

A NEW PARADIGM FOR THE NUMERICAL SIMULATION OF STOCHASTIC PETRI NETS WITH GENERAL FIRING TIMES

Graham Horton
Computer Science Department
University of Magdeburg
Universitätsplatz 2,
39106 Magdeburg, Germany
graham@cs.uni-magdeburg.de

KEYWORDS

Proxel, Stochastic Petri Net, State Space, Simulation, Supplementary Variables.

ABSTRACT

This paper is concerned with the simulation analysis of discrete-state stochastic models such as queueing systems or stochastic Petri nets, in which arbitrary probability distributions may be assigned to the activities. The analysis is performed on the state space using a numerical approach, rather than the usual discrete-event simulation at the model level. A new computational paradigm, the so-called *Proxel* (probability element) is introduced, which allows an approximation to the continuous stochastic process of the Petri net to be developed which does not require the use of differential equations. This proxel-based computational model directly yields a simulation algorithm which is readily understood and implemented. Simulation experiments are used to illustrate the behaviour of the method and to discuss the advantages and disadvantages of the method compared to the alternatives.

INTRODUCTION AND OVERVIEW

The goal of this paper is to present a new computational model for discrete-state stochastic models such as Stochastic Petri Nets (SPN) or queueing systems. This computational model is based on the concept of a *Proxel*, or probability element, as the basic unit of computation.

Discrete-state stochastic models are almost always analysed using discrete-event simulation, which directly mimics the behaviour of the model, using random numbers to obtain samples of the probability distributions of the activities and multiple replications of the simulation to obtain statistically useful results.

In principle, an alternative approach is possible, whereby the state space of the SPN is generated and a partial differential equation (PDE) is set up which describes the stochastic process of the model. This PDE can then be solved numerically to obtain information about the behaviour of the model. In practice, this approach is never used, owing to the significant difficulties involved in setting up and solving the PDE.

The new, proxel-based method approaches the problem of analysing the model from a different angle. A proxel represents a certain state of the model, together with its probability at a given point in simulation time. The method generates the state space of the model and computes as solution variables the probabilities of each state of the model. The simulation method then simply consists of tracing the path of probability as it moves around and is redistributed within the state space. Starting from any given proxel, successor proxels may be easily derived, which represent the reachable states of the model and their probabilities at subsequent points in simulation time. The iterative process of generating proxels is a deterministic simulation of the user model. By contrast to the computational model based on partial differential equations, the proxel-based approach is algorithmic, rather than analytic, although it is otherwise similar in many respects. For this reason, it directly yields a simulation technique which is comparatively easy to understand and implement.

The paper is organised as follows. In the next section, Stochastic Petri Nets and current methods for their simulation analysis are briefly described. We then introduce the Proxel formally and explain the instantaneous rate function and method of supplementary variables, which form the basis of the simulation algorithm, which is presented next. Various aspects of the algorithm are then discussed, in particular its memory requirements and computational complexity. Next, a simple SPN is used to illustrate the behaviour of the new algorithm and to allow a comparison with the alternative approaches. Finally, the conclusions are presented and directions for further research are suggested.

STOCHASTIC PETRI NETS (SPN)

Modelling with SPNs

Stochastic Petri Nets are a well known modelling paradigm which are used, for example, to model traffic and material flow, computer networks, and manufacturing systems, as well in safety and reliability modelling. They represent user models whose behaviour is stochastic and is discrete in both time and space. Some advantages of SPNs are that they allow graphical editing of the model, they are easy to understand, they are powerful and flexible, and they can be simulated automatically by a computer.

The term "Stochastic Petri Net" was originally used to denote those Petri nets in which all firing times were exponentially distributed (Molloy 1980). In this paper, the term "SPN" will be used to denote a Petri net whose transition firing times may have any distribution.

SPN Analysis with Discrete-Event Simulation

The simulation analysis of SPNs and similar user models, such as Stochastic Activity Networks (SAN) and queueing systems, is usually performed using a discrete-event simulation. These computational models use computer-generated pseudo-random numbers to sample the distributions of the random variables describing the activities in the model. They directly mimic the behaviour of the user model in that they compute directly on the states of the user model and are themselves discrete in time and space and are stochastic in nature. Owing to the stochastic nature of the simulation, multiple replications must be carried out, in order to obtain statistically useful results, for example in the form of confidence intervals.

The primary advantages of this computational model are that it is conceptually simple, and that it has very low memory requirements. The simplicity results from the similarity to the user model (it is also stochastic, and discrete in space and time); this allows discrete-event simulators to be implemented relatively easily. The memory requirements are proportional to the size of the user model, which, in the case of an SPN, is seldom larger than 100 places.

On the other hand, difficulties can arise, such as:

- Stiff models. Models can be stiff when they contain competing activities whose rates or probabilities vary strongly in magnitude. In such cases, the simulator will only rarely choose the slower or less probable event. This can lead to a significant loss of accuracy.
- Very high accuracy requirements. High accuracy is required, for example, in safety and reliability modelling, where failure probabilities are measured in fractions of one per cent.
- Measures with a large statistical variance. The random variable describing the quantity of interest in the model may have a large variance.

Models with one or more of these attributes can require a very large number of replications in order to achieve sufficient accuracy, this number may be as high as 10^5 or 10^6 . This can result in very long computation times for the simulation experiment. An additional difficulty is the uncertainty in the accuracy of the simulated result; in general, there is no way of knowing whether the number of replications performed is sufficient to accurately capture the behaviour of the model.

There are many examples of discrete-event Petri net simulators to be found in the literature. For example, Heller et al describe one such tool which is used to research safety and reliability issues in the automotive industry (Heller et al 2002).

SPN Analysis with Differential Equations

Alternatively, SPNs can be simulated using an analytical approach based on partial differential equations (PDEs). This approach is based on the method of supplementary variables described by Cox in (Cox 1955), and for SPNs by German in (German 2000). This method considers the density functions for the elapsed times of all transitions which are enabled in each marking of the SPN, and constructs a PDE describing the behaviour of these densities over time. The PDE can be derived by creating a balance equation for the probability flow into and out of each state. In order to describe the rate of flow of probability between discrete markings, supplementary variables are introduced, which represent the elapsed enabling times of the transitions, and the rate of probability flow is then given by the instantaneous flow rate of the transition's firing time. This is described in more detail for the proxel-based simulation method in the following section.

In contrast to the user model, this computational model is deterministic and continuous in both time and space. The resulting system of PDEs is linear, first-order and hyperbolic, with complex boundary conditions. This PDE is then discretised and solved numerically.

The primary advantage of this mathematical, analytic approach is that it yields a deterministic simulation algorithm. It is thus free from the above-mentioned problems resulting from the use of random numbers during the simulation. More specifically, no replications are necessary, and the accuracy of the result can be controlled easily, since it depends continuously on the discretisation step size. For many very simple models, a higher accuracy can be achieved than by using a discrete-event simulation, at lower cost and with a higher degree of certainty.

The disadvantages of this approach are twofold: memory requirements and complexity. The size of the state space grows exponentially with the number of concurrently enabled transitions; Even comparatively small and simple SPNs can yield computational models that exceed the available memory space. The equations to be solved are quite complex, making them hard to understand and to implement. For these reasons, to the author's knowledge, analysis based on PDEs has never been used in a general-purpose SPN simulator.

PROXELS

A *Proxel* (probability element) is the basic unit of computation that will be used to develop the new method for the analysis of SPNs and similar user models. Its name was chosen in analogy to the well-known *pixel* (picture element) from Computer Graphics. A pixel contains application information concerning a computer image (RGB values), together with coordinates (x and y coordinates on a computer screen) at which this information is located. In an analogous manner, a proxel contains application information concerning the state of a simulation model (a probability value p), together with its location (in

the state space of the SPN). As will be shown in the next section, the state S of an SPN consists of the marking m of the net, a vector of activation times τ for the transitions enabled in that marking, and the simulation time t .

Definition (Proxel):

A Proxel $P = (p, S)$ is a vector consisting of a state $S = (m, \tau, t)$ of an SPN, and a probability p , such that p (approximately) represents the probability that the SPN is in state S .

In order to derive a numerical simulation algorithm, we will define a discrete time step Δ . For any proxel $P = (p, S)$ at time t , successor proxels $P' = (p', S')$ at time $t+\Delta$ can be readily determined, which describe how the probability p is redistributed among successor states at the new time step.

INSTANTANEOUS RATE FUNCTION AND METHOD OF SUPPLEMENTARY VARIABLES

In order to construct the proxel-based simulation method, the method of supplementary variables is used. This approach is described for SPNs in more detail by German (German 2000).

We consider a random variable Φ , which is described by its cumulative distribution function $F(\tau)$ and corresponding probability density function $f(\tau)$. Then the instantaneous rate function $h(\tau)$ of Φ is defined as

$$h(\tau) = \frac{f(\tau)}{1 - F(\tau)}$$

The instantaneous rate function (IRF) represents the continuous rate of flow of probability for the random variable Φ . The function h is also known as the hazard rate or instantaneous failure rate.

Figure 1 (left) shows an SPN with places A and B and transition T . From the perspective of the user, the states of the system are defined by the markings of the SPN, and the state changes are stochastic and instantaneous. Figure 1 (right) illustrates the method of supplementary variables, which considers the probabilities π_A, π_B for each of the states A and B .

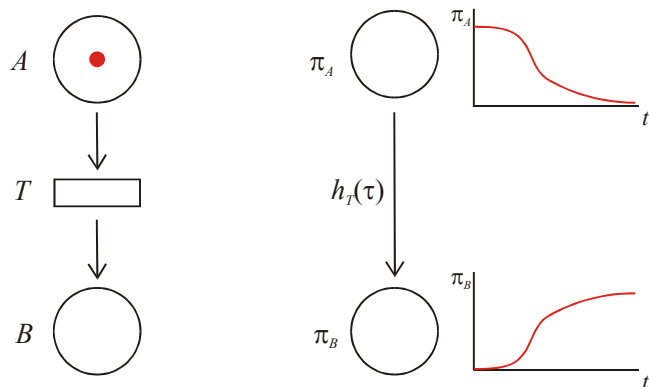


Figure 1: Dual View of the Stochastic Process

π_A and π_B are continuous variables which are governed by the equations

$$\frac{d\pi_B}{dt} = -\frac{d\pi_A}{dt} = h_T(\tau) \cdot \pi_A \tag{1}$$

$h_T(\tau)$ is the IRF of the firing time distribution of transition T , where τ is the elapsed enabling time of T . Note that in this example, τ is equal to t , since T becomes enabled once only at $t=0$. In general, this is not true; at any point in simulation time t , the enabling time τ of each transition may take on any value $\tau \leq t$. It is for this reason that the stochastic process is described by partial differential equations, and not merely ordinary differential equations.

Given a transition T_i with firing time Φ , and markings m_i, m_j of an SPN, we write $m_j = \text{succ}(m_i, T_i)$ to denote that the firing of T_i in marking m_i leads to marking m_j . Let $\pi_i(t)$ and $\pi_j(t)$ be the probabilities for the markings m_i and m_j at time t . The rate of probability flow from $\pi_i(t)$ to $\pi_j(t)$ is then given by $\pi_i(t) \cdot h_T(\tau)$, where τ is the length of time that T_i has been enabled at time t , and $h_T(\tau)$ represents the IRF of the firing time distribution of transition T .

We now define the time-dependent enabling time vector $\tau(t) = (\tau_1, \dots, \tau_n)$. At any time t , $\tau(t)$ represents the times for which transitions $T_1 \dots T_n$ have been enabled and n the number of currently enabled transitions plus the number of currently disabled transitions with an age memory policy and a partially expended firing time. The enabling time vector $\tau(t)$ is part of the specification of the state of the net. The overall state S of the Petri net is thus described by the vector $S = (m, \tau_1, \dots, \tau_n, t)$.

Each transition in an SPN has a memory policy. The permissible types are called AGE and ENABLE. The memory policy determines the behaviour of a transition when it becomes re-enabled after having been previously enabled and then disabled. In the case of AGE, the transition ‘‘remembers’’ the firing time consumed during the first enabling phase, and the remaining firing time is correspondingly reduced. In the case of ENABLE, the previous enabling time is forgotten, and the transition firing time starts once again from the beginning when the transition is re-enabled.

S is a random variable, and the stochastic process of the SPN is described by a system of linear, hyperbolic first-order partial differential equations for the density of S . The number of unknowns in the system is equal to the number of discrete states of the net, and the dimension of the differential equation is \mathfrak{R}^n . For a number of reasons, this system of differential equations is very difficult to solve numerically in the general case. It is for this reason that, although the method of supplementary variables has been known for a long time, to the author's knowledge, no attempt has yet been made to create a general-purpose simulation tool for SPNs using this approach.

PROXEL-BASED SIMULATION ALGORITHM

The Basic Idea

The idea behind the proxel-based simulation method is to use proxels $P = (p, S)$ to represent states S of the SPN model and their probabilities p . For any state S , we can determine each possible successor state, compute the probabilities that the model will transition into each of those states within a discrete time step Δ , and create new proxels accordingly. Δ is a discretisation parameter of the derivative in Equation (1), whose value is chosen by the user; the stochastic process itself is continuous.

Successor states are found by determining which transitions are enabled in the marking of the current proxel, and the transition probabilities are computed from the instantaneous rate functions of the enabled transitions. As the simulation progresses, proxels can be created, stored temporarily in a queue, removed, and destroyed.

In the usual case, the simulation begins with the initial proxel $P_0 = (1, m_0, \mathbf{0}, 0)$, which states that with a probability of 1, in the initial marking m_0 , all transition enabling times are 0 and the simulation time is 0. Of course, if the user model specifies several initial markings (whose probabilities must then sum to 1), then more than one initial proxel can be specified appropriately. As the simulation progresses, a logical tree of proxels is generated (see Figure 3 for an example), in which the k -th level of the tree contains all the proxels reached at the k -th discrete time step. Generation of new proxels ceases when the maximum simulation time has been reached, and the simulation terminates when the proxel queue Q is empty.

In general, during the course of the simulation, many proxels will be generated whose values of both m and t are identical, but whose values of τ differ. These proxels all represent the same discrete marking of the SPN, but are reached via different “routes” during the course of the simulation. The sum of the probabilities for such a set of proxels yields the overall probability for the discrete marking m at time t .

Specification of the Algorithm

We define a queue Q for the temporary storage of proxels. The implementation of Q is crucial for the performance of the algorithm and is discussed in the next section. We write $P.x$ to denote an element x contained in proxel P . The variable $\pi_m(t)$ is used to store the simulation result, i.e. the probability that the SPN will be in marking m at time t . The constant $tmax$ represents the maximum simulation time specified by the user. Δ represents the discrete time step; the real-valued simulation time t takes on the values $k \cdot \Delta$, where the integer-valued k denotes the number of the discrete time step. \emptyset denotes the empty set, and T a transition in the SPN. Assignments are denoted by the \leftarrow symbol.

Algorithm 1 describes the proxel-based numerical simulation of the SPN.

```

1:  $Q \leftarrow \emptyset$ 
2: addproxel(1,  $m_0$ ,  $\mathbf{0}$ , 0)
3: WHILE  $Q \neq \emptyset$ 
4:    $P \leftarrow \text{getproxel}()$ 
5:    $\pi_{P.m}(P.t) \leftarrow \pi_{P.m}(P.t) + P.p$ 
6:   IF ( $P.t < tmax$ )
7:     addproxel( $P.p * (1 - \Delta * \sum_T h_T(\tau))$ ,  $P.m$ ,
               update( $P.\tau$ ,  $P.m$ ,  $\emptyset$ ),  $P.t + \Delta$ )
8:      $\forall T$ : IF (enabled( $P.m$ ,  $T$ ))
9:       addproxel( $P.p * \Delta * h_T(\tau)$ , succ( $P.m$ ,  $T$ ),
               update( $P.\tau$ ,  $P.m$ ,  $T$ ),  $P.t + \Delta$ )

```

Algorithm 1: Proxel-Based Simulation of an SPN

The following functions are used by the algorithm:

succ(m , T)	returns the marking reached from marking m by firing transition T .
enabled(m , T)	returns TRUE if transition T is enabled in marking m .
addproxel(P)	inserts proxel P to proxel queue Q .
getproxel()	deletes a proxel from Q and returns its value.
update(τ , m , T)	updates the enabling time vector τ when transition T fires in marking m .
memory(T)	returns memory policy of transition T (i.e. ENABLE or AGE).

Explanation of the Algorithm

First, we give a line-by-line commentary of the algorithm:

- Line 1: The proxel queue Q is initialised to the empty set.
- Line 2: The initial proxel P_0 representing the initial state of the model is inserted into the queue.
- Line 3: Loop until the proxel queue is empty.
- Line 4: Get the next proxel P from the queue.
- Line 5: Add the probability of the current proxel $P.p$ to the solution.
- Line 6: Continue only if maximum simulation time $tmax$ has not yet been reached.
- Line 7: Add a new proxel representing the case that the SPN remains in the marking $P.m$.
- Line 8: Consider all transitions T that can fire in the marking of the current proxel $P.m$.
- Line 9: Add a new proxel to the queue for each of these cases.

Line 7 describes the case in which the SPN remains in the same marking as time advances from t to $t + \Delta$. The probability for this case is equal to 1 minus the probability of leaving the state during this interval, which is computed as the sum of the corresponding probabilities for each individual enabled transition firing.

The function update() modifies the j -th element of the vector τ , according to the behaviour and type of the j -th transition as follows:

```

update( $\tau$ ,  $m$ ,  $T$ ):
FOR  $j = 1$  TO  $n$  DO
CASE 1. ( $T_j = T$ ):
 $\tau_j \leftarrow 0$ 
CASE 2. ( $\text{enabled}(m, T_j) \wedge (T_j \neq T)$ ):
 $\tau_j \leftarrow \tau_j + \Delta$ 
CASE 3. ( $\neg \text{enabled}(m, T_j) \wedge (\text{memory}(T_j) = \text{ENABLE})$ ):
 $\tau_j \leftarrow 0$ 
CASE 4. ( $\neg \text{enabled}(m, T_j) \wedge (\text{memory}(T_j) = \text{AGE})$ ):
 $\tau_j \leftarrow \tau_j$ 

```

The function `update` considers all $j=1..n$ supplementary variables which are active in marking m and modifies their values as appropriate. Case 1 concerns the transition which fires, resetting its enabling time to 0. In case 2, a transition is considered which is enabled, but does not fire. Here, the enabling time is incremented by Δ . Case 3 treats disabled transitions with memory policy ENABLE, for which the value of τ is set to (or remains at) 0. Finally, in case 4, disabled transitions with memory policy AGE retain their stored enabling times.

Note that supplementary variables can be mapped to different transitions in different markings. A general-purpose SPN simulator must automatically determine the number of variables needed and determine this mapping.

The algorithm essentially uses an explicit Euler-like discretisation of Eq. (1) and then performs an exhaustive search of the thus discretised state space of the SPN. In this sense the algorithm is analogous to many other discrete search methods, such as chess-playing programs. In the form given here, the number of proxels created grows exponentially with the number of time steps $tmax/\Delta$. Obviously, this is much too expensive. In the next section, techniques for reducing the complexity are discussed.

DISCUSSION AND IMPLEMENTATION ISSUES

Reducing Complexity

The time and space complexity of the algorithm in this form is exponential in the number of discrete time steps. This is obviously far too expensive for practical use. Fortunately, heuristics for reducing the complexity are available. The simplest of these is based on the observation that during the course of the simulation, the probability of 1 for the initial state is just redistributed among the different states at each new discrete time step. Specifically, for any proxel P' that is generated from a predecessor proxel P , we have $P'.p \leq P.p$. This allows us to define a threshold value ϵ for proxel probabilities such that any proxel created with a probability value less than ϵ will be discarded, since its contribution to the overall solution is not significant. We can then replace Line 6 in Algorithm 1 by:

```
6': IF (( $P.t < tmax$ ) AND ( $P.p > \epsilon$ ))
```

This can lead to a substantial saving in the complexity of the algorithm, since the state space tree is pruned wherever the probabilities are too small to be of interest. By summing

the probability values that are discarded, the total error incurred using this approach can be computed easily.

Further reductions in complexity can be achieved by using appropriate storage schemes for Q . These are described in the next section.

Temporary Proxel Storage

Proxels that are generated are stored temporarily in a data structure Q . Different implementations of Q can significantly affect the efficiency of the algorithm.

The basic queueing strategies LIFO and FIFO yield a depth-first and breadth-first enumeration of the state space, respectively. The depth-first strategy minimises memory requirements, needing $O(b * tmax/\Delta)$ proxels, where b is the branching factor, i.e. the average number of transitions that are enabled at any time. Both of these simple strategies have exponential computational complexity, and are thus too expensive to be practical.

A better storage scheme would allow proxels $P = (p, S)$ with a given value of S to be located quickly. This would allow two proxels with identical values of S to be combined by simply adding their probability values. This in turn would significantly reduce both memory requirements and computational complexity, since it would permit substantial pruning of the proxel tree.

The fastest simulation algorithm is achieved when proxels are stored in an array that is indexed by S . This allows proxels to be accessed in $O(1)$ time. However, this requires storing an array that is large enough to be indexed by all possible values of S . This is prohibitive, since it essentially leads to the same data requirements as the PDE solver.

An appropriate compromise is probably to be achieved using a sorted dynamic tree. This would lead to a data structure whose size corresponds to the number of currently active proxels, and which allowed an access time which is only $O(\log tmax/\Delta)$.

EXAMPLE AND EXPERIMENTS

In order to illustrate the proxel-based simulation algorithm and its behaviour, we use the simple SPN of Figure 2.

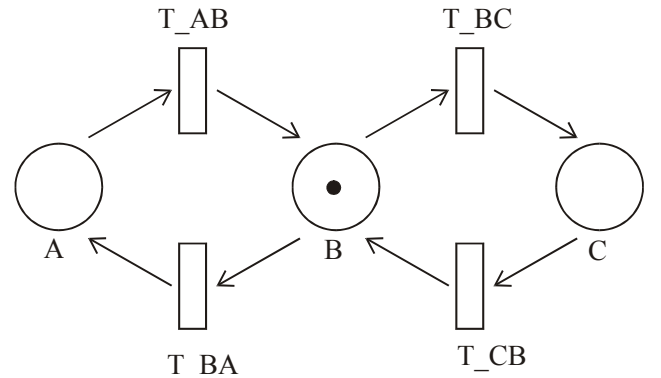


Figure 2: Example SPN

We choose the following distributions for the transition firing times:

F_{AB} = Exponential (0.5)

F_{BA} = Uniform (0.3, 0.5)

F_{BC} = Deterministic (1.0)

F_{CB} = Uniform (0.2, 0.3)

Transition T_{BC} has memory policy AGE, all others have the policy ENABLE.

Since at most two transitions can have non-zero enabling times simultaneously, we will require two enabling time variables $\tau = (\tau_1, \tau_2)$. We use τ_1 to represent the enabling time of transition T_{BA} in marking B and of transition T_{AB} in marking A. Similarly, τ_2 represents the enabling time of transition T_{BC} in markings B and A, and of transition T_{CB} in marking C.

The proxels $(p, m, \tau_1, \tau_2, t)$ generated by the simulation algorithm for the first three discrete time steps are shown in Figure 3, whereby the ? symbol signifies computed probabilities, and the times τ_1, τ_2 and t are shown as integer multipliers of Δ , i.e. a 2 represents 2Δ . From the initial proxel representing the marking B, all three successor states are reachable, depending on whether transition T_{BA} , transition T_{BC} , or no transition fires during the time interval $(0, \Delta)$.

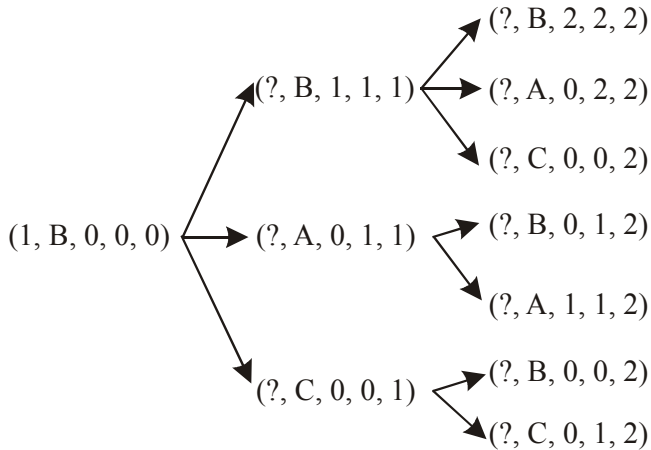


Figure 3: First Steps of the Proxel-Based Simulation

Numerical results from the proxel-based simulation are shown in Figure 4. The probabilities for each of the three system states A (red), B (green), and C (blue) are shown for the first 10 time units of operation (upper diagram); a detailed view for the first two time units is shown in Figure 4 (lower). In this simulation, all transition memory policies were of type ENABLE. A time step of $\Delta=0.005$ was used, yielding an estimated maximum error of no greater than 0.01. The solution contains both discontinuities and non-differentiable points. The solution tends towards a steady state, which has almost been reached by $t=10$. Since the proxel algorithm conserves probability, the solution values sum to 1 at all times during the simulation.

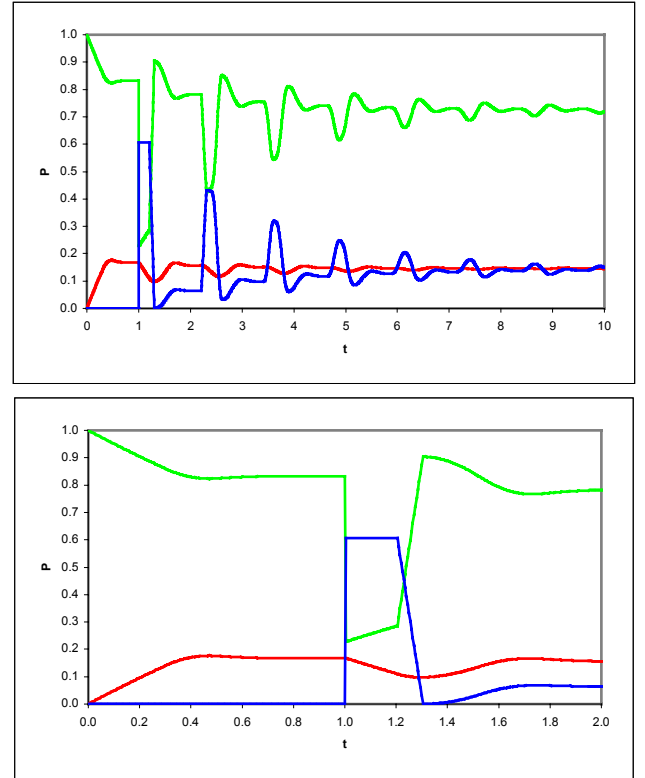


Figure 4: Simulation Results for the Example SPN

Figure 5 shows the number of variables needed by the proxel and the PDE methods to simulate the example model for maximum simulation times t_{max} from 0.5 to 15. In the case of the proxel-based method of Algorithm 1, these variables are proxels, in the case of the PDE solution they are floating-point numbers. The green curve shows the case where Q is implemented as FIFO queue, i.e. the simulation proceeds as a breadth-first enumeration of the state space, without storing the proxels in a special data structure in order to prevent duplication. The red curve shows the proxel algorithm in which Q is implemented as an array indexed by S and duplicate proxels are eliminated. The blue curve shows the results for the numerical solution of the PDE. The integration of the differential equation was a first-order characteristic method on a 500×500 spatial grid. A time step of $\Delta=0.01$ was used for both methods and a threshold value of $\epsilon=1e-8$ was used for the proxel simulation. A logarithmic scale has been used for the vertical axis. The green curve shows the exponential growth in the number of proxels needed for the basic algorithm. This is clearly unacceptable. The proxel algorithm using array storage is approximately three orders of magnitude cheaper than the PDE solution. This is due to the fact that the PDE solver accesses all variables at all time steps, whereas the proxel-based algorithm truncates the processing of proxels whose probability values drop below the threshold ϵ . The actual memory savings are a factor of four smaller, since in this implementation, one proxel requires four times as much memory as a floating-point number (32 bytes compared to 8).

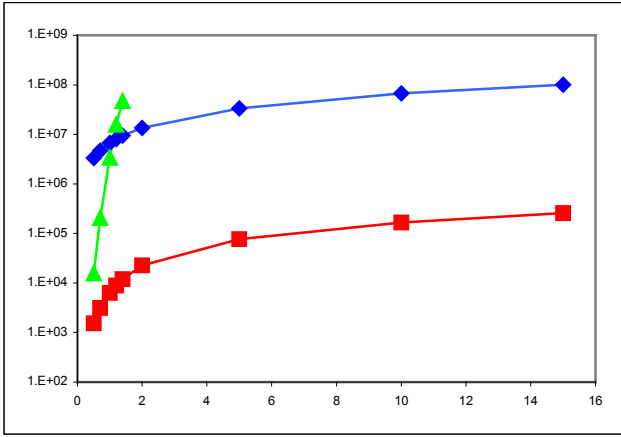


Figure 5: Computational Complexity

Figure 6 shows the solution value obtained for the probability of state B at time $t=5$ for the proxel-based simulation using different values of the discrete time step Δ . The linear convergence towards a value close to 0.7 as $\Delta \rightarrow 0$ towards is evident. This linear convergence results from the first-order discretisation of the first derivatives used by the proxel method. This allows us to use solutions obtained with two different values of Δ in order to extrapolate linearly to $\Delta=0$, obtaining an even more accurate solution. Higher order extrapolations using more than two individual solutions are also possible. The solution values for states A and C also exhibit linear convergence.

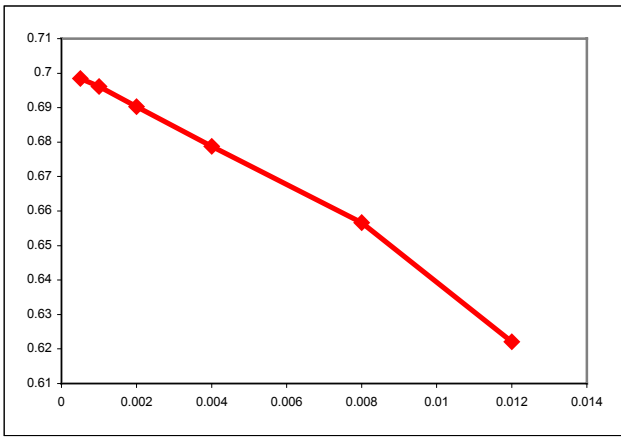


Figure 6: Convergence of Proxel Solution

This contrasts with the convergence behaviour exhibited by the stochastic discrete-event simulation. This is shown in Figure 7 for the mean and 99% confidence interval for the probability of state B for 10, 20, 40, ... 5120 replications. In this case, the stochastic nature of the simulation results in non-monotonic convergence towards the solution. The general-purpose simulator SIMPLEX 3 (Schmidt 2001) was used for this experiment.

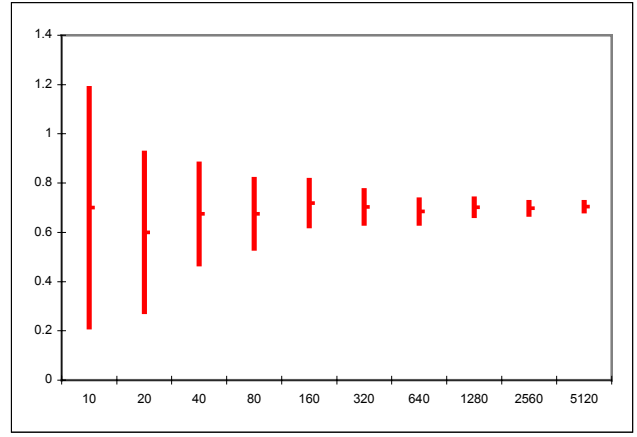


Figure 7: Convergence of Discrete-Event Solution

Figure 8 shows the relationship between computation time and solution error for the proxel-based (blue) and discrete-event (red) simulation methods on a log-log scale. The estimate of accuracy is based on a comparison with the solution obtained by linear extrapolation of the proxel solutions obtained for $\Delta=0.0005$ and $\Delta=0.001$ to $\Delta=0$ in Figure 6. The proxel method is clearly more efficient, achieving a comparable error within a computation time which is approximately one order of magnitude shorter. Furthermore, the proxel method shows monotonic behaviour, whereas the discrete-event simulation is stochastic.

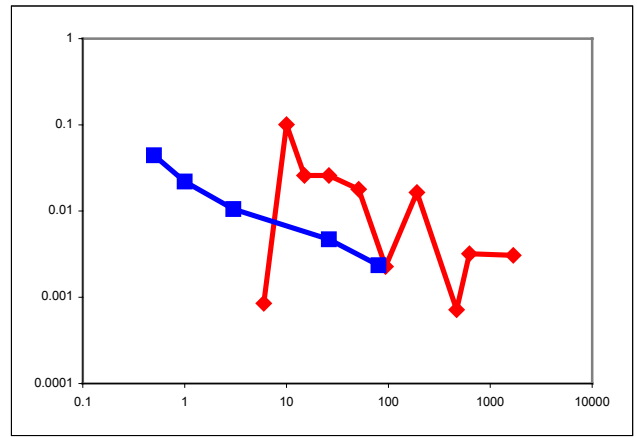


Figure 8: Error vs. Computation Time

Caution should be used, however, when drawing conclusions from this result, since SIMPLEX 3 appears to have a high computational overhead per replication.

SUMMARY AND OUTLOOK

In this paper, a new paradigm for the numerical simulation of discrete-state stochastic models such as stochastic Petri nets has been presented. The concept of a proxel as a basic unit of computation was introduced. Proxels yield a deterministic, continuous computational model, which is approximated using a discretisation of the time variable.

The model is based on the concept of tracing the movement of probability as it moves around in the state space. This computational model is similar in principle to the partial differential equation obtained by application of the method of supplementary variables. The advantages of the new approach over the differential equation are its conceptual simplicity, the opportunities it gives for reducing the number of variables computed, and the fact that it immediately suggests a straightforward simulation algorithm.

In its simplest form, the proxel-based algorithm is very expensive, and techniques for bounding the state space that is generated are needed. Some such bounding techniques, such as threshold-based pruning and storage schemes which allow searching have proven to be effective. Further research will look at using a tree-based storage scheme.

More experience with the new algorithm is needed in order to fully determine its usefulness. Situations in which it is expected to prove more efficient than a discrete-event simulation based on pseudo-random numbers include cases where the behaviour of the random number generator leads to very long simulation times, such as is the case for stiff models, or when there is a large statistical variance in the simulation results, as is found in some safety and reliability analyses.

One added advantage of the proxel computational model is that it is also capable of analysing hybrid systems containing both continuous, deterministic components as well as stochastic, discrete ones. One example of such systems are the Hybrid, or Fluid Stochastic Petri Nets (Horton et al 1998). This implies that these models can also be simulated using a proxel-based algorithm. This possibility is currently being investigated.

Further work will also include the creation of a general-purpose proxel-based SPN simulator. This will allow comparisons of accuracy and efficiency to be made for a wide range of models.

REFERENCES

Cox, D. 1955. "The analysis of non-Markov stochastic processes by the inclusion of supplementary variables." *Proc. Camb. Phil. Soc. (Math. And Phys. Sciences)* 51, 443-441.

German, R. 2000. *Performance Analysis of Communication Systems. Modeling with Non-Markovian Petri Nets*. John Wiley & Sons, Chichester.

Heller, S., S. Greiner, G. Horton. 2002. "PeNeTo: A Petri Net Simulator for Fast Safety and Quality Analysis and Cost Prediction". In *Proceedings of the European Simulation Multiconference 2002* (Darmstadt, Germany). Society for Computer Simulation. San Diego.

Horton, G., Kulkarni, V. G., Trivedi, K., Nicol, D. 1998. "Fluid Stochastic Petri Nets: Theory, Applications and

Solution Techniques." *European Journal of Operations Research* 105, January 1998, 184-201.

Molloy, M., "Performance Analysis Using Stochastic Petri Nets." *IEEE Trans. Comput.* C-31(9), Sept. 1982, 913-917.

Schmidt, B. 2001. *The Art of Modelling and Simulation. Introduction to the Simulation System SIMPLEX 3*. Society for Computer Simulation, San Diego.

AUTHOR BIOGRAPHY

Graham Horton studied Computer Science at the University of Erlangen, Germany, obtaining his Masters Degree (*Diplom*) in 1989. He obtained his PhD (*Dr.-Ing.*) in 1991 for research in parallel simulation algorithms, and his *Habilitation* in 1998 for work involving the numerical solution of Markov chains and hybrid modelling techniques. He has been Professor for Simulation and Modelling at the University of Magdeburg, Germany since 2001. His research interests include efficient simulation methods, multi-level algorithms, hybrid modelling, and creativity techniques. He can be reached via email at graham@cs.uni-magdeburg.de and in the WWW at <http://www.wisg.cs.uni-magdeburg.de/~graham>.