Journal of Object Technology
Published by AITO — Association Internationale pour les Technologies Objets
http://www.jot.fm/

SimSG: Rule-based Simulation using Stochastic Graph Transformation

Sebastian Ehmes[a] Lars Fritsche[a] Andy Schürr[a]

a. Real-Time Systems Lab, Technische Universität Darmstadt

Abstract Stochastic models can be found in various domains. For example, biochemical processes such as molecular interactions or the dynamics of wireless network topologies, where changes occur with certain probabilities. Having the ability to simulate scenarios in these domains can be crucial when real-life observations of certain processes are infeasible, e.g., protein-protein interactions in biochemistry, or expensive, e.g., building large wireless networks for research purposes. Stochastic graph transformation systems provide the means to describe the structure and simulate the behavior of such probability-driven environments in an adequate way, by modelling the state transitions using graph transformation rules, whose application depends on the current state and their application probabilities. To the best of our knowledge, there is currently no general-purpose simulation tool available anymore that performs rule-based simulations using stochastic graph transformation. Therefore, we developed SimSG a modular stochastic simulation tool that addresses the needs of a wide range of application domains - in contrast to most specialized simulation tools that are limited to one domain only. To facilitate the versatility of the tool, SimSG can be configured to employ different general-purpose tools for incremental graph pattern matching (currently, Democles and Viatra). We evaluate SimSG based on two use cases: First, using an example of the biochemistry domain, we conduct a comparative evaluation against the domain-specific tool KaSim. Second, we underpin the general-purpose applicability of SimSG by analyzing the simulation of a wireless sensor network scenario.

Keywords Stochastic Graph Transformation, Model-driven Development, Rule-based Simulation

1 Introduction

When we want to model processes, such as molecular reactions in biochemistry or the topology dynamics of a wireless sensor network, we are often confronted with complex and dynamic structures as well as structural changes that are stochastic in

Sebastian Ehmes, Lars Fritsche, Andy Schürr. *SimSG: Rule-based Simulation using Stochastic Graph Transformation*. In Journal of Object Technology, vol. 18, no. 3, 2019, pages 1:1–17. doi:10.5381/jot.2019.18.3.a1

nature. Graphs are a powerful formalism to describe such structures, while graph transformations are a natural means to describe modifications of these structures according to a set of rules. For example, in biochemistry nodes represent molecules and edges describe bonds between molecules. In this case, graph transformation rules describe molecular binding or decay processes. Another scenario that can be modeled with graphs is that of wireless computer networks, where communication link reconfigurations or failures can be described by graph transformations as well. Both scenarios have in common that they are driven by stochastic model changes, which occur with certain probabilities. For example, bonds between two molecules or wireless connections between two devices can break with certain probabilities. Such a behavior can be described by stochastic graph transformations [HLM04]. Stochastic graph transformation approaches annotate graph transformation rules with certain probabilities that determine whether a rule is applied or not. By extending graph transformations, it is possible to model structures of the previously mentioned scenarios and to capture their stochastic properties as well. Since stochastic graph transformations are able to model stochastic environments, this approach naturally lends itself to be used as an approach for the model-driven development of rule-based stochastic simulations. An advantage of this approach to simulations is the fact that it is possible to model processes as a set of rules, abstracting from the complexities of a specific problem domain (e.g. differential equations in biochemistry).
There are many domain specific tools that implement simulations based on stochastic graph transformations (e.g. KaSim [BFKF19] and RuleMonkey [CMG+10]) but, to the best of our knowledge, there is currently no general-purpose tool available (GraSS [THR10] was discontinued). To fill this gap we developed a new tool called SimSG, which performs rule-based simulations using stochastic graph transformations in order to describe stochastic processes. The tool offers well-defined interfaces to a number of exchangeable core components such as incremental graph pattern matchers like Viatra [VBH+16] and Democles [VD13]. The industrial-strength tool Viatra offers, e.g., better scalability for large graph data simulations, whereas the research prototype Democles is more appropriate for experiments due to its light weight implementation. SimSG is designed to be a general-purpose tool and, therefore, allows for the simulation of scenarios from very different problem domains. Our simulation tool can be applied to all scenarios where structures can be described by graphs and changes in those structures can be expressed by means of stochastic graph transformations.
This concept of (stochastic) graph transformations is explained in Section 2. In Section 3 we present our tool and explain the modular architecture as well as the implementation of a rule-based stochastic simulation. We evaluate the tool in Section 4 by using examples from biochemistry and the network domain. In Section 5 we discuss related work. Finally, in Section 6 we sketch further prospective application domains and suggest possible future extensions of the tool.

2 Background

2.1 Graph Transformation

Graph transformations (GT) [EEPT06] provide the means to express model transformation in a declarative and rule-based manner. Models and metamodels are, therefore, considered as graphs $G = (V, E)$ with V being the set of vertices representing model entities and E being the set of directed edges representing relations between entities.

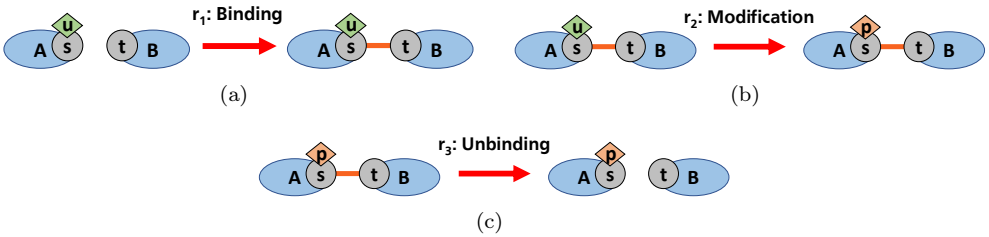


Figure 1 – Running Example - Phosphorylation

A graph transformation system $GTS = (G, R)$ consists of a graph G and a set of graph transformation rules R.

$$r \in R : G \xrightarrow{r@m} G' \tag{1}$$

A rule r as shown in Eq. (1) describes a transformation from a graph G to a graph G'. Additionally, a rule contains a left-hand side (LHS) and a right-hand side (RHS). The LHS defines a precondition, which has to be present in G before the rule can be applied, while the RHS defines the postcondition that has to hold in G' after the precondition was met. A subgraph in G that matches the LHS of the rule is called a match, with $m(G, r)$ representing the set of all matches to the rule r. The $r@m$ notation in Eq. (1) indicates that r is applied to a match of this set. The process of finding a match is referred to as pattern matching.

Pattern matching approaches can be sorted roughly into two categories: batch and incremental pattern matching approaches. Batch pattern matcher usually find matches by solving a constraint satisfaction problem [LV02] or through local search algorithms using search plans [Zün96]. When a graph transformation engine eventually modifies the model graph according to a rule, all previously found matches are discarded and the matching process is restarted from scratch. Incremental pattern matcher try to avoid discarding and regathering matches in the event of a rule application. Instead, changes to the model are registered and the set of matches is updated incrementally, according to those changes. A well-known approach to incremental pattern matching is the Rete algorithm developed by Forgy [LF82]. Prominent tools implementing Rete are Viatra and Democles.

Phosphorylation serves as a running example from biochemistry to motivate the use of graph transformations and later stochastic graph transformations. This process is a recurring mechanism in biochemistry, especially in biochemical signaling pathways. A molecule activated through phosphorylation can interact with other molecules that require a phosphoryl group for binding. Such an activated molecule may also spontaneously deactivate itself again, i.e., lose its phosphorylation. This mechanism serves as a regulator for metabolic processes in most life forms, where phosphorylation acts like a switch. The rule in Fig. 1(a) is an example of a graph transformation rule, using a domain-specific notation. Rule r_1 models the first part of the phosphorylation reaction. The LHS contains a molecule `A` with a site `s` where other molecules can bind to. Site `s` is in an unphosphorylated state `u`, which means that no phosphoryl group is attached to it yet. Additionally, the LHS describes a second molecule `B` with its own site `t`. Consequently, when r_1 is applied, the activating molecule `B` attaches its own

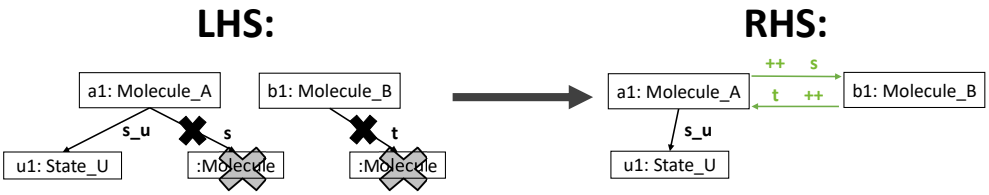


Figure 2 – GT Example: Binding

site `t` to the unphosphorylated site `s` of the left molecule `A`, as described by the right side of r_1. The second rule r_2 describes molecule `B` phosphorylating `A` by attaching a phosphoryl group to site `s` (see Fig. 1(b)). Rule r_3 describes molecule `B` detaching again, i.e., the bond between site `s` and `t` breaks and leaves behind a phosphorylated molecule, as shown in Fig. 1(c).

Graph transformation rules are applied as follows, elements that appear on both LHS and RHS are considered context and are not to be changed. Elements that appear only on the LHS are to be deleted, while elements only appearing on the RHS are created [BCW17]. However, this notion does not allow to specify the explicit absence of unwanted nodes. For this purpose, negative application conditions (NACs) can be used to specify the elements that must be absent from a subgraph in order to fulfill the precondition of a rule. In short, the LHS specifies elements that must exist in a matching subgraph, while NACs describe what must not be attached to a matching subgraph, such that a rule may be applied to it. Fig. 2 shows a graph-based representation of rule r_1 to illustrate the usage of NACs. The LHS of a rule is a connected graph consisting of three vertices, two representing a molecule of type `A` and type `B`, respectively. The third vertex describes an unphosphorylated state `u`. The unphosphorylated state of site `s` is expressed via the labeled edge `s_u` pointing to state `u`. Molecular sites are represented by labeled edges `s` and `t`, which in this case must not yet exist, i.e., the molecules must not be connected to any other molecules. This is depicted by the crossed-out reference arrows in Fig. 2 and is an example of a NAC. Consequently, matches are discarded if labeled edges `s` and `t` exist and are already connected to some other molecule. When a match to this LHS is found, the rule is applied to the subgraph of the model, which is altered to comply with the RHS.

2.2 Stochastic Graph Transformation

The intention of our tool is to simulate stochastic processes such as binding reactions in biochemistry or the dynamics of wireless sensor networks. Both scenarios have in common that changes occur with a certain probability, which has to be modeled appropriately. A system modeling the occurrence of such stochastic processes can be described by *continuous-time Markov Chains* (CTMCs) [Nor97]. A $CMTC = (S, Q)$ consists of a countable set of states S and a transition matrix Q. Entries in the Q-matrix describe whether a transition from a state to another state exists and with which probability it might occur. Typically, the transition delay, i.e., the time to wait for a certain state transition to happen, is distributed exponentially. The probability is being determined by the corresponding entry in Q. However, it is also possible to use other probability distributions besides the exponential distribution. This is useful whenever a stochastic model is required to model waiting times, i.e., transitions becoming more likely the longer they have to wait. When transition delays are

not described by the memoryless exponential distribution, we speak of semi-Markov processes [KL07].
When we think about graph transformations in the context of CTMCs, we can describe a state by a graph G and the set of states by a set of graphs $\mathcal{G}$. Hence, the transition from a state G to G' corresponds to the application of rule $r : G \xrightarrow{r@m} G'$ with $G, G' \in \mathcal{G}$. Yet, a GTS still does not incorporate the stochastic property of state transitions in CTMCs, which leads to the idea of stochastic graph transformation systems (SGTS), introduced by Heckel et al. [HLM04]. A $SGTS = (GTS, \rho)$ consists of a graph transformation system, where the graph transformation rules are associated with an application rate $\rho(r) \geq 0$ with $r \in R$. These rates model $P(r, G)$, the probability with which a rule r is applied to a graph G, when a single match of its LHS occurs. Furthermore, $P(r, G)$ determines the transition delay, i.e., the time to wait until a transition that is described by a rule occurs. Given that a transition from G to G' corresponds to a rule application, rates $\rho(r)$ correspond to entries in Q, which again model state transition probabilities. In [HLM04], Heckel et al. show that such a stochastic graph transformation system can be translated into a CTMC.
To build a rule-based simulation of stochastic processes using SGTSs, two practical questions have to be answered. First, how do we pick a rule with a certain application rate, given a state represented by a graph G and second, how do we calculate the transition delay?
There are different methods to solve these problems. A well-known approach from biochemistry that is used in various simulation tools is Gillespie's algorithm [TG77]. Another approach that was used for the simulation of peer-to-peer networks is the generalised semi-Markov scheme [KL07]. Both works describe how to determine the next state transition as well as the transition delay in their respective domains.

Gillespie's Algorithm implements the idea of a CTMC using exponentially distributed transition delays, which can be seen in Eq. (2). Given a system state modeled by a graph G, $P(\tau, r)$ describes the joint probability for the next occurring state transition described by rule $r : G \to G'$, happening in the time interval $[t, t + \tau]$. Gillespie shows that $P(\tau, r) = P(\tau)P(r)$, with $P(\tau) = a_0 e^{-a_0 \tau}$ representing the probability for any rule application occurring within a time step τ and $P(r) = a_r \,/\, a_0$ describing the probability that the next rule is r. The parameter a_r is the rule activity and corresponds to the probability of a rule r being applied, given a certain graph G. Parameter a_0 is the system activity, it represents the current state of the system and corresponds to the overall probability of a rule application within the next time step τ.

$$P(\tau, r) = \begin{cases} a_r e^{(-a_0 \tau)} & \text{if : } \ 0 \leq \tau < \infty \text{ and } r \in R \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

Consequently, each step of a simulation implementing Gillespie's algorithm determines the next rule and the corresponding time interval τ by generating the random pair (τ, r), w.r.t. Eq. (2).
To determine τ and r, each simulation step begins by calculating the rule activity a_r for each rule $r \in R$ according to Eq. (3), given the current graph G.

$$a_r = |m(g, r)|\, \rho(r) \ , \qquad r \in R \tag{3}$$

Therefore, the activity of a certain rule r is the product of the number of matches to the LHS of r and the annotated rate $\rho(r)$, mentioned in Section 2.2. Graph $G1$ in Fig. 3 represents a system state that contains six molecules. $G1$ contains two matches

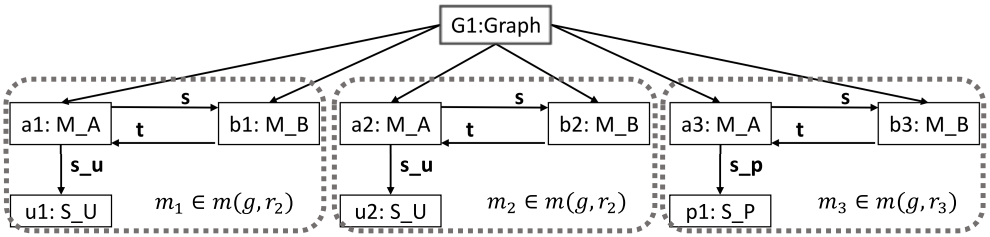


Figure 3 – Example Graph

to the LHS of rule r_2, one for rule r_3 and none for r_1. Assuming each rule is annotated with a probability of $0,2$, the activities would amount to $a_2 = 0,4$, $a_3 = 0,2$ and $a_1 = 0$.
The next step is to determine the system activity a_0 according to Eq. (4), representing the sum of all rule activities. In the example the system activity amounts to $a_0 = 0,6$.

$$a_0 \;=\; \sum_R a_r \;=\; \sum_R |m(g,r)|\,\rho(r) \tag{4}$$

Given a_0 and a random number $x_1 \in [0,1]$ drawn from the uniform distribution $\mathcal{U}(0,1)$, the next time step τ with the corresponding probability $P(\tau)$ results from Eq. (5). Consequently, the time interval τ decreases if the system activity a_0 increases.

$$\tau = \left(\frac{1}{a_0}\right) ln \left(\frac{1}{x_1}\right) \tag{5}$$

Usually more than one rule may be applicable to a certain graph G. Therefore, a specific rule $r \in R$ with the probability $P(r)$ has to be selected from a set of competing rules. Given a_0 and a random number $x_2 \in [0,1]$ drawn from the uniform distribution $\mathcal{U}(0,1)$, the rule r is picked, which satisfies Eq. (6).

$$\sum_{i=1}^{r-1} a_i \;<\; a_0\, x_2 \;\leq\; \sum_{i=1}^{r} a_i \tag{6}$$

In essence, the rule with the highest activity has the highest probability to be picked from the set of competing rules. Regarding the example, rule r_2 has the probability $P(r_2) = \frac{2}{3}$, r_3 has $P(r_3) = \frac{1}{3}$ and r_1 has 0 probability of being picked. This makes r_2 the most probable candidate to be picked as the next rule.
In this section we presented a compact explanation of Gillespie's algorithm. Please refer to Gillespies original work [TG77] for the full derivations of Eq. (5) and Eq. (6).

Generalised semi-Markov scheme (GSMS) devised by Kosiuczenko et al. [KL07] represents a different approach for determining transitions and the corresponding delay compared to Gillespie's algorithm. A $GSMS = (\mathcal{G}, E)$ has a set of graphs $G \in \mathcal{G}$ representing states and a set of events $e \in E$. An event $e(m,r)$ contains a rule-match pair, with $m \in m(G,r)$ and $r \in R$. A GSMS tracks individual matches to preconditions of rules as distinct events, where each event is annotated with its own continuous distribution function. In each subsequent graph G, newly discovered matches create new events, while matches that became invalid due to a rule application lead to the deletion of the corresponding event. Each event has its own execution time,

where the contained rule r is applied to its assigned match $m \in m(G, r)$, leading to a transition described by $r : G \xrightarrow{r@m} G'$. The execution time is determined by the current state described by a graph G and the annotated continuous distribution. When the exponential distribution is used, the execution time can be determined similar to τ in Eq. (5). A rule is chosen by picking the event that has the smallest time to execute left. The advantage of the GSMS approach is the fact that it enables the usage of other probability distribution functions and by tracking individual events allows for distributions to have a memory. In contrast to Gillespie's algorithm, where the next rule and the transition delay is determined by solely relying on match counts and the exponential distribution. On the downside, the GSMS approach requires tracking each match to every rule's LHS, which can be more costly when compared to just counting match occurrences.

3 Implementation

This paper presents a new tool for rule-based simulations of stochastic processes, such as biochemical reactions or wireless sensor network topology reconfigurations. SimSG[1] is composed of several modular components. As depicted in Fig. 4, the architecture of the tool can be divided into three major aspects: *Models*, *Configuration*, and *Simulation*.

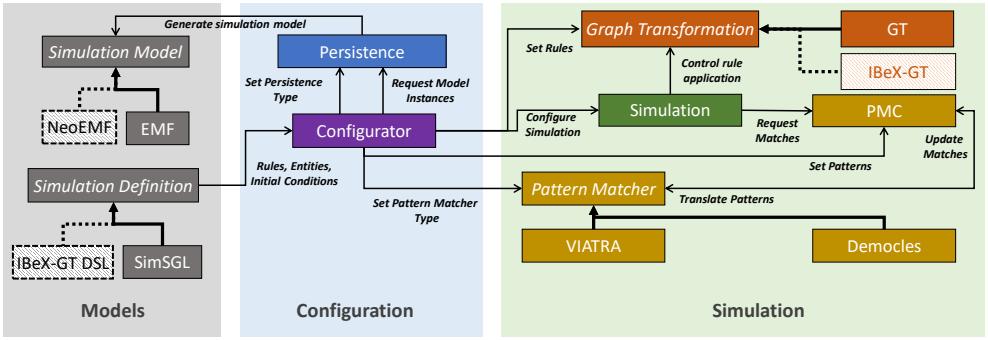


Figure 4 – SimSG Modules

The Models section encompasses the simulation definition and the simulation model. The simulation definition describes all occurring simulation entities and their initial number according to a set of initial conditions. Additionally, it contains a set of rules modeling the desired behaviour of the simulation and its contained entities. Currently, a simulation definition can be created with the domain-specific language (DSL) SimSGL, which was specifically created for this tool in order to describe the simulation definition in a textual fashion.

```
rule r1 A(s{u}[free]), B(t[free]) -> A(s{u}[1]), B(t[1])
```

Listing 1 – Phosphorylation binding rule r1 Fig. 1(a) in SimSGL syntax

For example, the first step of the phosphorylation process described by rule `r1` in Fig. 1(a) can be expressed in SimSGL syntax as shown in Listing 1. On the left-hand side of the arrow, `r1` requires a pair of molecules, where one is of type `A`, one is of type `B` and both are not connected (`free`) at sites `s` and `t`. Additionally, site `s` must be in

[1]SimSG eclipse plugin and example project: https://github.com/eMoflon/SimSG-Example

an unphosphorylated state `u`. The outcome of a rule application is described on the right-hand side of the arrow. Consequently, when `r1` is applied both molecules will be connected at sites `s` and `t`, since both share the common index `1`. In the future we plan to support other DSLs that describe graph transformations besides SimSGL.
A simulation model contains all simulation entities that have been created according to initial conditions. These entities are contained within a model graph, on which stochastic graph transformations are performed according to the rules specified in the simulation definition. Currently, the EMF[2] framework is used to implement these graph structures. Since, EMF tends to work less well with increasing numbers of entities in the model graph, the architecture was designed to be able to use other EMF conform graph frameworks. For example, NeoEMF[3], which is an EMF wrapper for graph databases.

An important characteristic of our tool is to provide different configurations and thus modularity. Parts of the Configuration aspect are: model persistence, i.e., saving and loading of simulation models as well as simulation definitions and the ability for the user to activate, deactivate or change modules of the simulation tool. The configurator-module allows the user to set the implementation type of the model graph, e.g. EMF and advises the persistence module to create or load a simulation model according to a given simulation definition. Consequently, it provides the graph transformation module with graph transformation rules and the pattern matcher module with the corresponding patterns. Furthermore, the configurator enables the user to choose between different pattern matching tools and in the future between different graph transformation tools. Currently, the tool supports Viatra[4] [VBH+16] and Democles[5] [VD13], which represent general purpose incremental pattern matcher.

Simulation as the third aspect of the tool's architecture covers all modules and functionality that form the functional core of the rule-based simulation. This includes: pattern matching, stochastic model graph transformation through rule application, checking for termination conditions and recording simulation statistics. The simulation module is designed in such a way that it is agnostic to the used pattern matcher. The pattern matching controller (PMC) serves as a layer of abstraction between SimSG and graph pattern matching tools. The PMC translates patterns that were defined in the simulation definition to the adequate representation of the configured pattern matcher. This module is interchangeable as well, which makes it possible to customize and adapt the pattern matching process to better suit the specific needs of a domain.

3.1 Simulation Approach

The focus was to provide a simple and modular architecture to support different approaches, such as Gillespie's algorithm and GSMS, or other algorithms that use stochastic graph transformations. Fig. 5 shows the tool's implementation of a rule-based simulation driven by stochastic graph transformations.

A simulation begins with step 1 where the state of the model is updated. This means that the pattern matching engine is ordered to update its internally stored

[2]EMF project page: https://www.eclipse.org/modeling/emf/
[3]NeoEMF project page: https://www.neoemf.com
[4]Viatra project page: https://www.eclipse.org/viatra/
[5]Democles Git repository and documentation: https://github.com/eMoflon/emoflon-ibex-democles

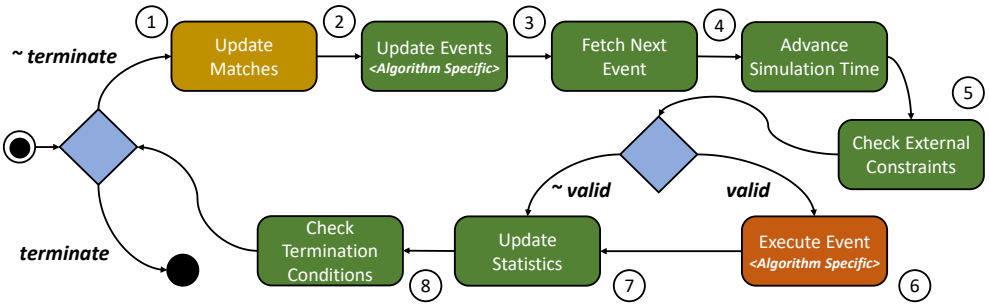


Figure 5 – Simulation Loop

matches to patterns that are specified by the rules in the simulation definition model. EMF conform frameworks ensure that a notification is automatically sent to the pattern matcher, each time the model is changed by the application of a rule. Following that, all registered changes that were applied to the model in the last iteration are taken into account and all matches are incrementally updated.

In step 2, the updated matches are used to update a sorted queue of events. These events contain a rule-match pair (see Section 2.2) and the time at which they will be executed. The top of this queue is always the event that has the smallest amount of time left to execute. Gillespie's algorithm is the tool's standard implementation for this step. Since its transition delay time is calculated by a memoryless exponential distribution function, it will always put exactly one event into the queue. In contrast, an approach using GSMS (see Section 2.2) will create a new event for each new match that was found, remove invalid matches from the queue and sort the queue by time to execute.

Step 3 pops the next event from the queue, followed by step 4, which advances the simulation time according to the execution time, stored within the current event.

In step 5 possible user defined external rule application constraints are checked before the rule of rule-match pair, contained within the current event, is applied to its corresponding match. The idea behind this intermediate step, is to give the user the ability to define complex rule application conditions that go beyond the expressiveness of graph patterns. For example, a user might want to simulate molecular binding processes but prevent impossible bindings, which can be analyzed through geometric constraints. Therefore, such a geometric constraint would check whether parts of molecules that should be connected, according to some rule, will actually collide. In this case the constraint would prevent the application of the rule and the simulation would continue at step 7. If no external constraint is violated, the simulations continues with the next step.

In step 6, the rule of the rule-match pair is applied to the match. Consequently, a graph transformation according to the rule is performed, which leads to further changes in the model graph.

Following that, in step 7, the simulation's state is used to record simulation statistics. For example, a pre-implemented statistic module tracks match counts to user defined

patterns that are of particular interest to the user. These statistics are stored in addition to the current simulation time and can be displayed as an xy-plot, once the simulation has finished. Additionally, the user may implement any other custom statistic modules, in order to track simulation data of interest.

In the final step of the loop, it is checked if one of the termination conditions defined in the simulation definition model is satisfied. For this purpose, the simulation's state is checked whether a possible time or iteration limit has been reached. When this is not the case, it is checked whether limits for match counts to certain patterns have been defined and if the corresponding match counts surpass these limits. In addition to these pre-implemented criteria, the user may implement additional custom termination conditions. When one of the defined termination criteria is fulfilled, the main loop of the simulation is exited. Otherwise, the simulation is resumed at the first step.

4 Evaluation

We use SimSG to simulate a model of the biochemistry domain as well as a wireless sensor network model and present the obtained results. The aim is to evaluate if the implementation of the rule-based simulation approach delivers plausible results. Using the biochemistry example we will evaluate if our simulation results are consistent with results of the domain specific tool KaSim [BFKF19]. Furthermore, the simulation of a wireless sensor network scenario serves as a demonstrator for the general applicability of SimSG to different domains.

4.1 Simulation of the EGF Signal Pathway

The epidermal growth factor (EGF) signaling pathway [CC79] is a regulatory mechanism for the growth of skin cells and serves as an example of a biochemical process that can be simulated with the tool presented in this paper. The process begins with EGF molecules bonding to EGF-receptors (EGFR) on the outside of the cell, whereby the EGFRs activate adjacent EGFRs as a result. Consequently, EGFRs release phosphorylated molecules, also called activated molecules (Section 2.1), into the cell. These molecules, for example, Grb2 activate other molecules, which in turn activate more molecules. This cascade of activation processes continues until ERK molecules reach the nucleus and initiate growth processes, as illustrated by Fig. 6. Note that through EGFRs activating their neighboring EGFRs, a single EGF may cause multiple reaction cascades at once.

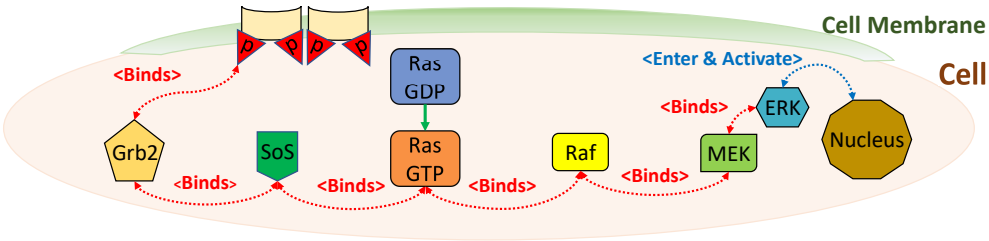
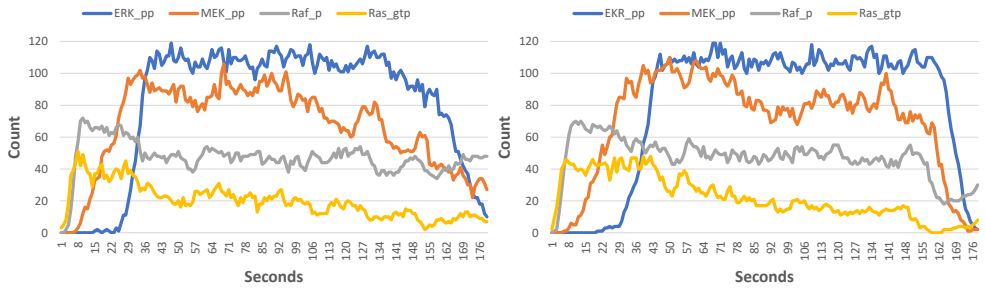


Figure 6 – The EGF Signal Pathway

This process was simulated once with SimSG (Fig. 7(a)) and once with KaSim (Fig. 7(b)), each using a model of the EGF signal pathway created by Danos et al.

[DFF+07]. Consequently, each simulation was initialized with the same number of objects according these specifications.
Despite both results in Fig. 7 not being exactly equal, which makes sense since this is a stochastic simulation, they do display a rather similar behavior. Both simulations show the expected cascade of activated molecules in the same order, beginning with the rise of activated RasGTP molecules (`Ras_gtp` plot) that cause a subsequent rise in activated molecules of the pathway. Towards the end of the simulation initially activated molecules begin to fade out causing a collapse of activated ERK molecules (`ERK_pp` plot).



(a) SimSG (b) KaSim

Figure 7 – Simulation Results - EGF

To examine the scalability of the tool a series of simulations of the EGF signal pathway were performed, while measuring the runtime until completion. Each set of simulations used different simulation model sizes, meaning that the number of molecules generated according to initial conditions were scaled. The scaling begins with 25% of the original model size, then continues to 50%, to 75%, to 100%, and finally, up to 125%. At 100% size the model graph contains 1235 nodes (modeling molecules) and 1810 edges (modeling states and molecular bonds). The y-axis in Fig. 8 indicates the runtime in seconds, while the x-axis displays the relative model size. Each data point in both series represents the average runtime of five simulations, while Viatra was used as the incremental graph pattern matching tool.

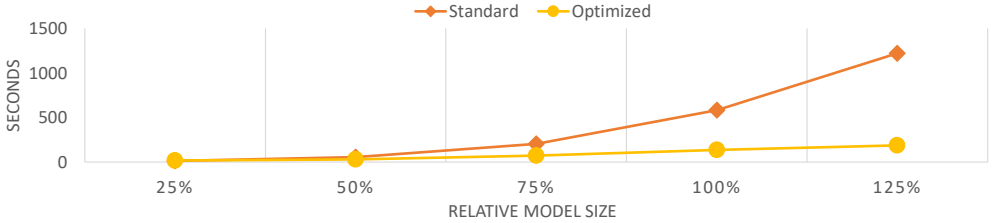


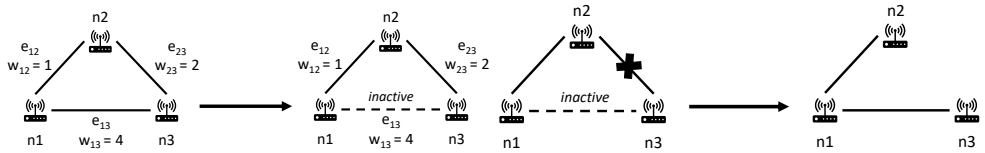
Figure 8 – Simulation Runtime Measurements

The first plot (`Standard`) shows simulation runtime results using the standard tool configuration. These results indicate that runtimes are growing in an almost exponential fashion. This exponential growth can be explained by the typical patterns that are often found in the biochemistry domain. When we look at the first rule of the running example (Section 2.1), we can see that the LHS requires two unbound molecules, which is an example for a disjoint pattern. Unfortunately, the number of match occurrences to these types of patterns grows very fast, finding all instances of them in a model graph leads to an exponential increase in runtime.

The second plot (`Optimized`) shows runtime results created with a different tool configuration that uses an optimized pattern matching technique by utilizing domain specific knowledge to mitigate the negative effects of disjoint patterns. The optimization is based on the fact that we only need the match counts to calculate probabilities in each simulation step. That means we don't actually need to find all the matches. Instead of finding and counting all possible match combinations of disjoint patterns, we can calculate the number of matches based on the match count of their non-disjoint sub-patterns. Consequently, by only using the sub-patterns we prevent the pattern matcher from finding all possible matches of the original disjoint pattern. When a rule application requires an actual match we construct a corresponding match on demand from the matches of the non-disjoint sub-patterns. As a result, the growth in runtime becomes almost linear, demonstrating the advantage of a modular architecture. However, the domain specific tool KaSim is still much faster, because it is highly optimized for the simulation of protein-protein interactions instead of pursuing a general-purpose approach.

4.2 Simulation of Topology Control in Wireless Sensor Networks

Topology control (TC) algorithms [San05] operate on the underlay topology of a physical network, such as wireless connections between adjacent devices. The aim of such algorithms is to maintain a stable connected network in a dynamic environment, while reducing the energy consumption by deactivating unnecessary connections.



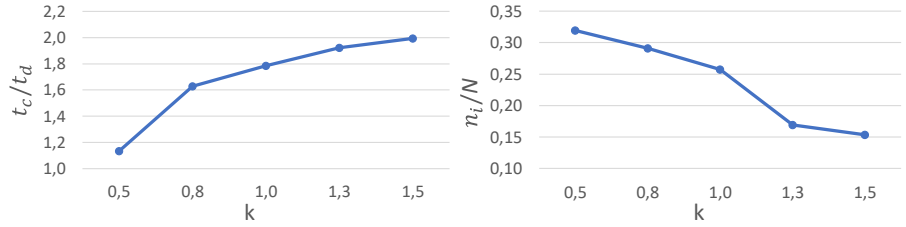
(a) Inactivation (b) Reactivation

Figure 9 – kTC Algorithm

The kTC algorithm [SWB+12] is an example of a TC algorithm that assumes a local view of the network with a limited knowledge of the neighboring network nodes. For example, in Fig. 9(a) node `n1` only knows its two neighbors `n2` and `n3`. Edges between nodes model communication links that are annotated with weights that represent the cost of using the link. The algorithm inactivates links that satisfy the following criteria: First, if a link e_{13} is part of a triangle (here, with links e_{12} and e_{23}) in which $w_{13} > w_{12}$ and $w_{13} > w_{23}$ is satisfied, it is the weight-maximal link. Second, a link shall be inactive if it is weight-maximal and its weight is at least k times larger than the minimal weight, i.e., $w_{13} > k \cdot min(w_{12}, w_{23})$. In Fig. 9 edge e_{13} has a larger weight than edges e_{12} and e_{23}. Assuming $k = 2$, w_{13} is at least two times larger than w_{12}, leading the kTC algorithm to deactivate e_{13}. The value of k can be tuned to adapt the kTC algorithm to different scenarios. Consequently, the k-value decides whether the algorithm has to be less or more aggressive when deciding which communication link to inactivate. Besides deactivating links, kTC can also reactivate links when crucial connections break (Fig. 9(b)) or their weights change due to changes in the environment.

For a simulation of the kTC we use a network model that contains seven nodes and 13 links between nodes. The simulation tool is used to simulate a dynamic environment in which wireless sensor nodes form a network, while the kTC algorithm runs

on top and manages the network topology. Changes in the environment, also called context events, represent obstacles appearing between adjacent devices or varying distances between moving devices. Such context events are modeled by rules that delete links or modify the cost of a link. In the simulation, context events occur at random according to certain probabilities in order to represent the probabilistic properties of real world scenarios. Therefore, context event rules are annotated with certain rates (Section 2.2) modeling the occurrence of obstacles or changes in distance between devices as stochastic events.



(a) Robustness (t_c / t_d) (b) Inactive links (n_i / N)

Figure 10 – Analysis of kTC using SimSG

We performed a series of simulations were we varied the value of k and evaluated the robustness (Fig. 10(a)) of the network as well as the ratio of inactive links (Fig. 10(b)). Each data point in both figures represents the average result of five simulations. The plot in the left figure shows the ratio of the total time during which the network was connected t_c to the total time where the network was disconnected t_d. We can see that a larger k-value leads to a network that remains in a connected state for longer periods of time, making the algorithm less aggressive and the network more robust. The plot in the right figure, displays the average ratio of inactive edges n_i to the sum of active and inactive edges $N = n_a + n_i$, giving a rough indicator of potentially saved energy. As the plot shows, a lower k-value leads to a higher amount of inactive edges, making the algorithm more aggressive and potentially saving more energy.
In summary, the observations made in our simulations fall in line with findings presented in [SWB+12] and show that SimSG produces plausible results in different problem domains.

5 Related Work

The presented tool performs rule-based simulations of stochastic processes using stochastic graph transformation. This is an approach that has over the years gained more traction in the biochemistry domain. Consequently, several domain-specific rule-based simulation tools have been developed to simulate biochemical processes such as protein-protein interactions. These processes are usually modeled with the help of domain-specific languages such as Kappa [DL04] or the BioNetGen-Language [LBFGH04]. For example, Kappa does not require the user to write down differential equations to model biochemical processes. Instead, the Kappa language uses rules to define interactions among molecules. The modeling language SimSGL developed for our tool is inspired by Kappa but adds the possibility to define additional application constraints to rules, such as attribute constraints. This enables the user to define simulation models (e.g., wireless sensor networks) that go beyond the capabilities of biochemistry domain-specific languages.

KaSim [BFKF19] and RuleMonkey [CMG+10] are two well known examples of biochemistry simulation tools. Like most rule-based methods, KaSim is based on Gillespies's algorithm, while RuleMonkey uses a method closely related to that. Both tools perform stochastic simulations of biochemical processes in a similar fashion compared to SimSG but they are highly specialized to protein-protein interactions and, therefore, tend to deliver a better performance in such scenarios. At the same time the high degree of specialization present in KaSim and RuleMonkey limits their scope of application domains. The tool presented in this paper is not limited to a specific problem domain nor is it limited to Gillespie's approach. The use of general-purpose pattern matching tools enables the user to define rules suitable for a wide variety of problem domains, while the modular architecture allows for domain specific optimizations and the use of other simulation algorithms such as GSMS.

To our best knowledge, GraSS developed by Torrini et al. [THR10] is the only other rule-based general-purpose simulation tool. GraSS performed graph-based stochastic simulations by using stochastic graph transformations as well. Unfortunately, this tool is not available anymore. In contrast to RuleMonkey and KaSim, GraSS did not implement some variant of Gillespie's algorithm. Instead, Torrini et al. employed the concept of generalised semi-Markov schemes (Section 2.2). Therefore, GraSS supported the definition of other probability distributions when calculating the delay between state transitions during the simulation. They presented the simulation of P2P network reconfigurations as an application example. For this scenario, a variety of different probability distributions were required to adequately model processes in the network, therefore, highlighting the strenghts of the GSMS approach.

6 Conclusions and Future Work

With SimSG we created a new general-purpose tool for performing rule-based simulations of stochastic processes using stochastic graph transformations. The model-driven approach and the integration of general-purpose pattern matching tools enables the simulation of scenarios of different problem domains. Our evaluation demonstrated that SimSG is indeed able to cope with scenarios from different domains, such as biochemistry and wireless sensor networks. Additionally, the modular nature of the tool's architecture allows for domain specific optimizations, which was shown in Section 4.1. There are several aspects of our tool that we wish to extend and improve in future works. First, the replacement of the proprietary graph transformation module with the graph transformation engine of the meta-case tool eMoflon [LAS14]. This includes the integration of the IBeX-GT[6] DSL used to describe graph transformation rules in a textual fashion to increase the accessibility of SimSG. Furthermore, we plan the implementation of external constraints such as geometric constraints for scenarios that require spatial and geometric context information to prevent impossible configurations. Another promising extension is the use of complex event processing (CEP), which specializes on the analysis of large event streams. Understanding the rule occurrences as events, a CEP engine could be used to infer deeper knowledge from our simulations by providing information on causal or temporal dependencies.

[6] eMoflon-IBeX project page: https://github.com/eMoflon/emoflon-ibex

References

[BCW17] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. *Model-Driven Software Engineering in Practice: Second Edition*. Morgan & Claypool Publishers, 2nd edition, 2017.

[BFKF19] Pierre Boutillier, Jérôme Feret, Jean Krivine, and Walter Fontana. The kappa language and kappa tools. `https://kappalanguage.org/sites/kappalanguage.org/files/inline-files/Kappa_Manual.pdf`, 2019 (accessed June 6, 2019).

[CC79] Graham Carpenter and Stanley Cohen. Epidermal growth factor. *Annual Review of Biochemistry*, 48(1):193–216, 1979. `doi:10.1146/annurev.bi.48.070179.001205`.

[CMG+10] Joshua Colvin, Michael I. Monine, Ryan N. Gutenkunst, William S. Hlavacek, Daniel D. Von Hoff, and Richard G. Posner. Rulemonkey: software for stochastic simulation of rule-based models. *BMC Bioinformatics*, 11(1):404, Jul 2010. URL: `https://doi.org/10.1186/1471-2105-11-404`, `doi:10.1186/1471-2105-11-404`.

[DFF+07] Vincent Danos, Jérôme Feret, Walter Fontana, Russell Harmer, and Jean Krivine. Rule-based modelling of cellular signalling. In Luís Caires and Vasco T. Vasconcelos, editors, *CONCUR 2007 – Concurrency Theory*, pages 17–41, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[DL04] Vincent Danos and Cosimo Laneve. Formal molecular biology. *Theoretical Computer Science*, 325(1):69 – 110, 2004. Computational Systems Biology. URL: `http://www.sciencedirect.com/science/article/pii/S0304397504002336`, `doi:https://doi.org/10.1016/j.tcs.2004.03.065`.

[EEPT06] Hartmut Ehrig, Karsten Ehrig, Ulrike Prange, and Gabriele Taentzer. *Fundamentals of Algebraic Graph Transformation*. Springer Berlin Heidelberg, 2006. `doi:10.1007/3-540-31188-2`.

[HLM04] Reiko Heckel, Georgios Lajios, and Sebastian Menge. Stochastic graph transformation systems. In Hartmut Ehrig, Gregor Engels, Francesco Parisi-Presicce, and Grzegorz Rozenberg, editors, *Graph Transformations*, pages 210–225, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[KL07] Piotr Kosiuczenko and Georgios Lajios. Simulation of generalised semi-markov processes based on graph transformation systems. *Electronic Notes in Theoretical Computer Science*, 175(4):73–86, 07 2007. `doi:10.1016/j.entcs.2007.04.018`.

[LAS14] Erhan Leblebici, Anthony Anjorin, and Andy Schürr. Developing emoflon with emoflon. In Davide Di Ruscio and Dániel Varró, editors, *Theory and Practice of Model Transformations*, pages 138–145, Cham, 2014. Springer International Publishing.

[LBFGH04] Michael L. Blinov, James Faeder, Byron Goldstein, and William Hlavacek. Bionetgen: Software for rule-based modeling of signal transduction based on the interactions of molecular domains. *Bioinformatics (Oxford, England)*, 20:3289–91, 12 2004. `doi:10.1093/bioinformatics/bth378`.

[LF82] Charles L. Forgy. Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, 19(1):17–37, 09 1982. doi:10.1016/0004-3702(82)90020-0.

[LV02] Javier Larrosa and Gabriel Valiente. Constraint satisfaction algorithms for graph pattern matching. *Mathematical Structures in Computer Science*, 12(4):403–422, 08 2002. doi:10.1017/S0960129501003577.

[Nor97] J. R. Norris. *Markov Chains*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 1997. doi:10.1017/CBO9780511810633.

[San05] Paolo Santi. Topology control in wireless ad hoc and sensor networks. *ACM Comput. Surv.*, 37(2):164–194, June 2005. URL: http://doi.acm.org/10.1145/1089733.1089736, doi:10.1145/1089733.1089736.

[SWB+12] Immanuel Schweizer, Michael Wagner, Dirk Bradler, Max Mühlhäuser, and Thorsten Strufe. ktc - robust and adaptive wireless ad-hoc topology control. In *IEEE International Conference on Computer Communication Networks (ICCCN)*, pages 1–9, 07 2012. doi:10.1109/ICCCN.2012.6289318.

[TG77] Daniel T. Gillespie. Exact stochastic simulation of coupled chemical-reactions. *J. of Physical Chemistry*, 81(25):2340–2361, 12 1977. doi:10.1021/j100540a008.

[THR10] Paolo Torrini, Reiko Heckel, and István Ráth. Stochastic simulation of graph transformation systems. In David S. Rosenblum and Gabriele Taentzer, editors, *Fundamental Approaches to Software Engineering*, pages 154–157, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[VBH+16] Dániel Varró, Gábor Bergmann, Ábel Hegedüs, Ákos Horváth, István Ráth, and Zoltán Ujhelyi. Road to a reactive and incremental model transformation platform: three generations of the viatra framework. *Software & Systems Modeling*, 15(3):609–629, Jul 2016. URL: https://doi.org/10.1007/s10270-016-0530-4, doi:10.1007/s10270-016-0530-4.

[VD13] Gergely Varró and Frederik Deckwerth. A rete network construction algorithm for incremental pattern matching. In Keith Duddy and Gerti Kappel, editors, *Theory and Practice of Model Transformations*, pages 125–140, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[Zün96] Albert Zündorf. Graph pattern matching in progres. In Janice Cuny, Hartmut Ehrig, Gregor Engels, and Grzegorz Rozenberg, editors, *Graph Grammars and Their Application to Computer Science*, pages 454–468, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

About the authors

Sebastian Ehmes is a PhD Student at the Real-Time Systems Lab, contact him at sebastian.ehmes@es.tu-darmstadt.de.

Lars Fritsche is a PhD Student at the Real-Time Systems Lab, contact him at lars.fritsche@es.tu-darmstadt.de.

Andy Schürr is a professor at the Institute for Computer Engineering of the Department for Electrical Engineering and Communication Technology at Darmstadt University of Technology, where he is head of the the Real-Time Systems Lab. You can contact him at `andy.schuerr@es.tu-darmstadt.de`.

Acknowledgments This work has been funded by the German Research Foundation (DFG) as part of the A1 subproject within the Collaborative Research Center (CRC) 1053 – MAKI.