# An Experimental Study of the Behaviour of the Proxel-Based Simulation Algorithm

Sanja Lazarova-Molnar, Graham Horton*
Otto-von-Guericke-Universität Magdeburg

**Abstract**

The paradigm of the proxel ("probability element") was recently introduced in order to provide a new algorithmic approach to analysing discrete-state stochastic models such as are represented by stochastic Petri nets or queueing systems. Proxel-based simulation is not related to either of the standard simulation approaches: it is in no way analogous to discrete-event simulation, and, although it is based on the model's underlying stochastic process, it does not require the use of differential equations. Instead, the proxels dynamically trace the movement of probability from one state of the model to another using discretized time steps. Since the method is new, it still needs to be studied and experiments must be performed in order to fully understand its properties and behaviour. Results of such experiments will be presented and analysed in this paper.

## 1 Goals of the Paper

Proxels are a new method for the simulation of discrete-state stochastic models. The goal of this paper is to present the method, and to complement theoretical research into the proxel-based method by experiments which study its behaviour.

When a new algorithm is proposed, its usefulness must be determined, in particular in comparison with existing (and competing) methods. The properties of interest in this paper are the complexity of the algorithm (in terms of both memory and computation time) and the accuracy of the simulation results. Our experiments are designed to determine the effects of the principal algorithmic and model parameters, such as the size of the time step, the method of proxel storage, and the number of discrete states and competing activities in the model. A comparison with the standard approach, discrete-event simulation, will also be made. This initial study is limited to small, simple models.

## 2 Proxel-Based Simulation

### 2.1 The Proxel-Based Approach to Simulation

Proxels were recently introduced [Hor02] as a new technique for analysing discrete-state stochastic models such as queueing systems or stochastic Petri nets. For this class of model, the analysis is usually carried out using discrete-event Monte Carlo simulation. By contrast, proxel-based simulation is deterministic, and works with the state-space

* Fakultät für Informatik, Institut für Simulation und Graphik, D-39016 Magdeburg, Germany.
[sanja | graham]@isg.cs.uni-magdeburg.de

view of the model. Normally, this approach entails constructing and solving partial differential equations [Ger00], but Proxels yield a purely algorithmic approach to the simulation, in which differential equations are avoided completely. The goal of the new approach is to develop an easily-understood, deterministic algorithm, which, for certain classes of model, may prove to be more efficient than discrete-event simulation.

A Proxel is a basic computational unit for the simulation algorithm, which represents the probability that the simulation model has reached a specific state at a given time via a specific path. We use the term "Proxel" as an abbreviation of "probability element" by analogy to the well-known "pixel" in Computer Graphics.

The idea behind the simulation algorithm is to discretise the continuous stochastic process of the user model using a discrete time step *dt*. This yields a computational model consisting of a set of discrete states at each time point, each of which has a certain probability of occurring. The proxel simulation algorithm creates the discrete states on the fly and tracks probability as it is redistributed between these states as time progresses.

We illustrate the idea intuitively using the simple user model in Figure 1. This model consists of three discrete states DSS1, DSS2, and DSS3. Arrows indicate the state changes (activities) that are possible. The delays for each activity are described by probability distributions. We assume the model to be in state DSS1 at time $t$=0.

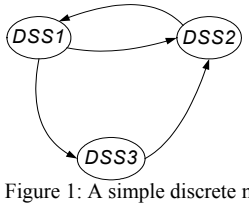


Figure 1: A simple discrete model

We now choose a discrete time step *dt*. This user-defined parameter must be chosen carefully, since it affects the accuracy of the simulation; specifically, it must be chosen so that the probability of the model making two or more state changes during any time interval ($t$, $t$+*dt*) is significantly smaller than the probability of one state change only.

Figure 2 shows the states of the model at times 0, *dt*, and 2**dt*. The state is composed of two parts - the discrete state of the user model and the length of time that an activity has been going on without a state change occurring. The latter is known as the age intensity; it is needed to compute the probabilities for each state at the new time step, as will be explained in the next section. For this simple model, only one age intensity variable is needed. For more general models, several will be necessary.

At time $t$=0, the model is in state (DSS1, 0). At time $t$=*dt*, the model can have done any of three things – transition to state DSS2 or DSS3, or remain in state DSS1. The probability for each of these occurrences can be computed from the probability distributions. States at time $t$=2**dt* and later are generated correspondingly. Proxel-based simulation is the repeated computation of the new set of proxels as simulation time progresses.

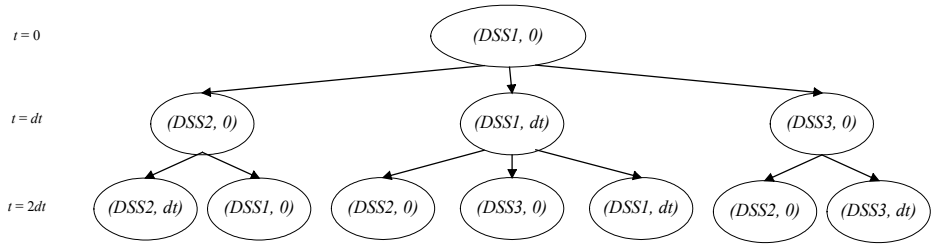


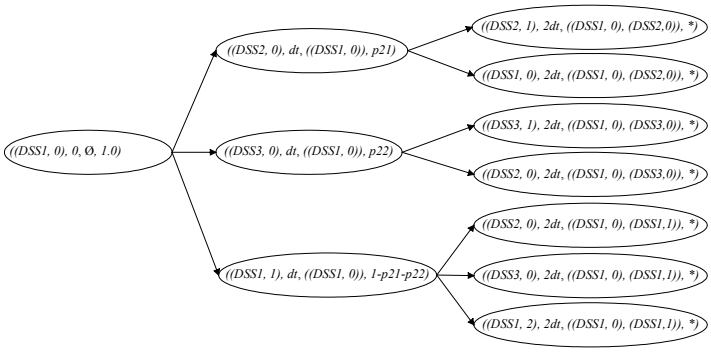

Figure 2: The first few states reached by the model

## 2.2 Definition

The proxel, as a basic unit of computation, describes a state of the model in a complete and minimal way. A proxel $P$ is defined as the following:

$$P = (S, T, R, P)$$
$$S = (\textit{Discrete State of the System}, \textit{Activation Time Vector})$$
$$\textit{Route} = (\textit{State1}, \textit{State2}, \ldots, \textit{StateN})$$

where $T$ is the global simulation time, and $P$ denotes the probability that the model is in state $S$ at time $T$, given that it has been reached through the sequence of states named as route $R$. The null sequence is denoted by $\varnothing = ()$. The *Activation Time Vector* is a vector containing the elapsed activation times of the activities in state $S$. These are needed for a complete definition of the state of the model because the scheduling of the next event is dependent on how long the concurrent activities have been active. The instantaneous rate function, denoted by $\mu(x)$ is used for this purpose. It uses as a parameter the activation time of the activity that caused the transition from the previous to the current state, and is a function of the distribution function that is associated to the state change. In the following, the term "state" will denote the vector made up of the discrete state of the system and the activation time vector.




$$p21 = 1.0 * \mu(activation\_time\_of\_the\_relevant\_activity) * dt$$

Figure 3: Beginning part of the proxel tree for the model in Figure 1.

In Figure 1, an example user model was shown for which Figure 2 shows the tree of state changes. The beginning part of the corresponding proxel tree for this model is shown in Figure 3. Note the duplication of the state (DSS2, 0) in two different proxels.

The proxel-based method distributes the initial probability of 1 for being in the initial discrete state among all of the subsequent discrete states of the model. The proxels are the computational units that store and keep track of the flow of probability from one state to another. This means that each proxel generates new proxels and this process repeats until the end of the simulation time, generating a tree structure of proxels. This structure will be referred to as the "proxel tree". The "proxel tree" is actually the state space of the model in terms of proxels.

During the process of generating the proxel tree, at each discrete time step, the same state may be generated many times, each time via a different sequence of predecessors, i.e. a different route $R_i$. In order to obtain the total probability for that state and to optimise the storage of the proxels, the probabilities of all of its instances are summed up and a new proxel that represents the state is generated. This proxel is stored as a representative of the corresponding state and its route parameter is the union of all the routes that lead to that state. This can be formally represented in the following way:

$$P = (S,\ T,\ \bigcup_{\forall i, R_i \longrightarrow S} R_i\ ,\ Pr(\text{Model in } S \text{ at time } T))$$

$$Pr(\text{Model in state } S \text{ at time } T) = \sum_{\forall i, R_i \longrightarrow S} Pr(\text{Model in } S \text{ at time } T \mid S \text{ reached via } R_i)$$

$\longrightarrow$ = *"leads to"*

Then, at each discrete time step, this routine repeats. In many cases, after a certain number of discrete time steps, a steady state is reached, which means that the sums of the probabilities of the proxels that represent the same discrete state of the system converge to a stationary value. The formal representation of the probabilities of the different discrete states at different time steps as well as the calculation of the proxels' probabilities in the next discrete time step based on the previous one is the following:

$$Pr(\text{Model in discrete state DSS at time } T) = \sum_{\forall j, S=(DSS, j)} Pr(\text{Model in state } S \text{ at time } T)$$

$$Pr(\text{Model in state } S_k \text{ at time } T+dt) = \sum_{\forall j, R_j \longrightarrow S_i} Pr(\text{Model in state } S_i \text{ at time } T) * \mu(t_{ik}) * dt$$

$t_{ik}$=*activation time of the concurrent activity that caused the transition from* $S_i$ *to* $S_k$

## 2.3 Implementation Details

Crucial for understanding the experiments that are performed is to explain the implementation details. Each proxel is stored in a record containing the probability value and state. A binary tree, which is indexed by a key calculated on basis of the state of the model, is used in order to store the proxels generated at one discrete time step. Another

tree is constructed based on the current one, which stores the probabilities for the next discrete time step. These trees are used alternately instead of storing the proxel tree for all time points. The only thing that determines the proxels in the next discrete time step are the proxels generated in the previous discrete time step. All the others can be forgotten.

Also, only one variable is needed for storing the activation time vector, because the experiments described in this paper contain no parallel activities. In other words, the dimension of the activation time vector is one. It is important to state that the method's complexity increases exponentially with the number of concurrent activities.

The models used for the experiments were all irreducible, i.e. all the discrete states are recurrent. Transient states lead to simpler structures and are cheaper to solve, which is why we do not consider them here.

The proxel-based method can be used with models containing activities with any distribution. The experiments that were performed here were based on models with activities distributed according to the exponential distribution in the interest of implementation simplicity only.

One heuristic for making the simulation method more efficient is to ignore proxels whose probability is smaller than a given threshold. These do not have a significant effect on the result, and ignoring them can substantially reduce the total number of proxels computed. In the experiments reported in Section 3, a threshold value of 1E-12 was used.

## 2.4 Alternative Approaches

The analysis of discrete-state stochastic models is usually performed using a discrete-event simulation or by solving partial differential equations.

Discrete-event simulation mimics the behaviour of the model stochastically using pseudo-random numbers. In order to get reliable results, a number of replications, each time with an independent set of random numbers, may need to be executed. Depending on the nature of the model, this number can be very large, for example if the model contains rare events. The advantages of discrete-event simulation are that it is easy to understand and that its memory requirements are very low.

On the other hand, differential equations also provide a tool for analysing discrete-state stochastic models. The approach is based on the method of supplementary variables. In contrast to the user model, the 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 disadvantages of this approach are the memory requirements and its complexity. The size of the state space grows exponentially with the number of concurrently enabled transitions. The equations to be solved are quite complex, which makes them hard to understand.

## 2.5 Example

In this section an example of the proxel-based method will be presented. A model with two states is considered and a cashier model will serve as an interpretation. He/She can be either in discrete state BUSY or discrete state FREE. The transition from FREE to BUSY is distributed according to $F_{fb}(x)$, and the return transition according to $F_{bf}(x)$, as illustrated in Figure 4. Correspondingly, the density functions are $f_{fb}(x)$ and $f_{bf}(x)$. The instantaneous rate functions $\mu$ are defined in the following way:

$$\mu_{fb}(x) = \frac{f_{fb}(x)}{1 - F_{fb}(x)} \qquad \mu_{bf}(x) = \frac{f_{bf}(x)}{1 - F_{bf}(x)}$$

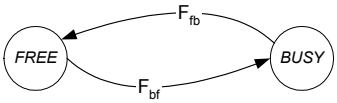



Figure 4: Example model

The corresponding proxel tree is shown in Figure 5. If $F_{fb}(x)$, $F_{bf}(x) \sim$ Exponential($\lambda$), then $\mu_{fb}(x) = \mu_{bf}(x) = \lambda$, i.e. the instantaneous rate function is constant. In this case, the probabilities in each proxel that represents a discrete state different than the predecessor would be calculated as a product of the parameter $\lambda$ and the probability of the predecessor proxel. Following that, $p_1$ in Figure 5 is equal to $\lambda$.

Furthermore, all the proxels at any discrete time step are constructed based on the ones in the previous discrete time step, using the instantaneous rate functions, in the same way as the first couple of steps shown in this example. Additionally, all the proxels at one step that represent the same state of the system are summed up and a new proxel is introduced, whose route parameter is the union of all routes that lead to the concrete state.

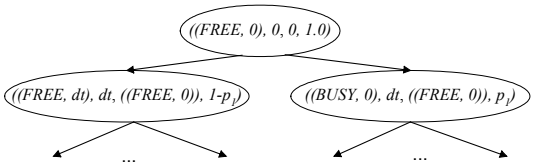



Figure 5: Beginning part of the proxel tree for the model in figure 4

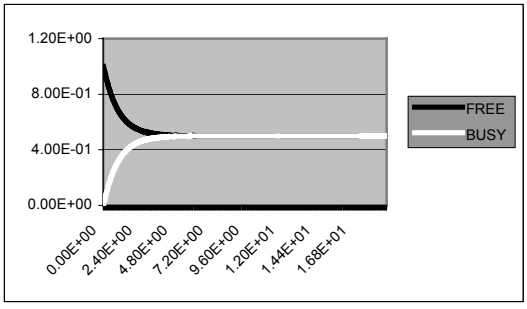



Figure 6: Results of the proxel-based method for the model in figure 4

The sums of probabilities of the proxels at each time step that represent the same discrete state are shown in Figure 6. These are the results of the proxel-based simulation for the model presented in Figure 4 using $\lambda = 0.5$ and $dt = 0.05$.

# 3 Computational Experiments

In this section, we report the results obtained from computational experiments using the proxel-based simulation method. Our goal is to study the behaviour of the method with regard to the following parameters:

- the size of the time step and the overall number of discrete time steps,
- the number of discrete states and competing activities in the model,
- the different strategies for storing and accessing the proxels in memory.

These factors affect both the memory and the computational complexity of the algorithm.

We limit ourselves here to small, simple models, since we are interested in studying basic behaviours only. The models we use are not representative of those which occur in practice. Exponential distributions are used in the interest of implementation simplicity only. This is not a restriction on the proxel-based method, and experiments with general distributions show the same or similar behaviour. The models used here contain no parallelism, i.e. there are no concurrent activities. This leads to very small computational models, which can be solved efficiently. In the general case, this limitation will, of course, not hold, and the numbers of proxels to be computed will be much larger.

## 3.1 Discretisation Step Size and Number of Time Steps

The computational complexity of the algorithm grows with the overall number of time steps, which is the maximum simulation time divided by the length of the discrete time step *dt*. In addition, the discrete time step affects the accuracy of the simulation result.

Figure 7 illustrates the effect of *dt* on the accuracy of the simulation result. The figure shows an arbitrarily chosen solution value (the probability that the model of Figure 4 will be in discrete state "FREE" at time $t$=2.4) as a function of *dt*. The linear convergence as $dt \rightarrow 0$ can be clearly seen. We conclude that the simulation method is first-order accurate. This property allows us to obtain accurate solution values by extrapolation of two or more simulation results to $dt$=0.

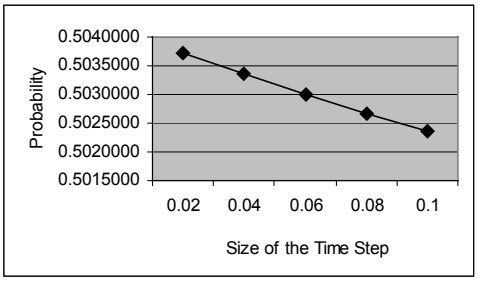



Figure 7: Effect of time step on accuracy

Figure 8 shows the number of proxels used by the algorithm as a function of the length of the simulation for the model of Figure 4 using a value of $dt$=0.05. During the early stages of the simulation, the complexity grows quadratically. Ultimately, however, the complexity grows only linearly with the number of time steps. This behaviour can be easily explained. During the first part of the simulation, the number of proxels at each level of the tree of Figure 5 grows linearly, so that the sum of all proxels over all time steps grows quadratically. Later in the simulation, the number of proxels at each time step remains constant, leading to a linear complexity. This property generalises to any model.

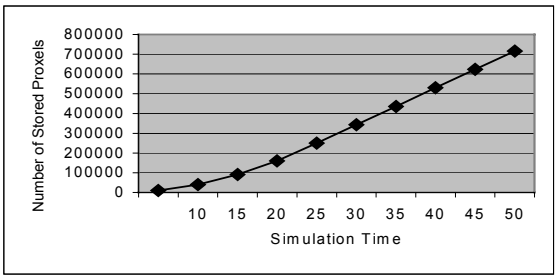



Figure 8: Complexity as a function of length of simulation

## 3.2 Influence of the Number of Discrete States

Next, we consider the effect of the number of discrete model states on the number of proxels generated by the algorithm. The first experiment uses the model shown in Figure 9, which consists of a ring of states of size $n$. This model contains $n$ activities, which is minimal while still retaining irreducibility.

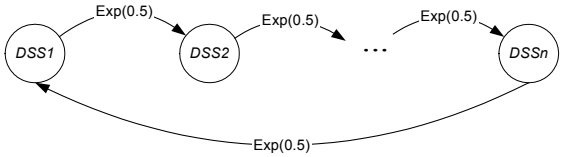



Figure 9: Example model

This model was simulated up to $t$=50 using a value of $dt$=0.05 for various values of $n$. The number of proxels used by the algorithm is shown in Figure 10. Clearly, the dependency is linear, as is perhaps to be expected.

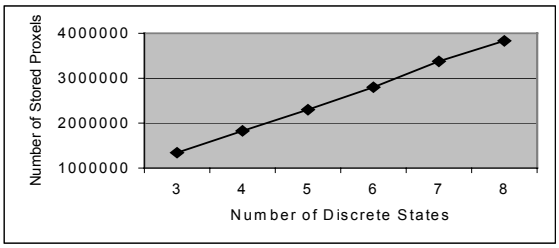



Figure 10: Relationship between numbers of proxels and discrete states

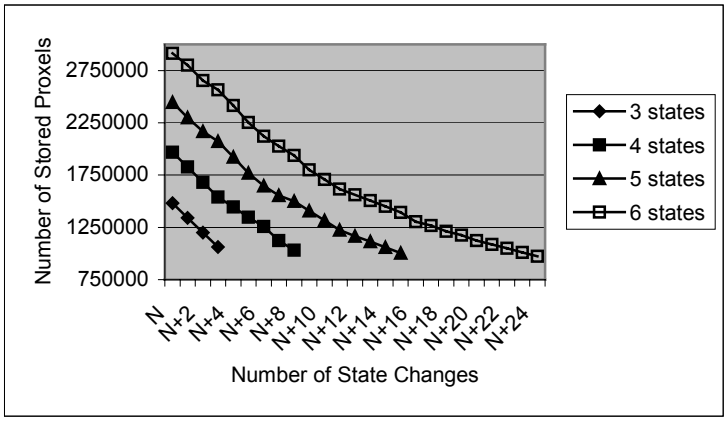



Figure 11: Dependence of the number of stored proxels on the number of activities for models with all recurrent states and different number of states

We now consider increasing the number of activities. The results of the corresponding experiment are shown in Figure 11. In this case, the number of proxels decreases as the number of activities per state goes up. For the maximal case of $n(n\text{-}1)$ activities per state, the number of proxels needed actually went down as the number of states went up. We initially found this behaviour to be counterintuitive.

The behaviour can be explained by the fact that the increasing number of activities increases the overall rate at which probability leaves each state. Our algorithm ignores proxels whose probability falls below a given threshold, and a higher outflow rate means that this threshold will be reached sooner, causing the generation of new proxels to cease.

This experiment also illustrates one advantage of the proxel method over the PDE-based approach to simulation. Since the proxel algorithm generates states on the fly, it can take advantage of effects such as this, leading to a significant saving in the total number of variables to be processed.

## 3.3 Influence of Proxel Storage Strategy

The fastest method of accessing proxels can be achieved by using an array indexed by the proxel state $S$. The time needed to access a proxel would then be a constant $O(1)$, which is optimal. However, this array of proxels may only be very sparsely used, leading to a considerable waste of memory space. This led us to use a binary tree, in which each proxel is stored as a node. The comparison between the proxels in order to find their correct location in the tree is based on a unique key for each proxel. The value of the key is obtained as a function of the proxel state $S$. In this way, memory usage is optimised, since memory is never allocated but not used. However, the price for this is longer access times of $O(\log d)$, where $d$ is the depth of the binary tree. Experiments showed that speed was a smaller problem than memory.

The binary tree that was used initially was created without any active balancing, and so the trees that were created were very deep, i.e. unbalanced, leading to longer access times. In order to decrease the search time, an AVL tree with rebalancing was used. Although this achieved optimal tree balance, it also made the program run more slowly owing to the repeated overhead involved in re-balancing the tree. Since the tree is actually used only once per discrete time step, rebalancing was very expensive.

The computation time for the simulation with and without balancing is presented in Figure 12. Clearly, using a tree without any balancing proved to be the more efficient strategy. We suspect that a compromise between the unbalanced tree and the AVL tree would yield better results, but this is still a subject of further research. Compromise means either rebalancing only after every $n>1$ discrete time steps, or relaxation of the condition on the AVL trees to allow the height differences between subtrees to be greater than 1. In this manner, we might be able to obtain approximately balanced trees at a cost which is lower than the gains through faster access times. Alternatively, we could attempt to develop a better hash function for the key, which will automatically generate a more balanced tree. This is also something that will be examined in future research.

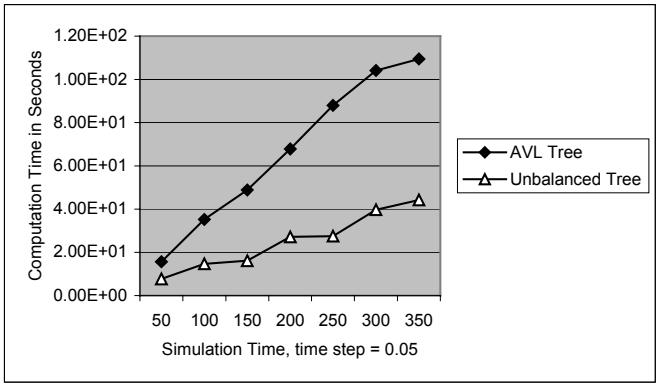



Figure 12: Comparison of the computation time when using an AVL and unbalanced tree

## 3.4 Comparison of Proxel- and Discrete-Event Simulation

It is very difficult to talk about comparison of the results obtained with discrete-event simulation and the ones from the proxel-based simulation because the quality of the results is different. Discrete-event simulation relies on random numbers and because of that the results will always be stochastic in nature. On the other hand, the proxel-based simulation is completely free of random behaviour. Therefore the quality of the results depends only on the size of the time step, making it more analogous to the discretised PDE approach.

The comparison also depends on the model being analysed. If it contains a very rare event, then it would take a large number of replications in the discrete-event simulation for that event to occur. On the other hand, the proxel-based simulation compute the rare events along with all the others.

In Figure 13, a model with two rare events is presented. There are four exponentially distributed activities with rate parameter 1, and two with rate parameter 0.0001. The model is in state DSS1 or DSS2 most of the time, and very rarely changes to the discrete state DSS3, from where it also most often transits back to DSS2, and only very rarely to DSS4.

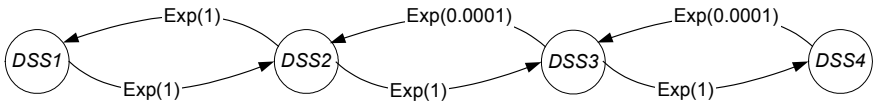



Figure 13: Example model for rare events

The model in Figure 13 was analysed using the discrete-event simulation package SIMPLEX3. After 5000 independent replications, each up to time $t$=10000, which took about 25 minutes, the simulator did not once visit the discrete state DSS4.

The same model was analysed using proxel-based simulation, using a time step of $dt$=0.05. The result for DSS4 is presented in Figure 14. It can be noticed that somewhere around $t$=7.8 the probability has converged to a value around 4.5E-9. The total running time of the algorithm, up to simulation time $t$=50, was 4.2 seconds.

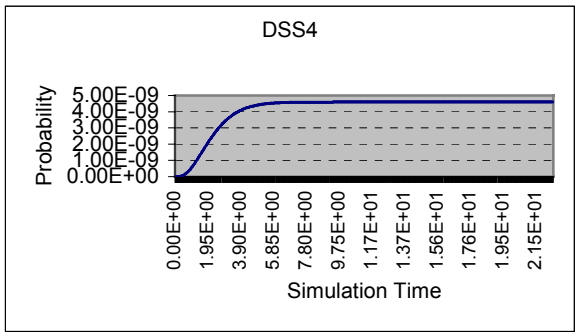



Figure 14: Results from proxel-based simulation for discrete state DSS4

This result makes us optimistic that proxel-based simulation could prove to be particularly useful for analysing small, stiff models for which a high level of accuracy is required, such as occur in safety and reliability modelling.

# 4 Summary

The proxel based method is alternative method for analysing the behaviour of discrete stochastic models. It follows the behaviour of the model considering every possible development, calculating the probabilities of the state changes based on the distribution functions that describe them.

The method is applied under the assumption that all the state changes happen within discrete time intervals *dt* and that the number of state changes that happen in time less than *dt* is negligible.

In this paper, the behaviour of the proxel based simulation under different circumstances was observed. It was discovered that the accuracy of the method has a linear dependence on the size of the time step and that the computational complexity is also linear in the number of discrete time steps. There is an inverse relationship between the complexity and the number of competing activities per state. Another conclusion from the experiments was that it was cheaper to work with an unbalanced tree than to perform active balancing using an AVL tree. The method performs well for models with rare events, for which Monte Carlo simulations can need a very large number of replications.

The results from these experiments can contribute to the further optimisation of the method in terms of memory complexity and computation time. The experiments were necessary with respect to the novelty of the method to give more insight into its use of memory and computation time. However, only a few aspects of the behaviour of the proxel-based simulation were examined. This means that there are still many cases to be examined. This will serve as a starting point for further testing of the method with respect to different types of models.

# References

[Hor02] G. Horton: A new paradigm for the numerical simulation of stochastic Petri nets with general firing times. Proceedings of the European Simulation Symposium 2002, Dresden. Society for Computer Simulation, 2002.

[Ger00] R. German: Performance Analysis of Communication Systems. Modeling with Non-Markovian Stochastic Petri Nets, John Wiley & Sons, Ltd, 2000.