $V_2(au) < V_1(au)$. This means that only one of the '$<$'-relations between ct_1 and ct_2 is true, leading to a contradiction to our assumption. □

Trichotomy of '$<$' for compound simulation times of events: two values are either equal, or exactly one is smaller than the other ($\oplus$ denotes "exclusive-or").

$$\forall ct_1, ct_2 \in CT : (ct_1 = ct_2) \oplus (ct_1 < ct_2) \oplus (ct_2 < ct_1)$$

Let us consider the case $ct_1 = ct_2$ first. The equation is fulfilled because neither $ct_1 < ct_2$ nor $ct_2 < ct_1$, which follows directly from irreflexivity.

It remains to prove that if $ct_1 \neq ct_2$, either $ct_1 < ct_2$ or $ct_2 < ct_1$ holds. There are two parts to this proof: first, we must show that never $ct_1 < ct_2$ and $ct_2 < ct_1$, which we have already shown (asymmetry). We thus only have to show that the '$<$'-relation is **linear** (in the restricted sense of an irreflexive relation):

$$\forall ct_1, ct_2 \in CT : \\ (ct_1 \neq ct_2) \longrightarrow \big((ct_1 < ct_2) \vee (ct_2 < ct_1)\big)$$

The proof is similar to the one for asymmetry. Assume $T_1 \neq T_2$: then either $ct_1 < ct_2$ or $ct_2 < ct_1$ (trichotomy of '$<$' for real numbers). We thus only need to consider the case $T_1 = T_2$. We know that $V_1 \neq V_2$ because $ct_1 \neq ct_2$. Assume now further that $V_1 < V_2$ or $V_2 < V_1$: then obviously $ct_1 < ct_2$ or $ct_2 < ct_1$, and the proposition holds. It thus remains to show that it is also true in the case $V_1 \parallel V_2$.

The minimal path priorities are now inspected: if $P^{1,2}_{\min} \neq P^{2,1}_{\min}$, the proposition becomes true. What happens if $P^{1,2}_{\min} = P^{2,1}_{\min}$? There is exactly one atomic unit au for which the minimal path priority is achieved ($P^{1,2}_{\min} = P^{1,2}_{\min} = P_1(au) = P_2(au)$), because of the use of globally unique priorities. However, $P_1(au)$ has been considered in the computation of the minimal path priority, which means that $V_1(au) \neq V_2(au)$. Thus one of the entries is less than the other (trichotomy of '$<$' for naturally numbered priorities), and the proposition is thus true in this final case as well. □

Transitivity of '$<$' for compound simulation times, i.e.

$$\forall ct_1, ct_2, ct_3 \in CT : \\ \big((ct_1 < ct_2) \wedge (ct_2 < ct_3)\big) \longrightarrow (ct_1 < ct_3)$$

Application of Equation 4 to the two conditions leads to $(e_1 \to e_2) \wedge (e_2 \to e_3)$. The relation '$\to$' on events $e_i \in E$ is obviously transitive: if an event e_1 has to be executed before e_2, and e_2 before e_3 in a sequential simulation, e_1 needs to be executed before e_3 as well. Thus we conclude that $(e_1 \to e_3)$, from which $ct_1 < ct_3$ follows with Equation 3. □

Eternity of a time scheme means that there is always a smaller and a greater time point for any given value.

$$\forall ct_1 \in CT \; \exists ct_2, ct_3 \in CT : ct_1 > ct_2 \wedge ct_1 < ct_3$$

This is obviously true because already for the simulation time part T, which is a real number, there is always a smaller and a greater value. □

Density is a similar case: there is always a compound simulation time between any pair of different times, because the real-valued simulation time entries are dense as well.

$$\forall ct_1, ct_3 \in CT \; \exists ct_2 \in CT : \\ (ct_1 < ct_3) \longrightarrow \big((ct_1 < ct_2) \wedge (ct_2 < ct_3)\big)$$

As a conclusion, the relation '$<$' on CT is a *strict total order*, because it has been shown to be irreflexive, trichotomous, and transitive. Moreover it is a *well-founded relation*, thus ensuring that for any non-empty subset of CT there is a unique '$<$'-minimal element. This property is necessary for each atomic unit when the future event with the smallest associated time has to be selected for execution from the event list.

7 GLOBAL STATES

A common problem in distributed algorithms (and thus simulations) is to determine a global state over the numerous local states of each process. Access to global state information is necessary in a SDES model at several places. Guard functions of colored Petri nets are an example for a state-dependent enabling of actions. Issues like a maximum capacity lead to a specification of a state condition for a state variable. Finally, the derivation of performance measures requires to obtain state variable values at certain time points as well as the execution times of actions.

An evaluation of any expression with parameters that depend on remote state variables requires a correct computation of a global state for the exact point in time it is requested for. This leads to an additional problem: the atomic unit in which the state variable is maintained might not have reached the simulation time for which the state is requested. In that case we follow the idea of optimistic simulation by assuming that the state will not change until the requested time. If it does so later, the affected atomic unit is notified and rolls back accordingly.

State variable access is however only locally possible in a distributed simulation. The issue of deriving information about remote states at a given point in time is known as a *global predicate evaluation problem* [13]. Due to the possibly different speed and numbers of events to be processed at each atomic unit, the local simulation times may significantly vary. Information about remote states can thus be obsolete, incomplete, or inconsistent. In the general setting of distributed algorithms this lead to the development of methods to obtain a global state, which use only the causality relation between message sending and reception as well as the sequence of event executions of local processes [14, 11, 13].

A global state of a distributed simulation consists of a set of local states, one for each logical process (i.e. atomic unit in our approach). Every local state associates a value to each locally maintained state variable. A state is valid between two event executions at the atomic unit due to the nature of discrete-event systems. It is thus possible to talk about events or states when analyzing the correctness of a global state.

Events in a distributed system are often visualized in a *space-time diagram* (see e.g. [1, 13]), in which the event and state sequences of each atomic unit are sketched in horizontal time-lines. The different lines model the spatial distribution of the processes, and messages between the processes and thus causal relationships can be drawn as arrows between the horizontal lines. If we select a local state for each atomic unit in the graph and connect all these points by a zigzag line, we have a graphical representation of a global state. Every global state that an observer may obtain can obviously be depicted in that way. The strong relationship between events and states in that aspect is clear because every state can be uniquely identified by its rightmost predecessor event.

Such a state-connecting line cuts the sets of events at each atomic unit into a past and a future set. It is thus called a *cut* C and defined as a finite subset of an event set E such that for every event e in it, all events are also included which were executed before e locally in the atomic unit producing e [11]. $E(au)$ denotes the set of events belonging to atomic unit au.

$$C \subseteq E \quad \text{subject to} \quad \forall au \in AU, \forall e \in E(au) \in C : \\ (e' \to e) \longrightarrow (e' \in C)$$

A cut thus respects local causality: all events left of the cut in the space-time diagram are included. However, not every one of these possible observations corresponds to a consistent state.

A cut C is *consistent* if it respects global causality as well: for every event e in the cut, all events that causally precede it are included in C [11].

$$\forall e \in C : (e' \to e) \longrightarrow (e' \in C) \tag{6}$$

In a distributed algorithm, causality is related to local event execution and message sending and reception. It is thus required that if the receive event of a message has been recorded in the state of a process, then its send event is also recorded in the state of the sender [1]. This property can be checked graphically in the time-space diagram: if there is an arrow (modeling a message transfer) which crosses the cut line backwards, the cut is not consistent.

A global state in a distributed computation is consistent, if it belongs to a consistent cut. A consistent cut corresponds to a state that is *possibly* observed in a run of the algorithm, but not necessarily reached. Consistent cuts and thus states are not unique: it is possible to add or delete events that are concurrent to all other events in remote atomic units, without interfering with consistency.

This is not acceptable in our environment for SDES performance evaluation, where the causal, timing and priority relations between events must be obeyed correctly. We have already shown above that the introduced compound simulation time CT allows to order events exactly. It is verified in the following that our use of CT leads to consistent global states.

Every inquiry about remote states corresponds to a compound simulation time ct, for instance resulting from the enabling-check of an action at a certain local simulation time of its atomic unit. Depending on the result, an event $e = (ct, \cdot)$ might be executed at time ct. The remote states that are valid at this time point correspond to a cut C_e. Every atomic unit can easily decide which of its own events belongs to C_e based on the compound simulation times.

$$\forall au \in AU, e \in E(au) : \\ \big(e \in C_e\big) \iff \big(e \in Past_{ct}^{EvList^{au}}\big)$$

In the equation, $Past_{ct}^{EvList^{au}}$ denotes the part of the local event list $EvList^{au}$ of atomic unit au that lies in the past of ct.

The past of the event lists contains all events with a compound simulation time before ct, and the cut C_e is derived as the union of the local event sets.

$$C_e = \bigcup_{au \in AU} Past_{ct}^{EvList^{au}}$$

Such a cut is obviously unique by construction, because the membership of events to $Past_{ct}^{EvList^{au}}$ is well-defined due to the trichotomy of '$<$' for compound simulation times. We can thus deduct that an event is in the cut iff it was executed before ct.

$$\forall e' \in E : \big(ct(e') < ct(e)\big) \iff \big(e' \in C_e\big)$$

With the isomorphism between '$<$' for compound simulation times and '$\to$' for events (Equation 5) it follows directly that

$$\forall e' \in E : \big(e' \to e\big) \iff \big(e' \in C_e\big)$$

To check if a cut C_e is consistent (Equation 6), we assume that it is possible to find events $e', e'' \in E$ subject to

$$\big(e'' \in C_e\big) \quad \wedge \quad \big(e' \to e''\big) \quad \wedge \quad \big(e' \notin C_e\big)$$

From $e'' \in C_e$ we know that $e'' \to e$, which leads to $ct(e'') < ct(e)$. Moreover $ct(e') < ct(e'')$ because of $e' \to e''$. Thus also $ct(e') < ct(e)$ because of the transitivity of '$<$', and therefore $e' \in C_e$ contradicting the assumption. □

Every global state that is constructed for a certain compound simulation time is thus consistent. It should however be noted that this discussion assumes a "correct" simulation that does not violate the local causality constraints; the rollback mechanism ensures that other runs are taken back.

8 CONCLUSIONS

We have discussed the properties of *compound simulation time* in the paper, which has been proposed as a logical time for distributed simulation recently. It allows the correct ordering of events in the presence of zero delays and global event priorities, and can thus be used for a fine-grained model partitioning of stochastic discrete event systems. The paper shows that the proposed logical time adheres to common models of time, that it imposes a strict total order on time-stamped events, and that correct global states can be obtained using it.

REFERENCES

[1] Lamport, L. 1978, "Time, clocks, and the ordering of events in a distributed system." Communications of the ACM, 21, no. 7: 558–565.

[2] Jefferson, D. 1985, "Virtual time." ACM Transactions on Programming Languages and Systems (TOPLAS), 7, no. 3: 405–425.

[3] Fujimoto, R. M. 1990, "Parallel Discrete Event Simulation." Communications of the ACM, 33, no. 10: 30–53.

[4] Misra, J. 1986, "Distributed Discrete-Event Simulation." ACM Computing Surveys, 18, no. 1: 39–65.

[5] Jefferson, D.; Sowizral, H. 1985, "Fast Concurrent Simulation Using the Time Warp Mechanism." In P. Reynolds, ed., *Distributed Simulation '85*, SCS, La Jolla, California, 63–69.

[6] Knoke, M.; Kühling, F.; Zimmermann, A.; Hommel, G. 2004, "Towards Correct Distributed Simulation of High-Level Petri Nets With Fine-Grained Partitioning." In J. Cao, ed., *2nd Int. Symposium on Parallel and Distributed Processing and Applications (ISPA'04)*, Springer Verlag, Hong Kong, China, vol. 3358 of *Lecture Notes in Computer Science*, 64–74.

[7] Knoke, M.; Kühling, F.; Zimmermann, A.; Hommel, G. 2005, "Performance of a Distributed Simulation of Timed Colored Petri Nets with Fine-Grained Partitioning." In *Design, Analysis, and Simulation of Distributed Systems Symposium, (DASD 2005)*, San Diego, USA.

[8] Nicol, D. M.; Mao, W. 1995, "Automated parallelization of timed Petri-net simulations." Journal of Parallel and Distributed Computing, 29, no. 1: 60–74.

[9] Chiola, G.; Ajmone Marsan, M.; Balbo, G.; Conte, G. 1993, "Generalized Stochastic Petri Nets: A Definition at the Net Level and Its Implications." IEEE Transactions on Software Engineering, 19, no. 2: 89–107.

[10] Zeng, Y.; Cai, W. T.; Turner, S. J. 2003, "Causal Order Based Time Warp: A Tradeoff of Optimism." In D. M. Ferrin; D. J. Morrice, eds., *Proceedings of the 35th Winter Simulation Conference (WSC'03)*, ACM, New Orleans, LA, USA.

[11] Mattern, F. 1989, "Virtual Time and Global States of Distributed Systems." In M. Cosnard, ed., *Proc. Workshop on Parallel and Distributed Algorithms*, Elsevier Science Publishers, 215–226.

[12] Fidge, C. 1991, "Logical Time in Distributed Computing Systems." Computer, 24, no. 8: 28–33.

[13] Babaoğlu, O.; Marzullo, K. 1993, "Consistent Global States of Distributed Systems: Fundamental Concepts and Mechanisms." In S. Mullender, ed., *Distributed systems (2nd Ed.)*, Addison-Wesley, New York, NY, USA, 55–96.

[14] Chandy, K. M.; Lamport, L. 1985, "Distributed Snapshots: Determining Global States of Distributed Systems." ACM Transactions on Computer Systems, 3, no. 1: 63–75.