Available online at www.sciencedirect.com
ScienceDirect
ELSEVIER
Electronic Notes in Theoretical Computer Science 175 (2007) 73–86
Electronic Notes in Theoretical Computer Science
www.elsevier.com/locate/entcs

# Simulation of Generalised Semi-Markov Processes based on Graph Transformation Systems

Piotr Kosiuczenko, Georgios Lajios[1]

*Department of Computer Science*
*University of Leicester, UK*
{pk82, gl51}@mcs.le.ac.uk

Abstract

Stochastic Graph Transformation combines graphical modelling of various software artefacts with stochastic analysis techniques. Existing approaches are restricted to processes with exponential time distribution. Such processes are sufficient for modelling a significant class of stochastic systems, however there are interesting systems which cannot be specified appropriately in such a framework. In several cases one needs to consider non-exponential time distributions. This paper proposes a stochastic model based on graph transformation with general probability distributions. This model is well suited to represent concurrency and performance aspects of architecture reconfiguration. It is also possible to apply Monte Carlo simulation techniques in order to analyse behaviour of complex stochastic systems. The new model is implemented and used to simulate simple networks.

*Keywords:* Stochastic modelling, graph transformation, simulation

## 1 Introduction

The specification of distributed systems, telecommunication systems, multimedia applications or computer networks must take into account not only functional properties but also real-time and performance aspects. To analyse such properties, stochastic methods are required.

Stochastic models like Generalised Semi-Markov Processes (cf. e.g. [13]) have a long history of application, but they do not provide primitives for modelling of concurrency aspects. They also lack mechanisms for compositional specification. Thus models of larger systems tend to be very complex.

[1] Corresponding author. This research was partially funded by European Community's Human Potential Programme under contract HPRN-CT-2002-00275, [SegraVis].


doi:10.1016/j.entcs.2007.04.018

There exist several formalisms for the analysis of performance and concurrency aspects. We will discuss briefly the most prominent of them. One of the first models used for this purpose were Stochastic Petri nets [8]. Generalized Stochastic Petri Nets found a wide acceptance (cf. [1]). Those nets are defined as usual Petri nets, with the addition of random assignment of a firing delay to each transition. There is a race condition between all enabled transitions. In the case of exponential probability distributions, this model corresponds to Continuous Time Markov Chains (CTMC) [13]. Stochastic Petri nets proved to be very expressive. They are well suited for specification of concurrency aspects, but the resulting model is rather low level.

Process algebras provide compositional facilities for modelling of concurrent systems. Stochastic process algebras are a natural extension of process algebras (cf. e.g. [12] and part two of [2]). They are used for performance modelling. As in the case of Stochastic Petri nets, a non-negative real number is randomly associated to an action. That number determines the delay of the corresponding action. While basic approaches rely on continuous-time Markov chains, there are also extensions to general distributions, based on Generalised Semi-Markov Processes and Stochastic Automata [2].

Nevertheless, architectural aspects of distributed systems, computer networks and mobile applications can be hardly specified with those formalisms. Especially in the case of high degree of architectural reconfiguration a high level formalism is needed with facilities for modelling architectural artefacts. This gave rise to the notion of Stochastic Graph Transformation, which combines the benefits of using graph transformation for system modelling with the power of stochastic analysis [11] (see also [18]). Those approaches enrich graph grammars by associating an exponential time distribution to each transformation rule. The distribution models the random delay of rule application. This model is a special case of a CTMC. There exist powerful model checking tools such as PRISM [14] which can be used for analysing properties of CTMCs. However, there are several stochastic phenomena, which cannot be modelled using exponential distribution. For example, file sizes and document transmission times over HTTP/IP and timeouts in communication protocols cannot be appropriately modelled with exponential distributions. Further, one would often like to include results of measurement into the modelling. There are standard techniques for extracting normal distributions from a random sample. Sometimes, only the minimum and maximum value of a quantity are known, thus modelling could be done by assuming a uniform distribution. Those cases can only be modelled using a wider class of distributions and a more general model.

In this paper we propose *Generalised Stochastic Graph Transformation Systems* to model and analyse architectural evolution with non-exponential time distributions. States of concurrent systems are modelled by graphs. Transitions of those systems are modelled by graph transformations. To model delays, we associate arbitrary continuous probability distributions with graph transformation rules. Graph transformation executions have delays, which adhere to those distributions. The model works as follows: a system state is modelled by a graph and the delay of rule

application is measured by a separate timer. Different rules may be applicable, but only the rule with the smallest delay can be executed. If a rule is executed, then a new state is reached, timers corresponding to enabled rules are decreased, timers corresponding to disabled rules are removed, and new timers are set for rules which become enabled. Let us observe that graph transformation rules can be in conflict, and an application of one rule may disable application of another one.

This model is well suited for modelling of concurrency aspects, architectural aspects as well as stochastic aspects. It uses timed events to model the concurrent execution of events, thereby giving a direct representation of the intuitive idea that each event has its own timer and will be applied when its time expired, independent of other events. Our approach can be seen in the line of research combining high-level modelling techniques with stochastic analysis.

Stochastic modelling is only useful if there are analysis techniques to investigate the properties of the systems which are modelled. The more general a modelling approach is, the more difficult is its analysis. Stochastic model checkers like PRISM [14] are very powerful tools for Markov Chains, but fail when more general stochastic processes are involved. Anyway, their power could not be fully exploited for stochastic graph transformation, because the complete state space of the model has to be generated first, before model checking tools can be used. This procedure emerged as a serious bottleneck, because the isomorphism checking involved is very complex. Even systems consisting of a small number of nodes and edges can lead to an enormous amount of states, a phenomenon known as *state space explosion*. When switching to arbitrary distributions, model checking itself becomes more complex [16].

We propose using Monte Carlo simulation techniques for testing stochastic graph transformation systems with arbitrary distributions. Monte Carlo Methods are stochastic simulation methods based on pseudo-random numbers. These methods allow us to make predictions about system's behaviour. The simulated system is traversed on randomly chosen paths; those paths simulate real-time behaviour. After a sufficient number of such paths is traversed, knowledge on the probabilistic behaviour the system is gathered, with a certain confidence interval which can be narrowed by further runs. Simulation is thus a very well scalable technique. First experiments with simulation have been promising, and we are currently developing a tool for the analysis of stochastic graph transformation systems based on these techniques.

The paper is organized as follows: The next section explains why it is necessary to use general distributions. Section 3 presents the underlying notion of Generalised Semi-Markov Processes. The new model called (Generalized) Stochastic Graph Transformation System and the corresponding stochastic process are defined in Section 4. Section 5 explains how Monte Carlo simulation techniques can be used in the case of stochastic graph transformation systems and presents some experimental results. Section 6 concludes this paper.

## 2 Integrating Graph Transformation and general distributions

Combining graph transformation systems with stochastic models which have non-exponentially distributed application delays is not as trivial as one might expect. Graph transformation is intrinsically concurrent; in general, more than one rule can be applied to a graph. Non-deterministic choice between the alternatives leads to the problem of sequential and parallel independence [19]. Stochastic graph transformation aims at making this choice stochastic. Rule applications have stochastic delays, and there is a *race* between all applicable rule matches which the fastest contestant wins. Formally, this means that the stochastic application delay of the rules is computed for each match by drawing a random sample according to a stochastic specification, and the rule match with the smallest delay is applied first. For instance, think of a graph grammar modelling a network where rules *connect* and *disconnect* can be applied to a certain state. For each rule, a probability distribution specifies the behaviour of these actions. Say, *connect* is exponentially distributed with expected value 1 sec, and disconnect is normally distributed with mean 25 sec and standard deviation 25 sec (truncated at 0). Then, in most cases *connect* will win the race, leaving only probability 1% for *disconnect*. When all actions are exponentially distributed, evaluating the race condition is easy because of the memoryless property. This means that the time which had already elapsed for the "loser" action need not be taken into account for the next race, as the probability that an exponentially distributed random variable is greater than $t + s$ conditional on being greater than $t$ is the same as being greater than $s$. Therefore, in a continuous-time Markov chain, which is the structure obtained in the pure exponential case, a transition can be performed and one can forget about everything which happened before.

In the case of general distributions, this simplification is not possible. In the example above, if rule *disconnect* has lost a couple of races, say adding up to 20 sec, then this waiting time has to be considered, making an application more probable – about 7% in one step, yielding more than 50% for ten consecutive steps. It is this semantics which a modeller presumably intends when assigning probabilistic waiting times to rules.

Proper assignment of waiting times in the context of graph transformation means that matches have to be traced through transformation sequences. The waiting time of a rule application has to be considered for every match of the rule as long as the match is present. So if a transformation step changes elements of the graph which are not in conflict with the match, i.e. if the productions are independent, the old match is still present in the new graph, and the value of the timer measuring waiting time has to be decreased. We will address this issue by using unfolding grammars, which allows unique identification of elements [3].

One solution avoiding waiting times is to approximate general distributions by introducing virtual states and combining exponential distributed transitions in such a way that the desired distribution results. This is known as Cox's method of phase-

type distributions [6]. A major drawback of this approach is that it expands the state space. Also, many interesting distributions can only be approximated with a very high number of virtual states. The resulting models are not intuitive, as there is no direct correspondence between the states of the stochastic model and the states of the system. We therefore propose a stochastic model which considers waiting times – Generalised Semi-Markov Processes.

## 3 Generalised Semi-Markov Processes

Generalising the notion of CTMCs to arbitrary distributions, there are two options: Semi-Markov Chains [16] or Generalised Semi-Markov Processes. As discussed above, semantic models of interleaving processes need to consider waiting times, which are not supported by Semi-Markov Chains. We therefore vote for the second alternative and adopt the following definition from [2].

**Definition 3.1** A generalised semi-Markov scheme (GSMS) is a structure $(Z, E, active, next, F)$ where

- $Z$ is the set of states;
- $E$ is a set of events;
- $active : Z \to \mathcal{P}(E)$ assigns a finite set of active events to each state;
- $next : Z \times E \to Z$ is a partial function that assigns the next state according to the current state and the event that is triggered. We assume that $next(z, e)$ is always defined for $z \in Z$ and $e \in active(z)$;
- $F : E \to (\mathbb{R} \to [0, 1])$ assigns to each event a continuous distribution func- tion such that $F(e)(0) = 0$; we write $F_e$ instead of $F(e)$.

As initial condition a state $z_0 \in Z$ is appointed. A generalised semi-Markov process (GSMP) is the stochastic process defined by a GSMS.

The behaviour of a GSMP can be described as follows. In each state $z$, all active events $e \in active(z)$ are assigned a real number $\rho(e)$, the remaining time to execute the event. The next step is determined by the active event $e^*$ with smallest number $\rho(e)$. One can think of race between the competing events which is won by the fastest event. Note that the probability that two events have the same time is 0 due to the fact that the distribution function is continuous. Once the event is chosen, the next state is given deterministically by $next(z, e^*)$. The set $New(z, e^*)$ of newly activated events is defined as

$$New(z, e^*) = active(next(z, e^*)) \setminus \Big(active(z) \backslash \{e^*\}\Big),$$

i.e. the events which became active in the new state and have not been active before. The set $Old(z, e^*)$ is given by all active events that have been active in the old state (without $e^*$):

$$Old(z, e^*) = active(z) \cap \Big(active(next(z, e^*)) \setminus \{e^*\}\Big).$$

Now, the value of $\rho$ is determined randomly for all newly activated events $e$ according to their distribution $F_e$, and the value of all old events is decreased by $\rho(e^*)$. Thus the updated function $\rho'$ is given by

$$\rho'(e) = \begin{cases} Random(F_e), & \text{if} \quad e \in New(z, e^*) \\ \rho(e) - \rho(e^*), & \text{if} \quad e \in Old(z, e^*) \end{cases}$$

where $Random(F_e)$ denotes a random number determined by drawing a sample according to distribution $F_e$.

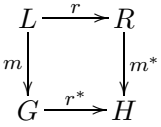
The operational semantics of this model is defined by mapping a GSMP to a Stochastic Automaton [2]. Despite their more complex semantics, GSMPs are a direct generalisation of continuous-time Markov chains (CTMC). In fact, a GSMP in which all events are associated an exponential distribution is a CTMC. Because of the memoryless property of the exponential distribution, it is not necessary to consider how long an event has already been active, as the conditional probability $\mathbf{P}(X > s + t \mid X > t)$ equals $\mathbf{P}(X > s)$. GSMPs provide an excellent basis for modelling state-based systems with arbitrary distributions.

## 4 (Generalised) Stochastic Graph Transformation Systems

In this paper we use the single-pushout approach for graph transformation [19]. To make the new model more general, we define graphs in a categoric way instead of set theoretic one. Typed graph transformation is a technique of key relevance in the modelling of visual languages and in model transformation. The type graph can be defined in several ways and used for various purposes; in particular it can be defined as a colimit obtained in a graph unfolding process [3]. Types can be understood in particular as object IDs.

**Definition 4.1** A directed graph is a quadruple $G = \langle G_V, G_E, src_G, tar_G \rangle$ with a set of vertices $G_V$, a set of edges $G_E$, and functions $src_G : G_E \to G_V$ and $tar_G : G_E \to G_V$ associating to each edge its source and target vertex. A graph morphism $f : G \to H$ is a pair of functions $\langle f_V : G_V \to H_V, f_E : G_E \to H_E \rangle$ preserving source and target, i.e., such that $f_V \circ src_G = src_H \circ f_E$ and $f_V \circ tar_G = tar_H \circ f_E$. A partial graph morphism $f : G \to H$ is a graph morphism which is defined on a subgraph $dom(f)$ of $G$. A typed graph $t$ over a (fixed) type graph $TG$ is a graph morphism $t : G \to TG$ which assigns types to nodes and edges [7]. A morphism of typed graphs is a graph morphism compatible with the typing.

A rule $p : L \xrightarrow{r} R$ consists of a rule name $p$ and an injective partial graph morphism $r$. A match for $r : L \to R$ into some graph $G$ is a total injective morphism $m : L \to G$. Given a rule $p$ and a match $m$ for $p$ in a graph $G$, the SPO-transformation from $G$ with $p$ at $m$ is the pushout of $r$ and $m$ in the category of graphs and partial graph morphisms.




A graph transformation system $\langle TG, P, \pi, G_0 \rangle$ consists of a type graph $TG$, a set $P$ of rule names and a function $\pi$ mapping each rule name to a $TG$-typed rule $\pi(p) : L_p \to R_p$. We make use of negative application conditions (NACs) [9], shown as crossed out nodes and edges in the figures. Images of crossed out elements must not be present in the instance graph, otherwise the rule is not applicable. Every crossing line represents one NAC, and all have to be satisfied in order to apply a rule. Formally, a NAC is given by an injective morphism $n : L \to N$ which maps the left hand side of a rule to a pattern $N$, and a rule is applicable iff its match $m : L \to G$ does not factor through $n$, i.e. there is no injective (total) morphism $f : N \to G$ such that $f \circ n = m$.

A generalised [2] stochastic graph transformation system associates with each rule name a distribution function governing the delay of its application.

**Definition 4.2** [stochastic GTS] A (generalised) stochastic graph transformation system $\mathcal{SG} = \langle TG, P, \pi, G_0, F \rangle$ consists of a graph transformation system $\mathcal{G} = \langle TG, P, \pi, G_0 \rangle$ and a function $F : P \to (\mathbb{R} \to [0, 1])$ associating with every rule name a continuous distribution function $F_p$ with $F(e)(0) = 0$.

**Example 4.3** Figure 1 shows the rules of a graph transformation system modelling a peer-to-peer network. New peers enter the network (rule *new*), establish a connection (rule *connect*), and eventually disappear (rule *kill*). The empty graph serves as start graph. We assume that there is no dangling edge condition preventing connected peers from being killed.

In order to obtain a stochastic graph transformation system, we have to associate continuous time distributions with the rules. We assume that the arrival rate of new peers is exponentially distributed with rate $\lambda$. This is a standard assumption in queuing theory. Let the application delay of rule *connect* be governed by a normal distribution with mean $\mu$ and standard deviation $\sigma$. So we assume that most peers tend to be connected approximately the same time, which is chosen to be the mean of the normal distribution. Finally, the lifetime of a peer (which corresponds to the application delay of rule *kill*) is assumed to have Erlang distribution with shape $k$ and rate $\kappa$. The Erlang distribution specialises to the exponential distribution for $k = 1$, but allows a greater flexibility for defining probability densities with peaks later than time 0. It is therefore often applied for waiting times.

[2] We call this kind of system *generalised* as it uses general distributions in contrast to the approach in [11]. In the rest of the paper, we will omit this word.

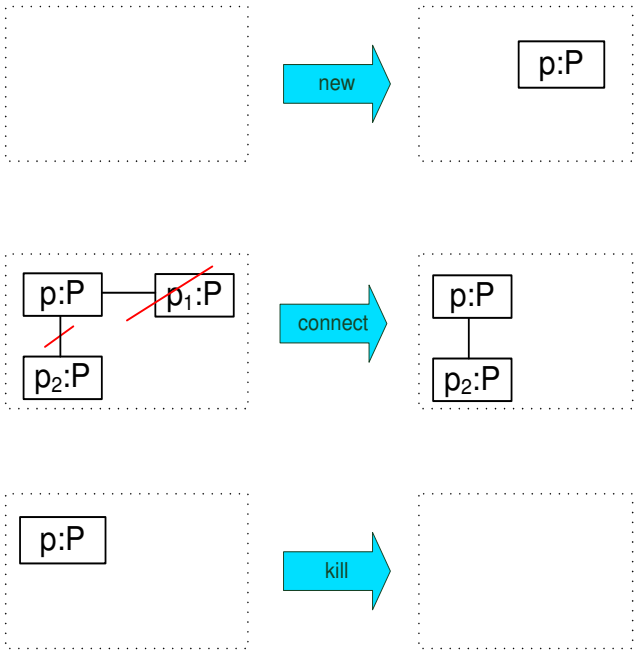



Figure 1. Peer-to-peer network: Basic rules

| rule name | distribution | parameters |
|---|---|---|
| new | $1 - e^{-\lambda x}$ | $\lambda = 0.5$ |
| connect | $\frac{1}{\int_0^\infty \exp(-(t-\mu)^2/(2\sigma^2))dt} \int_0^x \exp\left(-\frac{(t-\mu)^2}{2\sigma^2}\right)dt$ | $\mu = 1, \sigma = 1$ |
| kill | $\frac{1}{(k-1)!}\gamma(k, \kappa x)$ | $k = 2, \kappa = 0.5$ |

The intuitive idea of the semantics of a Stochastic Graph Transformation System can be explained as follows. In the start state, all matches for all rules are determined and stored as pairs $< rule, match >$. We call such a pair a *rule match*. For each rule match the application time is set to a random number corresponding to the distribution the rule obeys. Then, the rule match with the smallest time is chosen, applied, and the remaining rule matches are checked. If their match is no more applicable, they are removed. If it is still applicable to the new graph (we will soon define this thoroughly), its time is reduced by the time that already elapsed. All new rule matches are computed and assigned a random application time. Then again, the fastest of them is chosen for application. So the waiting time of the events which lost the last race is considered.

We formally define the semantics of a (Generalised) Stochastic Graph Transformation System by mapping it to GSMP. The rough idea is to define the set of states as all reachable graphs of the graph transformation system. An active event

in a state is a rule match. Thus, the newly activated events in a sequence of states are those rule matches which did not exist in the previous state. We therefore have to compare matches to different reachable graphs, which can be done by introducing a global name space. For this purpose, the concept of *unfolding grammars* [3] comes handy, as it allows to derive from an arbitrary graph grammar a safe (i.e. injectively typed) grammar providing a compact representation of the transition system. The unfolding type graph $TG'$ can be seen as a global name space, and the rules of the unfolding represent rule matches of the original grammar (typed over the global name space). By using the unfolding, we can directly compare matches into different graphs, just by calculating their intersection in $TG'$.

Let $\mathcal{SG} = \langle \mathcal{G}, F \rangle$ be a Generalised Stochastic Graph Transformation System, where $\mathcal{G} = \langle TG, P, \pi, G_0 \rangle$ is a typed graph grammar, and let $\mathcal{UG} = \langle TG', P', \pi', G'_0 \rangle$ be the unfolding grammar associated with $\mathcal{G}$ (ignoring $F$). The construction of the unfolding is explained in detail in [3]. Roughly, the type graph $TG'$ is a colimit of the whole transition system which serves as a global name space, and a rule name $p' \in P'$ represents a rule match $\langle p, m \rangle$ of the underlying grammar $\mathcal{G}$, where $m$ is an embedding into $TG'$.

The unfolding is mapped over the original grammar by the so-called folding morphism $\chi = \langle \chi_T, \chi_P \rangle : \mathcal{UG} \to \mathcal{G}$. The first component $\chi_T : TG' \to TG$ is a graph morphism mapping each graph item in the type graph of the unfolding to the corresponding item in the type graph of the original grammar $\mathcal{G}$. The second component $\chi_P : P' \to P$ maps any production occurrence $\langle p, m \rangle$ in the unfolding to the corresponding production $p$ of $\mathcal{G}$.

We are now ready to define the GSMP associated with a Stochastic Graph Transformation System. The state space $Z$ consists of all graphs reachable from the start graph by applying the rules from the unfolding grammar. The set $E$ of events is defined as $E := P'$. Rule matches which coincide on the global name space are thus identified. The set *active* of active events in a state $G \in Z$ consists of all rules applicable to $G$. Given such a match, the result of function $next(G, p')$ is defined to be the unique graph resulting from applying rule $p'$ to $G$ in the unfolding grammar. We assume a concrete deterministic definition of the pushout.

The definition of function $F$ is extended from rule names to events:

$$F_{p'} = F_p \quad \text{for } p = \chi_P(p').$$

Putting the parts together, we obtain a generalised semi-Markov scheme $(Z, E, active, next, F)$. The GSMP associated with this scheme defines the semantics of $\mathcal{SG}$.

## 5 Analysis

Once a Generalised Semi-Markov Process is obtained, one can analyse its properties in order to get knowledge on the behaviour of the system which was modelled. Important aspects include the behaviour on the long run – the *steady state* – as well as transient analysis, which gives the probability that a transition is performed in

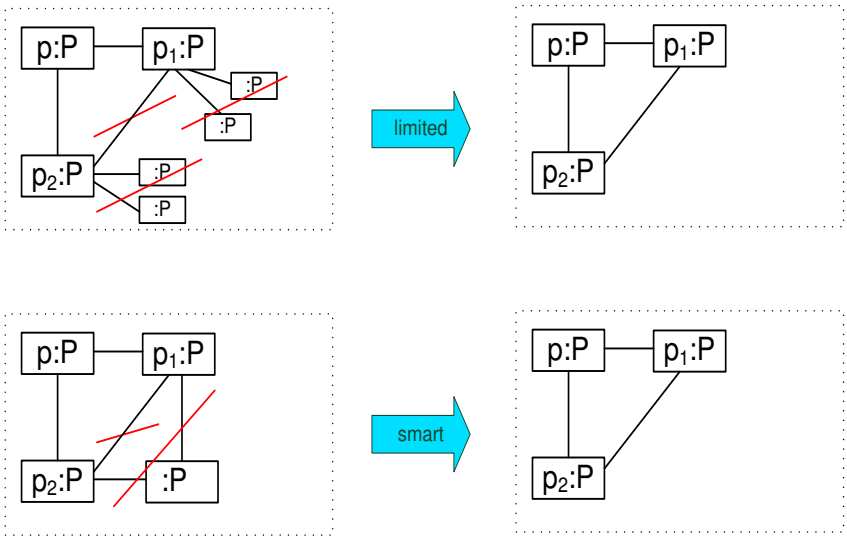



Figure 2. Peer-to-peer network: Alternative shortcut rules

a certain period of time.

The numerical analysis of stochastic graph transformation systems can be done in different ways. The approach proposed in [11] consists in generating the state space of the system, transforming the result to a stochastic model, and using existing model checking tools to analyse its properties. This procedure suffers from the drawback that state space generation is very complex due to the fact that isomorphic graphs have to be determined. When leaving the realm of exponential distributions, model checking tools become less efficient, and arbitrary distributions are usually not covered. So we use Monte Carlo simulation as an alternative approach [5]. The intuitive idea is to traverse a number of randomly chosen paths through the system, and thereby sample information on its behaviour. More precisely, pseudo-random numbers are generated according to a given distribution. Depending on the outcome of this pseudo-random experiment, the successor state of the currently occupied state is chosen. Repeating this procedure results in a path through the system, and generating a sufficient number of such paths, one can make predictions on the system's properties.

The *Event Scheduling Scheme* proposed in [5, Sect. 10.2] provides a simulation algorithm for GSMPs. First, an initialisation procedure has to be performed: The *state* of the system is set to the initial state, the *simulation time* is set to 0. Random numbers $t$ are determined for all events $e$ active in the initial state, and the *scheduled event list* is initialised with them, with all entries sorted in increasing order according to their scheduled times.

After that, the following steps are repeated until some termination condition is

reached:

**Step 1** Remove the first entry $(e, t)$ from the scheduled event list.

**Step 2** Update the simulation time by advancing it to the new event time $t$.

**Step 3** Update the state according to the state transition function $z' = next(z, e)$.

**Step 4** Delete from the scheduled event list all entries corresponding to inactive events in $z'$, i.e. delete all $(e_k, t_k)$ such that $e_k \notin active(z')$.

**Step 5** Add to the scheduled event list any active event which is not already scheduled (possibly including the triggering event removed in Step 1). The scheduled event time is computed by randomly generating a number according to the event's distribution function and adding it to the current simulation time.

**Step 6** Reorder the scheduled event list such that all entries are sorted in increasing order of their scheduled times.

We prepared an experimental implementation of the event scheduling scheme in Java, using AGG [21] and the stochastic simulation library SSJ [15]. AGG is a rule based tool for graph transformation providing a visual user interface and a Java API. Models are represented by attributed graphs which are typed by type graphs. AGG is based on the single-pushout approach, but also allows to simulate the double-pushout approach by checking the identification- and dangling-edge-conditions. The implementation was used to compare two different strategies for introducing shortcuts in peer-to-peer networks [10]. Figure 5 shows the rules *limited* and *smart*. The first one adds shortcut connections whenever there is no direct connection, with a limited total number of three connections for each peer. The latter rule adds shortcuts whenever there is neither a direct connection, nor a connection via one other peer. Thus, the shortcut is only added if there is no other peer to replace peer $p$ in case of failure [17].

Visual Modelling of stochastic graph transformation systems was done with the AGG graphical user interface. The additional information on the distribution associated with each rule was provided in a text based property file. Integration of this information into existing tools is an interesting issue as it would be more convenient for the user.

The main objective of stochastic simulation is to estimate quantities related to the modelled system by analysing the simulation results. Depending on whether we are interested in the *long run* behaviour of the system or in transient analysis, different simulation techniques have to be applied. We will shortly discuss both cases.

An interesting long run property of our network example is the proportion of unconnected peers. It is possible that a system converges to the steady-state when time progresses and can then be characterised by a discrete distribution over the state space. This is not always the case, e.g. when the number of nodes of a stochastic graph transformation system is not limited. However, we are in general not interested in the steady-state per se, but in the value of some function, such as for example the proportion of unconnected peers. These quantities may converge

as $t \to \infty$, even if no steady-state is reached. A simulation strategy for the long-run behaviour can be described as follows: Let $T$ be the length of the simulation run and let the quantity of interest be $\phi(T)$. Extend the simulation to $2T$ and determine $\phi(2T)$. If $|\phi(2T) - \phi(T)| < \varepsilon$ for some predefined $\varepsilon$, terminate. Otherwise, extend the simulation time $T$ until the condition holds. Of course, this does not guarantee that $|\phi(t_1) - \phi(t_2)| < \varepsilon$ for all $t_1, t_2 > T$, because the system may fluctuate, but it is a reasonable assumption in many practical cases [5].

With transient analysis, the simulation strategy is different. Consider for instance the probability that a peer which uses the network for 1 hour suffers disconnection in this period of time. Here, it is not reasonable to extend the simulation time for more than 1 hour. One simulation run will only give one sample path, and estimates can be obtained by repeating the simulation with the same initial conditions, following the method of *independent replications* [5]. Statistical analysis of the simulation results involves computation of confidence intervals, which is automised in the SSJ framework [15]. The number of simulation runs depends on the desired confidence level. Other transient properties can even involve the simulation time itself. For instance, the average time until an error occurs might be of interest. In this case, a simulation run is performed until an error state is reached, after that the next run is started. The number of runs depends again on the desired confidence interval.

We deployed our experimental implementation to analyse the system from Example 4.3, to which either rule *limited* or rule *smart* was added as shortcut strategy, and compared the results. Both rules were assumed to obey an exponential distribution with parameter 1. As start graph, the empty graph was used. Undirected edges were modelled in AGG by bidirectional one. The property under investigation was the proportion of unconnected peers in the long run. The simulation ran over 100,000 transitions, taking approximately 8 hours on a laptop computer with Pentium M 1.40 GHz processor and 500 MB RAM. Obviously, this time could be reduced significantly on high-capacity workstations. The result was that rule *smart* leads to 99.2% of connected nodes in the long run, while with *limited*, only 91.4% of the peers are connected.

Time measurement was done in milliseconds using the system time. As expected, almost the whole computing time was used for calculating matches. The time needed for one match differed widely depending on the rule and the size of the instance graph. While rule *new* does not require any significant computation time ($<$ 1ms), rule smart was the most time consuming rule, with a range between 100 and 400 ms for graphs of around ten nodes and between 20 and 40 edges. More detailed performance measurement is planned for the future.

The simulation algorithm involves computing all matches of rules to the current graph, and compare them to old matches, in order to compute the probabilities correctly. AGG provides an efficient graph pattern matching algorithm for calculating a single match on the basis of a constraint satisfaction problem. For every match of the current graph, two conditions need to be checked: First, we have to check whether it is a new match. This can be done easily by comparing the unique

object identities of the graph elements involved. Second, the negative application conditions have to be checked again, because they may be violated by parts of the graph which had not been present before the last production. The performance of this procedure proved to be tolerable for experimental purposes. However, in order to perform more realistic simulations, performance of the implementation needs to be improved. A possible solution is provided by the database approach presented in [22]. This technique keeps track of all possible matchings of graph transformation rules in database tables, and updates these tables incrementally to exploit the fact that rules typically perform only local modifications to models. Database management systems provide efficient algorithms for computing and updating views. We plan to take advantage of them for developing an efficient tool.

## 6 Conclusion

In this paper, we propose a new model of Stochastic Graph Transformation with general distributions. This model allows us to study a wider class of systems than the models based on exponential distribution. However analysis of those models is more complex than the restricted one. There are no powerful model checking techniques for Generalised Semi-Markov Processes. This can be partially remedied by Monte Carlo simulation techniques.

In the future we are going to implement a tool for stochastic graph transformation with general distributions. We are going to investigate to what extend data base management systems can be used for this purpose. Our goal is also to study more realistic examples. We will further investigate applicability of already existing simulation techniques to stochastic graph rewriting. On the other hand, we will develop a logic for specification of stochastic properties in SGT and investigate the possibilities of model checking. In many systems, there is a mix between stochastic and deterministic behaviour. We will therefore relate our model to Stochastic Automata.

## Acknowledgement

Discussions with Reiko Heckel improved the ideas presented in this paper. Karsten Ehrig and Olga Runge helped us with AGG. Many thanks to all of them.

## References

[1] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis, *Modelling with Generalized Stochastic Petri Nets.* John Wiley & Sons, 1995.

[2] P. R. D'Argenio, J.-P. Katoen, *A theory of stochastic systems: Part I and II*, Information and Computation, Vol. 203, number 1, pages 1–38 and 39–74, 2005.

[3] P. Baldan, A. Corradini, U. Montanari, *Unfolding and Event Structure Semantics for Graph Grammars.* FoSSaCS 1999: 73-89

[4] E. Brinksma, H. Hermanns, *Process Algebra and Markov Chains*, In: Lectures on Formal Methods and Performance Analysis, editors: Ed Brinksma and H. Hermanns and J.P. Katoen, Springer LNCS 2090, pages: 183 – 231, 2001

[5] Chr. G. Cassandras, St. Lafortune, *Introduction to discrete event systems* Kluwer Academic Publishers, Boston 1999

[6] D.R. Cox, A use of complex probabilities in the theory of stochastic processes. *Proc. Camb. Phil. Soc., 51, 1955, pp. 313-319*, 51:313–319, 1955.

[7] H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. *Fundamentals of Algebraic Graph Transformation.* EATCS Monographs in Theoretical Computer Science, Springer, 2006.

[8] P. J. Haas, G. S. Shedler, *Regenerative stochastic Petri nets*, Performance Evaluation, Volume 6, Issue 3 , September 1986, Pages 189-204

[9] Habel, A., Heckel, R., Taentzer, G.: Graph Grammars with Negative Application Conditions, *Fundamenta Informaticae*, **26**(3,4), 1996, 287 – 313.

[10] R. Heckel, *Stochastic analysis of graph transformation systems: A case study in P2P networks*, In H. Dan Van and M. Wirsing, editors, Proc. Intl. Colloquium on Theoretical Aspects of Computing (ICTAC'05), Hanoi, Vietnam, volume 3722 of LNCS. Springer-Verlag, October 2005

[11] R. Heckel, G. Lajios, S. Menge, *Stochastic Graph Transformation Systems.* In: H. Ehrig, G. Engels, F. Parisi-Presicce, G. Rozenberg (Hrsg.): Graph Transformations: Second International Conference, ICGT 2004, Rome, Italy, September 28–October 1, 2004. Proceedings. Lecture Notes in Computer Science 3256. Springer, Oktober 2004. pp. 210-225

[12] J. Hillston, M. Ribaudo, *Stochastic process algebras: a new approach to performance modeling.* In: K. Bagchi, J. Walrand, G. Zobrist (Eds.), Modeling and Simulation of Advanced Computer Systems, Gordon Breach, 1998.

[13] V.G. Kulkarni, *Modeling and Analysis of Stochastic Systems.* Chapman & Hall, 1995.

[14] M. Kwiatkowska, G. Norman, and D. Parker. PRISM: Probabilistic symbolic model checker. In T. Field, P. Harrison, J. Bradley, and U. Harder, editors, *Proc. 12th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation (TOOLS'02)*, volume 2324 of *LNCS*, pages 200–204. Springer, 2002.

[15] P. L'Ecuyer, L. Meliani, J. Vaucher, *SSJ: A Framework for Stochastic Simulation in Java*, Proceedings of the 2002 Winter Simulation Conference, IEEE Press, Dec. 2002, 234-242.

[16] G. G. Infante López, H. Hermanns and J.-P. Katoen, *Beyond Memoryless Distributions: Model Checking Semi-Markov Chains* PAPM-PROBMIV 2001, Springer LNCS 2165, pp. 5770, 2001.

[17] L. Mariani, *Fault-tolerant routing for p2p systems with unstructured topology.* In Proc. International Symposium on Applications and the Internet (SAINT 2005), Trento (Italy), 2005. IEEE Computer Society.

[18] O. M. Mendizabal, F. L. Dotti, L. Ribeiro, *Stochastic Object-Based Graph Grammars* Brazilian Symposium on Formal Methods SBMF-2005, Porto Alegre

[19] G. Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformation: foundations*, volume 1. World Scientific, River Edge, NJ, USA, 1997.

[20] R. Schassberger, *Insensitivity of steady-state distributions of generalized semi-Markov processes*, Annals of Probability, 5(1):87-99, 1977.

[21] G. Taentzer, *AGG: A Graph Transformation Environment for Modeling and Validation of Software*, Proc. Application of Graph Transformations with Industrial Relevance (AGTIVE'03), Pfaltz, J. and Nagl, M., Charlottesville/Virgina, USA, 2003, http://tfs.cs.tu-berlin.de/agg.

[22] G. Varró, D. Varró: *Graph Transformation with Incremental Updates.* In Proc. GT-VMT 2004, Graph Transformation and Visual Modelling Techniques, Barcelona, Spain, March 2004.