# Towards Correct Distributed Simulation of High-Level Petri Nets with Fine-Grained Partitioning

Michael Knoke, Felix Kühling, Armin Zimmermann, and Günter Hommel

Technische Universität Berlin
Real-Time Systems and Robotics
Einsteinufer 17, 10587 Berlin, Germany
knoke@cs.tu-berlin.de

**Abstract.** Powerful grid and cluster computers allow efficient distributed simulation. Optimistic simulation techniques have been developed which allow for more parallelism in the local simulations than conservative methods. However, they may require costly rollbacks in simulation time due to dependencies between model parts that cause violations of global causality. Different notions of time have been proposed to detect and remedy these situations. Logical time (or Lamport time) is used in many present-day distributed simulation algorithms. However, high-level colored Petri nets may contain global activity priorities, vanishing states, and global state dependencies. Thus virtual time is not sufficient to maintain the global chronological order of events for the optimistic simulation of this model class. The paper presents a new approach that guarantees a correct ordering of global states in a distributed Petri net simulation. A priority-enhanced vector time algorithm is used to detect causal dependencies.

## 1 Introduction

Stochastic Petri nets (PN) have been widely used for modeling the behavior of systems where synchronization of processes is crucial [1]. They provide a graphical representation and are able to represent discrete events as well as (stochastic) timing. Our simulation framework uses a variant of *colored* Petri nets (CPN) [2].

Real world systems consist of parts widely showing autonomous behavior but cooperating or communicating occasionally. This inherent concurrency and required synchronization can be modeled adequately using PNs. Distributed Petri net simulation (DPNS) can exploit this inherent parallelism efficiently using grid- and cluster computers. Hence, a partitioning algorithm is required that decomposes the model such that heavily communicating elements are not split. Each decomposed PN submodel is assigned to a *logical process* (LP) that is performing the simulation on a physical processor. A logical clock that denotes how far the simulation has progressed is assigned to a LP as well. LPs communicate using timestamped messages [3].

There has been significant work in the area of distributed simulation of PNs in the past few years. Almost all proposed algorithms assume a virtual time with an arbitrary high resolution to eliminate isochronous events. Some model specific activities can also cause events with the same virtual time, even for an assumed infinite resolution of time. Some of the activities in high-level PNs are:

- immediate transitions resulting in state changes without simulation time progress
- deterministic transitions that have a deterministic delay for state changes
- time guard functions which trigger state changes at a certain point in time

These properties of high-level PNs are either not allowed or adequately distributed to LPs, so that they are sequentially processed. Nicol and Mao [4] have contributed one of the most complete publications on distributed simulation of PNs, showing this limitation in each presented algorithm. It is obvious that in these cases the event ordering is simple and most research is focused on preferably good partitioning algorithms and early rollback detection. PN models for real world systems, such as detailed workflow modeling, may contain more than 50 percent timeless or deterministic activities.

A basic problem of distributed simulation is to avoid causality errors. Correctness of simulation can only be ensured if the (total) event ordering as produced by a sequential simulation is consistent with the (partial) event ordering due to distributed execution. Indeed, Jefferson [5] recognized this problem to be the inverse of Lamport's logical clock problem [6], i.e. providing clock values for events occurring in a distributed system such that all events appear ordered in logical time.

Lamport's algorithm allows to maintain time ordering among events [7]. However, a mapping from Lamport time to real time is not possible. Furthermore it is not sufficient to characterize causal relationships between events. But the detection of causal relationships between events is indispensable for transition priorities. Otherwise it is not possible to sort concurrent and independently fired events whose occurrence is based on a different priority. The Lamport time would impose an artificial order independent of their priority.

A logical time that characterizes causality and can be used to remedy last named problems is the *vector time* (VT) proposed by Mattern [8] and Fidge [9]. The causal relationships between events can be determined from their corresponding VT values. VT allows to detect indirect dependencies, that means comparing two VTs of different events provides information whether these events are causally dependent and if so, which event depends on which one. This has the following advantages in the context of DPNS:

- concurrent events can be identified and sorted by their priorities
- a very fine-grained model partitioning allowing deterministic and zero-firing times for output transitions of LPs is possible
- precise recovery of local LP states based on external events
- no need to solve equal Lamport time stamps

Many different high-level colored PN model classes, our class as well, allow different priorities for immediate transitions. That means if two events could be created at the same simulation time, the higher prioritized event is permitted first and may disable the second event through its occurrence. An example of a simple PN model is presented in Fig. 1. Transitions *Order* and *SendPart* are concurrently enabled and have different priorities, so that *Order* is processed first because of its higher priority. A sequential simulation is simple but a distributed simulation where both transitions fire optimistically, requires an order of execution.

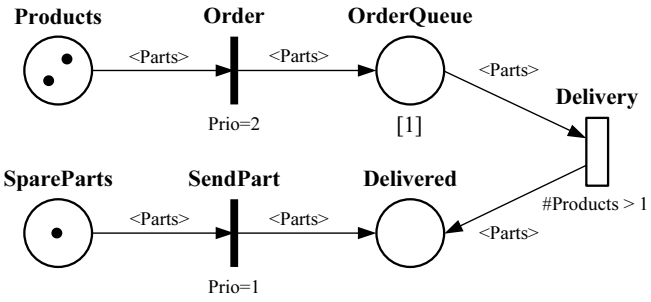


**Fig. 1.** Example of a high-level colored Petri net model

To the best of the authors knowledge this paper presents the first time a new logical time scheme for high-level PNs which has significant advantages for partitioning without any structural limitations. It offers correctness for isochronous events and is applicable for all types of PNs, even for timeless PNs. Our extensions to the logical time fulfil today's requirements for flexibility and maximum scalability for typical real world PN models. It is not the intention of this paper to compare performance measures with any of the numerous Time Warp variations for distributed simulation of PNs. Optimistic simulation of high-level PNs is, in contrast to PDES, heavily dependent on the abilities of the underlying net class. It's always possible to design PN models perfectly fitting to a given distributed simulation algorithm. Our objective in this paper is to show new algorithms for partitioning and distributed event processing based on a new logical time scheme that opens new possibilities for DPNS performance optimization.

The paper first presents our new partitioning approach in Sect. 2. The subsequent Sect. 3 introduces a logical time scheme for prioritized globally ordered states. Some information about successfully completed test scenarios are shortly presented in Sect. 4 and finally concluding remarks are given in Sect. 5.

## 2 A New Partitioning Approach

Based on the correct implementation of causal dependencies that is described later in Sect. 3, the following scheme of an event-driven distributed simulation

for high-level colored PNs was developed. Rollbacks can be performed more precisely and the flexibility of the partitioning is higher in particular if prioritized transitions and isochronous states are used in the model.

The simulation is composed of $N$ sequential event driven LPs that do not share memory and operate asynchronously in parallel. Unlike in other optimistic DPNS algorithms (e.g. introduced in [10]), an *atomic unit* (AU) is defined as the smallest indivisible part of the model, whereas a LP consists of one or more of these AUs. The basic architecture and formalism of the LPs and AUs used in this paper is:

- The smallest indivisible part of the model is an atomic unit $AU$.
- A transition $T_i$ is inseparably linked with all of its input places ${}^\bullet T_i$ and constitutes an atomic unit $AU$. This can lead to situations where more than one transition will be assigned to one $AU$, namely if a place has several output transitions.
- At least one $AU$ is assigned to every $LP_i$ which is running as a process on one physical node $N_i$.
- A communication interface attached to the $LPs$ is responsible for the propagation of messages to the remote $LPs$ and to dispatch incoming messages to local $AUs$. $AUs$ on the same $LP$ are communicating directly to avoid additional message overhead.
- Each $LP_i$, $AU_j$ has access only to a partitioned subset of the state variables $S_{P,i} \subset S$ and $S_{U,j} \subset S_{P,i}$, disjoint to state variables assigned to other $LPs$, $AUs$. State variables of $LP_i$ are the set of state variables of all local $AUs$ $S_{P,i} = \bigcup S_{U,j} (\forall j)$.
- The simulation of local $AUs$ scheduled within each $LP$ in a way that avoids local rollbacks.

The three basic items for event-driven DPNS are state variables which denote the state of the simulation model, an event list that contains pending events, and a simulation clock which keeps track of the simulation's progress. All of these parts have been integrated into the AUs. Only two basic messages are required for simulation progress of AUs: *positive event messages* for token transfers and *negative event messages* to perform a rollback to an earlier simulation time.

A fine-grained partitioning and a discrete storage of processed states have a bunch of advantages for DPNS. First of all, in contrast to existing DPNS algorithms, e.g. described by Chiola and Ferscha [11], a rollback of complete LPs will not happen. Each AU has it's own virtual simulation time and stores its state for each local event independently from other AUs. This characteristic is essential for migration to other LPs at runtime. AUs can restore their state accurately for a given simulation time and send rollback messages to other AUs if they are affected by this rollback. Thus, rollbacks are much more precise and unnecessary rollbacks are prevented if independent AUs are simulated by a single LP. Memory consumption is lower than the classical LP approach because rarely executing AUs don't need to save their states until their own net activity.

Very important for collecting the result measures is the discrete storage of processed states. This storage mechanism allows to revert exactly to a given

logical time without needing to resimulate already simulated sequences. In case of a rollback the last valid state is found with absolut precision. The disadvantage of a higher memory consumption is compensated by the much smaller size of AUs.

## 3 A Logical Time Scheme for Prioritized High-Level Distributed PN Simulation

In this section a logical time scheme for DPNS is presented and studied in detail. As per description in Sect. 1 it is essential for a correct ordering of states if model characteristics allows prioritized transitions and isochronous concurrent states. A distributed simulation is correct if its simulation results match the results of a traditional, single process simulator. Such sequential simulations are processing events in the order that takes the simulation time and the event priority into account. As a consequence we can conclude that a DPNS is correct if each AU is processing events in the traditional sequential manner and if incoming events are sorted exactly as they would be generated by a single process simulator. The following section presents an expanded logical time to fulfill these demands.

### 3.1 Event Priorities

For PN simulations on a single processor it is sufficient to have one global simulation time with an arbitrarily low resolution. All activities are running in succession and are responsible for the time increment. The simulated order of events is identical to the order in which they are simulated. Conflicts of concurrent activities are resolved by priorities or by random selection.

Immediate transitions have a priority greater than 0 and timed transitions have an implicit priority of 0. These priority values must be valid across AU borders, that means if transitions on different AUs are concurrently firing isochronously, the corresponding events must be ordered by their priority. Among identically enabled transitions one is chosen to fire first non-deterministically. For distributed simulation this approach is nonapplicable because of consistency reasons. Independent random generators on the AUs cannot guarantee the same ordering. Therefore we have decided to define a new *global event priority* (GEP) that includes the AU number into the priority value to determine an explicit relation for two equal event priorities. GEP is calculated as follows:

$$GEP = P_E * N_{AU} + i_{AU} \qquad \begin{array}{l} P_E : \text{ event priority} \\ N_{AU} : \text{ AU count} \\ i_{AU} : \text{ current AU no.} \end{array} \tag{1}$$

GEP forces the same global event ordering for concurrent events with different event priorities as a sequential simulation, but events with the same priority are ordered by the AU number in which they are created. A random selection of

equal prioritized concurrently enabled transitions is non-applicable for DPNS. It forces a synchronization of model parts which acting completely autonomous. We have decided to accept this limitation because some people identify this problem as a modeling mistake.

Calculating the event priority $P_E$ from the transition priority is nontrivial. The following order would be achieved by a sequential simulation of the model in Fig. 2: $T2 \rightarrow T4 \rightarrow T1 \rightarrow T3$. $T0$ is firing first and afterwards $T1$ and $T2$ are simultaneously enabled but $T2$ fires because of its higher priority. Now, without any simulation time elapsed, $T1$ and $T4$ are in conflict and $T4$ fires. Subsequently $T1$ and $T3$ fire in succession without taking the priority values into account. This example looks simple but it is observable that in case of a distributed simulation the firing order requires global knowledge.

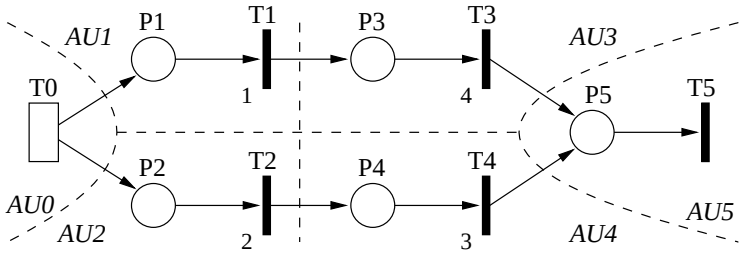


**Fig. 2.** An example for transition priorities

An optimistic distributed simulation doesn't need to resolve this priority problem when it appears but at the time when affected tokens are inserted into a place. This happens if at least two concurrent isochronous tokens must be ordered according to their priority. If the priority of the last fired transition was directly used to calculate the GEP it would give the token from path $T1 \rightarrow T3$ a higher order of precedence in the event queue because it was last fired from $T3$ which has a higher priority than $T4$.

To get the correct result it is important to create a priority path (herein after called *critical path*) from the last common transition or from the last timed transition. All priorities on each path must be considered for later event ordering. It can be shown that the minimum priority $P_{min}$ of each path is decisive because the transition with the lowest priority delays the propagation of an event until no other transition with a higher priority on other paths can fire. Using the minimum priority on both paths would deliver the correct result ($P_{min_{T1,T3}} = 1, P_{min_{T2,T4}} = 2$).

An AU-sized vector of the last firing priority of each AU would be needed for calculating the minimum priority on the critical path. Events within an AU are always sequentially ordered, so it is not required to store the priorities of all transitions. This *priority vector* $p(e)$ has to be assigned to each event $e$. It is

defined as follows:

$$p(e)_i = \begin{cases} \infty \quad \text{in case that } AU_i \text{ is not on the critical path} \\ \\ \text{otherwise the minimum priority of all preceding events on the} \\ \text{critical path of } e \text{ in } AU_i \end{cases} \tag{2}$$

To follow the path of AUs that a token has entered and to compute the minimum priority of this path, it is just required to compute the minimum value of the priority vector. It is a precondition that all components of this vector are set to the infinite value on initialization and if a timed transition fires. On equality of two calculated minimum priorities it is obvious that a specific AU which is on both paths has randomly defined the order. For $n$ AUs it must been $AU_i$ with $i = p_{min}\ mod\ n$, as derivable through (1). The order is then explicitly observable by the corresponding VT component.

Assuming that two concurrently fired isochronous events arrived at $AU5$ with $(VT), [P_E]$:

$$E_1 := (1,1,0,1,0,0), [-,1,-,4,-,-]$$
$$E_2 := (1,0,1,0,1,0), [-,-,2,-,3,-]$$

The VT indicates that both tokens are concurrent, but the minimum priority of $E_1$ is lower than the priority of $E_2$. As a result $E_1$ must be sorted after $E_2$.

### 3.2 Compound Simulation Time

Distributed PN simulations running on several processors in parallel, require a logical time to detect causal dependencies and to achieve a global order of events. Certainly, the simulation progress of the distributed simulation is further on driven by the *simulation clock time* which progresses independently on each AU. PN model specific characteristics and a limited resolution of this time permit the occurrence of isochronous events. To operate with these events this time is extended by a sufficient logical time, namely the VT and the GEP introduced in Sect. 3.1. The compound time is capable of processing these isochronous events and can detect all causal dependencies. The new logical time is the *simulation time* (ST) as defined in (3), with the corresponding ordering relation (4).

$$\begin{array}{l} ST = (T, V, G) \\ T: \text{ Simulation clock time} \\ V: \text{ Vector time} \\ G: \text{ Global event priority} \end{array} \tag{3}$$

$$\begin{array}{ll} u \leq v & \Leftrightarrow \\ & (T_u < T_v) \vee \\ & ((T_u = T_v) \wedge ((V_u < V_v) \vee \\ & (V_u \parallel V_v \wedge G_u \geq G_v))) \end{array} \tag{4}$$

**Fig. 3.** Compound simulation time

### 3.3 Transitivity of the Relation

A global ordering relation requires transitivity to offer explicit sorting of events. Relation (4) proposed in the last section is not transitive if it is not using the priority path for the GEP. An example of a simple Petri net that creates non-transitive events is shown in Fig. 4. Transitions $T1$, $T2$, and $T3$ create isochronous concurrent events which have to be sorted before merging the corresponding tokens at place $P4$.

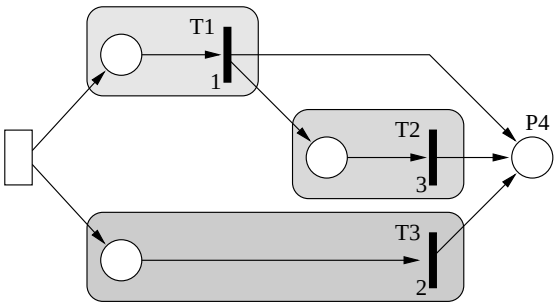


**Fig. 4.** Example petri net that can create non-transitive events

Assuming that the following three simulation times $S_1$, $S_2$ and $S_3$ have to be compared using (4):

$$\begin{aligned} S_1 &= (\text{2004-01-01 00:00:00}, [1,0,0], 1) \\ S_2 &= (\text{2004-01-01 00:00:00}, [1,1,0], 3) \\ S_3 &= (\text{2004-01-01 00:00:00}, [0,0,1], 2) \end{aligned} \tag{5}$$

All events have the same simulation clock time which is 2004-01-01 00:00:00. By comparing the VT values it can be observed that $S_2$ is causally dependent on $S_1$ but $S_3$ is concurrent to $S_1$ and $S_2$. $S_3$ has to be sorted with its priority value which is 2. The result is $S_1 \leq S_2 \leq S_3$ and due to the transitivity theorem should follow:

$$S_1 \leq S_2 \leq S_3 \Rightarrow S_1 \leq S_3$$

In fact it is:

$$S_3 \leq S_1$$

Events corresponding to the simulation times $S_1$ and $S_3$ are concurrent and originated from simultaneous and independently activated transitions. The event with timestamp $S_3$ must be fired first because of the higher priority. $S_2$ is causally dependent on $S_1$ and must be sorted behind $S_1$ and as a result behind $S_3$ even though $S_2 \leq S_3$.

**Theorem 1.** *If the correct global event priority (as depicted in Sect. 3.1) is used then (4) is transitive.*

*Proof.* Consider three events $e_i$ with $1 \leq i \leq 3$ and the corresponding priorities $p_i$ as well as the simulation time stamps $S_i$. We assume that all $S_i$ have the same simulation clock time. Then it is obvious that we have to account for the causal dependencies, namely the vector times and the priorities. The following notation is used for the causal dependency:

$$e_i \rightarrow e_j \Leftrightarrow e_j \text{ causally depends on } e_i$$
$$e_i \parallel e_j \Leftrightarrow e_i \text{ and } e_j \text{ are concurrent.}$$

Only if two events are concurrent their priorities have to be used for sorting. With this notation and on that condition (4) can be written as follows:

$$S_i \leq S_j \Leftrightarrow e_i \rightarrow e_j \vee (e_i \parallel e_j \wedge p_i \geq p_j) \tag{6}$$

It is necessary to show that this relation is transitive:

$$S_1 \leq S_2 \wedge S_2 \leq S_3 \Rightarrow S_1 \leq S_3 \tag{7}$$

If the correct GEP is used then it is clear that an event cannot have a higher priority than the event that it depends on. This constraint can be written as:

$$e_i \rightarrow e_j \Rightarrow p_i \geq p_j \tag{8}$$

Other helpful relationships directly deduced from (6) are:

$$e_i \rightarrow e_j \Rightarrow S_i \leq S_j \tag{9}$$
$$e_i \parallel e_j \wedge p_i \geq p_j \Rightarrow S_i \leq S_j \tag{10}$$
$$e_i \parallel e_j \wedge S_i \leq S_j \Rightarrow p_i \geq p_j \tag{11}$$

Furthermore the transitivity of the causal relationship $\rightarrow$ and the sorting relation for priorities $\leq$ is assumed.

In order to prove the transitivity it is essential to consider all possibilities to combine causal relationships and priorities of the three events. Implication (7) must be valid in all cases. First of all let's focus on the causal dependencies. The implication is fulfilled if the right side of the implication is true ($e_1 \rightarrow e_3$) or the left side is false ($e_1 \leftarrow e_2 \vee e_2 \leftarrow e_3$). The following eight cases remain:

1. $e_1 \rightarrow e_2 \wedge e_2 \rightarrow e_3 \wedge e_1 \leftarrow e_3$
2. $e_1 \rightarrow e_2 \wedge e_2 \rightarrow e_3 \wedge e_1 \parallel e_3$
3. $e_1 \rightarrow e_2 \wedge e_2 \parallel e_3 \wedge e_1 \leftarrow e_3$
4. $e_1 \rightarrow e_2 \wedge e_2 \parallel e_3 \wedge e_1 \parallel e_3$
5. $e_1 \parallel e_2 \wedge e_2 \rightarrow e_3 \wedge e_1 \leftarrow e_3$
6. $e_1 \parallel e_2 \wedge e_2 \rightarrow e_3 \wedge e_1 \parallel e_3$
7. $e_1 \parallel e_2 \wedge e_2 \parallel e_3 \wedge e_1 \leftarrow e_3$
8. $e_1 \parallel e_2 \wedge e_2 \parallel e_3 \wedge e_1 \parallel e_3$

The cases 1, 2, 3, and 5 contradict the transitivity of the causality relation and need not be considered further. In case 8 the events are sorted exclusively by their priorities whose ordering relation is assumed to be transitive. Only the cases 4, 6, and 7 remain and needs to be analyzed.

*Case 4.* We show that the right side of (7) must be true if the left side is true.

$$\left.\begin{array}{r} e_1 \rightarrow e_2 \Rightarrow p_1 \geq p_2 \\ e_2 \parallel e_3 \wedge S_2 \leq S_3 \Rightarrow p_2 \geq p_3 \end{array}\right\} \Rightarrow p_1 \geq p_3$$
$$e_1 \parallel e_3 \wedge p1 \geq p_3 \Rightarrow S_1 \leq S_3$$

*Case 6.* Analogous to case 4.

$$\left.\begin{array}{r} e_1 \parallel e_2 \wedge S_1 \leq S_2 \Rightarrow p_1 \geq p_2 \\ e_2 \rightarrow e_3 \Rightarrow p_2 \geq p_3 \end{array}\right\} \Rightarrow p_1 \geq p_3$$
$$e_1 \parallel e_3 \wedge p1 \geq p_3 \Rightarrow S_1 \leq S_3$$

*Case 7.* The right side of the implication is false. Assuming that the left side is true the transitivity would be violated. We show that this assumption is incorrect.

$$\left.\begin{array}{r} e_1 \parallel e_2 \wedge S_1 \leq S_2 \Rightarrow p_1 \geq p_2 \\ e_2 \parallel e_3 \wedge S_2 \leq S_3 \Rightarrow p_2 \geq p_3 \end{array}\right\} \Rightarrow p_1 \geq p_3$$
$$e_1 \leftarrow e_3 \Rightarrow p_1 \leq p_3$$

The outcome of this is $p_1 = p_3$. But concurrent events cannot have the same priority if unambiguous global priorities are used as depicted in Sect. 3.1. From this it follows that the left side of the implication must not be true. The transitivity is not violated.

So we can conclude that it is proven that (6) and as a result (4) are transitive.

## 4 Tests

In the course of our research and development we have designed a lot of models to verify our implementation of the new logical time scheme. The AU approach allows a truly distributed simulation of simple models to exploit parallelism. These simple models are not adequate for performance measurements but demonstrate the correctness of our approach. All experiments have been conducted on a 16 node, dual Intel Xeon processor, Linux cluster with 1 GB memory for each node. SCI has been used as a high-speed ring-topology-based networking.

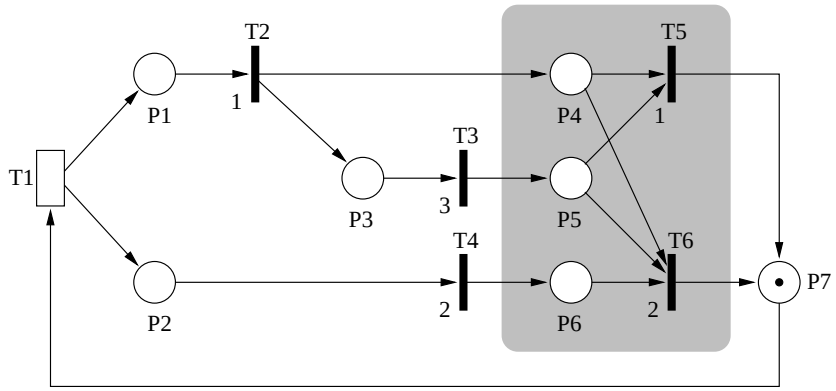


**Fig. 5.** Example petri net for testing transitivity

Figure 5 shows a modified version of the model in Fig. 4 to test transitivity. Transition $T5$ may not fire if correct event ordering is used. Running this model over a long simulation time with several million events shows that $T1$ and $T6$ fire equal times, but never $T5$.

More complicated models, which have been accrued for different research projects for global operative business companies, are already successfully tested but not shown here because of lack of space. Such models are mostly not qualified for verifying substantial characteristics of the newly developed mechanisms.

## 5 Conclusion

This paper presented a new mechanism for distributed simulation of high-level Petri Nets. We introduced the notion of prioritized logical time which allows for a mapping between simulation clock and logical time. Applied to high-level PNs, this logical time is sufficient to allow a fine-grained partitioning not possible with Lamports logical time. It can be viewed as a total ordering scheme for high-level PN events. The Petri net model is decomposed into atomic units which have an own virtual time. Plenty of advantages for the distributed simulation arise from this approach: a better partitioning flexibility, dynamic migration with low operational expense, and efficient rollbacks.

## References


1. Zimmermann, A., Freiheit, J., Huck, A.: A Petri net based design engine for manufacturing systems. Int. Journal of Production Research, special issue on Modeling, Specification and Analysis of Manufacturing Systems **39** (2001) 225–253
2. Jensen, K.: Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 1 : Basic Concepts. EATCS Monographs on Theoretical Computer Science, Springer-Verlag, Germany (1992)
3. Fujimoto, R.: Parallel and distributed discrete event simulation: algorithms and applications. In: Proceedings of the 1993 Winter Simulation Conference, Los Angeles, CA, Eds. ACM, New York, 1993 (1993) 106–114
4. Nicol, D.M., Mao, W.: Automated parallelization of timed petri-net simulations. Journal of Parallel and Distributed Computing **1** (1995)
5. Jefferson, D.: Virtual time. ACM Transactions on Programming Languages and Systems **7** (1985) 405–425
6. Lamport, L.: Time, Clocks, and the Ordering of Events in a Distributed System. Communications of the ACM **21** (1978) 558–565
7. Zeng, Y., Cai, W., Turner, S.: Causal Order Based Time Warp: A Tradeoff of Optimism. Proceedings of the 2003 Winter Simulation Conference (2003)
8. Mattern, F.: Virtual Time and Global States of Distributed Systems. Proceedings Parallel and Distributed Algorithms Conference (1988) 215–226
9. Fidge, C.: Logical Time in Distributed Computing Systems. Computer **24** (1991) 28–33
10. Ferscha, A.: Parallel and Distributed Simulation of Discrete Event Systems. McGraw-Hill (1995)
11. Chiola, G., Ferscha, A.: Distributed simulation of Petri Nets. IEEE Parallel and Distributed Technology **1** (1993) 33–50