Electronic Communications of the EASST
Volume 29 (2010)

Proceedings of the Ninth International Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT 2010)

Stochastic Graph Transformation with Regions

Paolo Torrini, Reiko Heckel, István Ráth and Gábor Bergmann

15 pages

Guest Editors: Jochen Küster, Emilio Tuosto
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Stochastic Graph Transformation with Regions

Paolo Torrini[1], Reiko Heckel[2], István Ráth[3] and Gábor Bergmann[4]

[1] pt95@mcs.le.ac.uk
[2] reiko@mcs.le.ac.uk
University of Leicester

[3] rath@mit.bme.hu
[4] bergmann@mit.bme.hu
Budapest University of Technology and Economics

Abstract: Graph transformation can be used to implement stochastic simulation of dynamic systems based on semi-Markov processes, extending the standard approach based on Markov chains. The result is a discrete event system, where states are graphs, and events are rule matches associated to general distributions, rather than just exponential ones. We present an extension of this model, by introducing a hierarchical notion of event location, allowing for stochastic dependence of higher-level events on lower-level ones.

Keywords: Graph Transformation, Stochastic Simulation, Topology

1 Introduction

Graph transformation combines the idea of graphs as a universal modelling paradigm with a rule-based approach to specify the evolution of systems [KK96]. Behaviour can be modelled in terms of labelled transition systems, where states are graphs and rule applications represent transitions. A discrete event system can be generally obtained by interpreting rule matches as events. Hierarchical graphs can be used to keep into account the spatial structure of graphs in terms of topological grouping, with advantages that have been underlined from the point of view of modelling and verification [BL09].

Stochastic graph transformation is applicable to probabilistic analysis and stochastic validation of graph-based modelling. Stochastic simulation can be particularly useful as validation technique when systems are too complex to be model-checked. It can be implemented relying on a discrete event system approach [CL08]. Transitions are labelled by scheduling times, randomly chosen according to given probability distributions — thus replacing stochastic determinism for indeterminism in the models.

A simple form of stochastic graph transformation can be obtained by associating rule names with exponential distributions [HLM06]. The associated Markov-chain analysis has been applied to integrated modelling of architectural reconfiguration and non-functional aspects of network models [Hec05]. However, this approach has some limits. Exponential distributions can express well the relative speed of processes, but are less than suited to describe phenomena characterised by mean and deviation. Generalised stochastic graph transformation can answer this problem, allowing for general distributions to be associated with rule names [KL07] and more generally

with rule matches [HT10, KTH09]. In the latter case, assignment of probability distributions to events may depend e.g. on attributes of match elements. Generalised semi-Markov processes provide the discrete event semantics for such systems.

In reality, events can often be described at different levels of spatial and causal detail. The expression of causal dependency in graph transformation depends solely on rules and their matching. On the other hand, in the approaches that we have considered up to now, each event is treated as stochastically independent from any other with respect to the assignment of probability distributions. Stochastic dependency on global variables and derived attributes has been considered [HT10]. Even allowing that, it is not generally possible to express in a direct way e.g. that the probability of a certain event depends on other events. This can make it particularly hard to model aspects that involve correlation between different levels of description, as in the case of geographic and biochemical systems, where information is usually found at different levels of spatial granularity [TSB02].

In this paper we propose an approach based on hierarchical graphs in order to introduce localisation and granularity of events, we define a notion of structured stochastic simulation allowing us to express stochastic dependency of higher-level events on lower-level ones, and we provide a semantics for it in terms of discrete event systems. Regardless of the specific approach, a major stumbling block in the implementation of stochastic simulation based on graph transformation is the need to compute all the matches at each step. This is hard in principle — the subgraph homomorphism problem is known to be NP-complete, though feasible in many cases of interest. However, the cost of recomputing can be prohibitive. For this reason, we rely on incremental pattern matching based on a RETE-style algorithm as implemented in VIATRA [BHRV08] (a model transformation plugin of Eclipse). In [THR10] we presented GRASS, a tool that extends VIATRA with a stochastic simulation control based on the SSJ libraries [LMV02]. By using a decoupled notion of graph hierarchy, it should be possible to implement hierarchical stochastic simulation in VIATRA/GRASS.

1.1 Hierarchical extensions

Hierarchy in graph models can be used to introduce a notion of topological grouping on model elements. Grouping information can be represented as a hierarchy graph, as distinct from the underlying graph, relying on a decoupled approach [BKK05]. In the case of bigraphs, the approach is to pair *place* graphs and *link* graphs, together with a specific notion of matching [Mil08]. Here we use topology to localise events, rather than elements, relying on a generic notion of rule matching. A model consists of an underlying graph coupled with a place graph, where the latter is a directed acyclic graph (*dag*) from which the hierarchy arises, as partial order ($\leq$). Topological grouping arises from rule matching through the hierarchy. Nodes in the *dag* are places and edges represent containment between places (*hierarchical containment*). The two graphs are linked together by containment edges (*coupling containment*) that map underlying graph nodes to places. Regions are defined as downward-closed sets with respect to the hierarchy, i.e. closed sets in the corresponding order topology [TSB02].

From the stochastic point of view, we use hierarchy to let lower-level events affect the assignment of probability distributions to higher-level ones. In particular, we allow for the distribution assigned to an event to depend (1) on the enabling of other events, and more generally on the

number of enabled matches of a certain type — what we call a *density* measure; (2) on the scheduling of other events — what we call an *activity* measure. In this way, we expect to be able to perform more sensitive stochastic analysis without resorting to making models and reachability analysis more complex.

In fact, density measures boil down to counting matches, therefore they could be handled by introducing additional attributes in a flat model — though this would mean making the model more complex. Activity measures are trickier, as in principle they might lead to circular dependencies. This is avoided by requiring that stochastic dependency between events as well as event time scheduling comply with the hierarchy. Therefore, no event may depend on higher level ones, and at each step of the simulation, the time scheduling of lower-level events takes place before that of higher-level ones.

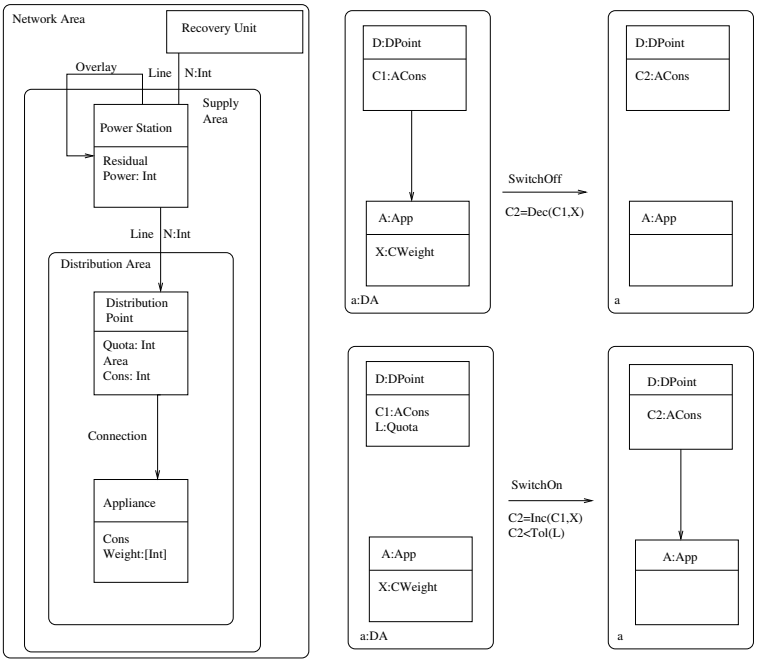
2 Example

We model a power grid as example in which higher-level events may depend stochastically on large numbers of lower-level ones. Each power source serves a number of distribution points, by allocating power quotas in a reconfigurable way. Appliances can be added to and removed from a distribution area, and they can be connected to and disconnected from distribution points, determining the level of consumption, which must remain within a tolerance of the quota. A power failure may occur when the quota is overstepped. A failure determines the disruption of the distribution point, with consequent loss of data, and it forces the intervention of a recovery unit. Actual reconfiguration is carried out following optimisation criteria that can be reflected stochastically in the application of the rule.

The model is based on the SPO approach, and uses typed graphs with attributes. A power station is connected to each of the distribution points by power lines denoted by multi-edges, i.e. sets of parallel edges represented as a single edge with an integer value. A station can reconfigure the capacity of each power line depending on the available power and the distribution area consumption — this takes place by changing the number of line edges, also updating residual power and local quota.

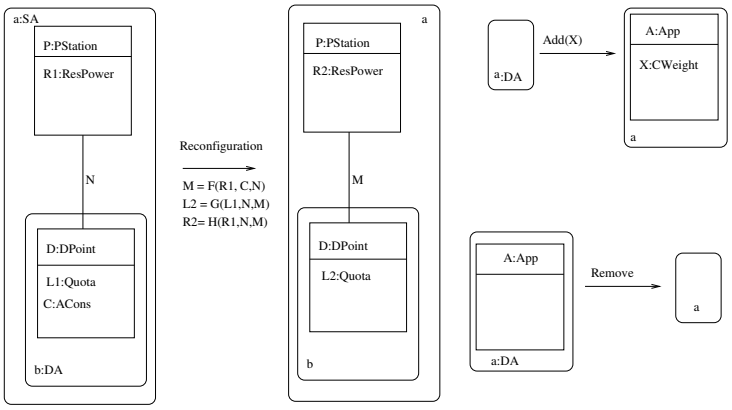
The spatial structure of the model is quite simple — there are three types of places: the *network area*, a *supply area* for each station, and a *distribution area* for each service point. Each place is represented as a rounded box. The hierarchy order $\leq$ is represented as containment (larger boxes are places higher up in the order, therefore associated with higher-level elements). The coupling order is also represented as containment in an obvious way — each underlying graph node (a square box) being coupled only with the smallest place box it is contained in. In this example, the place graph is a tree. The notation could be easily extended to the *dag* case, by associating places to intersections. The symbols $Dec, Inc, Tol, Add, F, G, H, P$ in the figures stand for given functions.

The distinction between higher-level and lower-level events here is comparatively straightforward. The former ones are those associated with reconfigurations, failures and recoveries, and located in regions generated by supply areas. The latter ones are associated with adding, removing, switching on and off appliances, and are located in regions generated by distribution areas. From the stochastic point of view, actions that depend heavily on external aspects, such



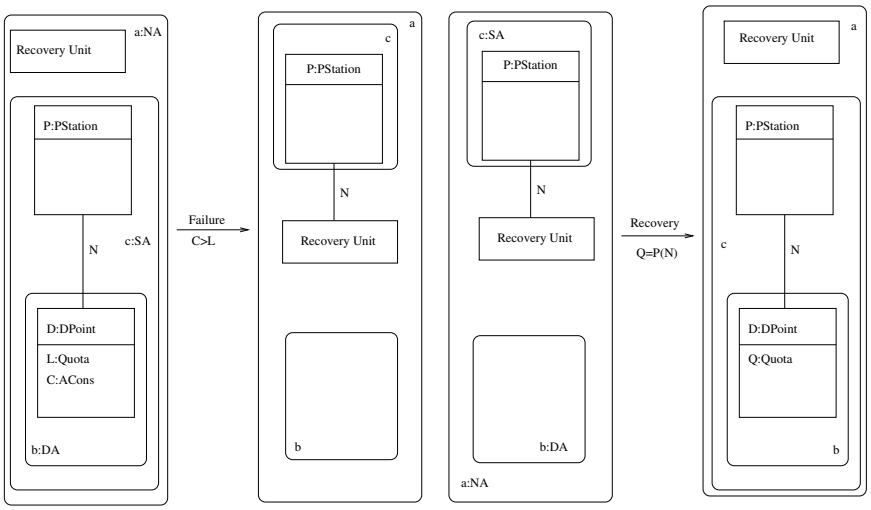
(a) Type graph, Switch-on/off

Figure 1: Type graph, Switch-on/off



(a) Reconfiguration, Add, Remove

Figure 2: Reconfiguration, Add, Remove



(a) Failure, Recovery

Figure 3: Failure, Recovery

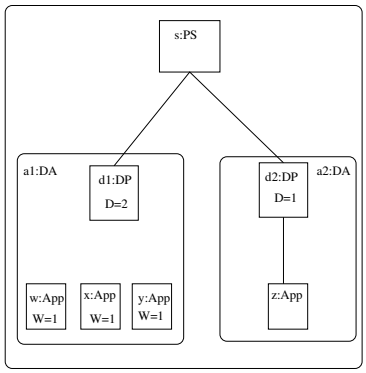
as adding appliances, switching on and failure, may be assigned exponential distributions. Actions more plausibly associated with mean values, such as switching off and recovery, may be more naturally associated with normal distributions. Crucially, reconfiguration can dependent stochastically on the context.

In the example, there are two possible matches for the reconfiguration rule — one with $a1$ and the other with $a2$ as distribution areas. In order to model stochastically a "smart" reconfiguration strategy, one could make the probability of application inversely dependant on the difference between quota and area consumption (a derived attribute associated with distribution points and denoted by D in the picture). However, if that is to be the only criteria, here there is little chance of modelling a high quality of service without changing the model. Given a high rate of switching on against a low one of appliance addition, the area $a1$ is more at risk than $a2$, in spite of the higher D value. This risk is essentially associated with the number of matches for *switch_on* in $a1$, and further than that — with their scheduling times.

Of course it would be possible to retain information about the number of appliances in an area explicitly, by adding an attribute — however, apart from the need to extend the model, this way of capturing the density measure would not be the most natural in this case, as it would not belong to the service point. Moreover, it is difficult to think of a similar way to capture an activity measure. On the other hand, the knowledge embedded in the reconfiguration strategy might be based on estimates rather than precise data. Therefore, modelling it in terms of implicit stochastic dependence seems realistic.

3 Stochastic Graph Transformation

Stochastic graph transformation for semi-Markov process modelling requires us to track matches through transformation. Incremental pattern matching is indeed based on tracking partial



(a) Example

Figure 4: Example

matches. Here we provide a general definition of typed graph transformation that supports tracking with respect to a generic approach (although the running example is based on SPO), allowing for node type inheritance and negative application conditions. We then extend the notion, by endowing graphs with hierarchy and derived topological structure.

3.1 Graph transformation

In existing axiomatic descriptions of graph transformation [KK96], a graph transformation approach is given by a class of graphs $\mathscr{G}$, a class of rules $\mathscr{R}$, and a family of binary relations $\Rightarrow_r \subseteq \mathscr{G} \times \mathscr{G}$ representing transformations by rules $r \in \mathscr{R}$. Here we assume that each rule r is associated to a left hand-side graph L_r and a set of negative application conditions N_r. This notion can be further refined by introducing a definition of rule match depending on a given approach (including SPO and DPO). In the following, we will sometimes use a syntax for function definitions with dependent types [Bar92] in a comparatively informal way, in order to specify functions that we assume to be implementable, by writing $\Pi x \in \alpha.\beta$ rather than $\alpha \to \beta$ when $x \in \alpha$ and β depends on x.

Basically, a graph is a triple $G = \langle Nodes_G, Edges_G, asg_G \rangle$, where for $x \in Nodes_G$, $asg_G(x) = \langle y,z \rangle$ with $y,z \in Nodes_G$. At an abstract level, a partial match of a rule r in a graph G can be represented as a triple $m = \langle r,g,c \rangle$, where $r = rule(m)$, $g = SG(m)$ is a partial graph morphism from L_r into G, and $c = AC(m) \subseteq N_r$ is the set of application conditions that are satisfied. We denote the graph elements of the match, i.e. the image of m, by $EL(m)$. We say that a match m is *valid* when $SG(m)$ is total and $AC(m) = N_r$. We denote by $M_{r,G}$ the set of the partial matches of r in G. $M_{r,G}$ is ordered by a relation $\sqsubseteq_{r,G}$, component-wise defined as subgraph and subset relation on the SG and AC components of the matches, respectively. We define the set of all partial matches in a graph G as $M_G = \bigcup_{r \in R} M_{r,G}$, and by $M_G^v \subseteq M_G$ the set of all those that are valid.

Def. 1 We define a function $\Rightarrow: \Pi G \in \mathscr{G}.M_G^v \to \mathscr{G}$. We write $G \Rightarrow_m H$ for $\Rightarrow(G)(m) = H$, and we say that this is the *transformation step* determined by the application of rule $rule(m)$ to match $SG(m)$ in graph G.

Notice that transformation steps correspond to a function that is partially defined with respect to the set of all matches. The functional requirement captures the idea that rule application is well-defined and deterministic once a valid match m is found in G. This is needed, in order to guarantee that matches form a proper set, unlike in more abstract presentations [EEPT06]. Also notice that, however, we implicitly allow for a global renaming action associated with a transformation.

Def. 2 A graph transformation system (GTS) $\mathbf{G} = \langle R, G_0 \rangle$ consists of a set $R \subseteq \mathscr{R}$ of rules and an initial graph $G_0 \in \mathscr{G}$. A transformation in $\mathbf{G}$ is a sequence of rule applications $G_0 \Rightarrow_{m_1} G_1 \Rightarrow_{m_2} \cdots \Rightarrow_{m_n} G_n$ using rules in $\mathbf{G}$ with all graphs $G_i \in \mathscr{G}$.

Assuming finite graphs and an enumerable set of rules, the graphs that are reachable from G_0 by a finite sequence of transformation steps form a set, denoted by $\mathscr{L}_\mathbf{G}$, and so do the partial matches over the reachable graphs, denoted by $M_\mathbf{G}$.

Even without fixing the approach, we can say that in general, a transformation step $t = G \Rightarrow_m H$ is associated with a set Del_t of all the graph elements that are deleted or modified by t, and with a set Cre_t of all the graph elements that are created by t. Correspondingly, the partial matches that are destroyed form a set $D_{G,t} \subseteq M_G$, those that are created form a set $C_{H,t} \subseteq M_H$, and those that are preserved are $M_{|t} = M_G \backslash D_{G,t}$. From the deterministic assumption, $M_H = \sigma_t(M_{|t}) \cup C_{G,t}$ follows, where σ_t is the renaming induced by t (assuming the name spaces are disjoint).

We are interested in defining a notion of persistent match, by identifying matches that are preserved through transformation. In particular, from the point of view of stochastic models, given $m_1 \in \sigma_t(D_{G,t}), m_2 \in D_{G,t}$ and $m_2 = \sigma_t(m_1)$, we want to identify m_1 and m_2 when they are both valid with respect to the same rule r. This can be generalised to partial matches. In [KTH09, KL07] we relied on a conservative naming policy — here we adopt a looser one altogether, though still defining a persistent match as equivalence class.

In order to abstract from renaming, we define, for $n_1 \in M_{R,G'}, n_2 \in M_{R,G''}$, the symmetric relation $n_1 =_a n_2$ that holds whenever for all transformation steps t, if $t = G' \Rightarrow_m G''$ and $n_1 \in M_{|t}$, then $n_2 = \sigma_t(n_1)$, and if $t = G'' \Rightarrow_m G'$ and $n_2 \in M_{|t}$ then $n_1 = \sigma_t(n_2)$. We can then define the transitive closure using the least fixpoint operator (μ)

$$n_1 \equiv n_2 =_{df} \mu E.(n_1 = n_2) \vee (\exists n_3.E(n_1, n_3) \wedge n_3 =_a n_2)$$

It is a matter of routine to show that $\equiv$ is indeed an equivalence relation. The persistent matches over the set of the reachable graphs in $\mathbf{G}$ can therefore be defined as the quotient class $\mathscr{M}_\mathbf{G} = M_\mathbf{G} / \equiv$. We call *event* a persistent valid match, and we denote with $\mathscr{E}_\mathbf{G}$ the corresponding set of events. We define $\mathscr{M}_G = \{[m] | m \in M_G\}$, and $\mathscr{E}_G = \{[m] | m \in M_G^v\}$. This in general will allow us to lift definitions from valid matches to events.

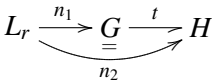


Figure 5: persistence of matches

3.2 Hierarchical structure

We use hierarchical graphs in the decoupled sense [BKK05], i.e. we define a hierarchical graph as graph that includes the underlying graph as well as a directed acyclic graph representing the hierarchical structure.

Def. 3 A hierarchical graph is a graph G in which there is a distinguished *dag* $P_G \subset G$ called the *place graph*, where the nodes of P_G are the *places*;
(a) the edges that link nodes in $G \backslash P_G$ to (non-empty) places, called *C-edges*, express coupling containment and form a distinguished set $CEdges_G \subset Edges_G$;
(b) the edges connecting places together (*H-edges*) express hierarchical containment.
Moreover, the following conditions must be satisfied:
(1) $\langle P_G, \leq_G \rangle$ is a partial order (*hierarchy* of G), where $\leq_G$ is the reflexive-transitive closure of hierarchical containment.
(2) $<_G$ is downward well-founded (i.e. there are no infinite descending chains).
(3) $\leq_G$ has finite degree (i.e. it is finitely branching).
In addition, (i) the *underlying graph* of G is defined as the largest subgraph of G which does not contain either nodes or C-edges — this may be expressed set-theoretically as $U_G = (G \backslash P_G) \backslash CEdges_G$;
(ii) for $X \subseteq Node_G$, $loc_G(X)$ denotes the finite set of the places associated to any element of X by C-edges (the *location* of X); we also write $loc_G(x)$ for $loc_G(\{x\})$, and $loc_G(m)$ for $loc_G(EL_G(m) \cap Nodes_G)$ with $m \in \mathscr{M}_G$.
We say that G is *complete* whenever for each node x in U_G, $loc_G(x) \neq \emptyset$.
We say that G is *grounded* whenever for each node x in U_G, $loc_G(x)$ contains at most one element.
We denote by $\mathscr{G}^h$ the class of the hierarchical graphs, by $\mathscr{G}^c$ the class of the complete ones, by $\mathscr{G}^g$ that of the grounded ones, and by $\mathscr{G}^{cg}$ that of the complete grounded ones.

Clearly, P_G may be a finitely branching tree. In practice, P_G is going to be mostly finite — indeed, assuming $\leq_G$ does not contain infinite ascending chains and has a finite number of minimal elements, finiteness follows, using Koenig's lemma.

Any partial order $\leq$ can be associated to an Alexandroff topology where the downward-closed sets are the closed sets and $\leq$ coincides with the specialisation order [Vic89]. This is too fine-grained though, as there is no need to consider downward-closed sets which are not generated by matches, and we prefer to take nodes that share a place to the same region. Therefore we define the following.

Def. 4 Given a partial match $m \in \mathscr{M}_G$, we say that the *graph region* of m, denoted by $greg_G(m)$, is the smallest subgraph $J \subseteq G$ such that
(1) set-theoretically, $SG(m) \subseteq J$, $loc_G(m) \subseteq J$, and all the C-edges and H-edges in G between elements of J are in J.
(2) J is downward-closed with respect to $\leq_G$, i.e. for w, z places in P_G, if $w \in J$ and $z \leq_G w$ then $z \in J$
(3) for each node x in G, if $loc_G(x) \cap J \neq \emptyset$ then $loc_G(x) \subseteq J$.
The *abstract region* of m is the place subgraph $reg_G(m) = greg_G(m) \cap P_G$.

We denote by Reg_G the set of the abstract regions defined on $\mathscr{M}_G$.
For $m,m' \in \mathscr{M}_G$, $p \in Reg_G$, we say that m is *located* in p iff $reg_G(m) \subseteq p$. We say that m is *lower-level* with respect to m' (resp. m' is *higher-level* wrt m) and we define $m \sqsubset_G m'$ iff $reg_G(m) \subset reg_G(m')$.

Notice that $m \subseteq m'$ follows from $EL(m) \subseteq EL(m')$ for any $m,m' \in \mathscr{M}_G$. Set-theoretically, it is not difficult to see that Reg_G is closed with respect to arbitrary intersection (condition 3 helps). Adding empty and global region, we may regard Reg_G as basis of an Alexandroff topology, where the specialisation order is monotonic with respect to $\leq_G$. From the downward well-foundedness of $<_G$, it follows that $\sqsubset_G$ is similarly well-founded, i.e. it has no infinite descending chains. Abstract regions can therefore be used to order events.

We can now introduce a notion of hierarchical transformation system, consistently with the generic approach, along the lines of the definitions 1, 2, by adapting them to classes of hierarchical graphs. We take $x = h \mid c \mid g \mid cg$.

Def. 5 A *hierarchical transformation step function* is a function $\Rightarrow^x$: $\Pi G \in \mathscr{G}^x.M_G^y \to \mathscr{G}^x$ such that (by the notation of def. 1), $G \Rightarrow^x_m H$ iff $G \Rightarrow_m H$, provided $G,H \in \mathscr{G}^x$.
A hierarchical graph transformation system (HGTS) $\mathbf{G} = \langle R, G_0 \rangle$ consists of a set $R \subseteq \mathscr{R}$ of rules and an initial graph $G_0 \in \mathscr{G}^x$.
A transformation in $\mathbf{G}$ is a sequence of rule applications using rules in $\mathbf{G}$ and applying them through $\Rightarrow^x$.

In case of $x = h$, the definitions are clearly generalisations of those for $\mathscr{G}$. The running example is meant to be a case of $x = cg$. Notice that in every case, the notion of match is implicitly assumed to be the most general — this can be stated by saying that for every match $m \in M_{\mathbf{G}}$, $SG(m) \in \mathscr{G}^h$. However, in order to keep $\mathscr{E}_G$ finite, we assume as fixed the names of empty places that are preserved by the transformation.

When the system is based on the SPO approach — as in the running example, each rule can be defined in terms of two components — one defined on the underlying graph, the other on the place graph. Figure 6 provides with a diagrammatic illustration, where 1–5 are pushouts and 6–9 are pullbacks. Notice that regions can be used to wrap the side-effects of SPO rules — e.g. the *Failure* rule in the running example, where all the line edges which are deleted are in fact between nodes that are included in the supply area — thus generally allowing for more modular transformations.

3.3 Stochastic Modelling

Generalised stochastic graph transformation has been defined in [HT10, THR10, KTH09], by associating events with general distributions, each of them expressed as a cumulative distribution function (*cdf*) — i.e. a function from real numbers to probability values. Here we denote by $Dist(e) \subseteq Real \to [0,1]$ the type of the *cdf* assigned to event e.

Def. 6 A generalised stochastic graph transformation system (GSGTS) is a structure $\mathbf{S} = \langle \mathbf{G}, \Delta_0 \rangle$ where $\mathbf{G}$ is a GTS, $\Delta_0 : \Pi e \in \mathscr{E}_{\mathbf{G}}.Dist(e)$ is a distribution assignment, which associates with every event a *cdf*.

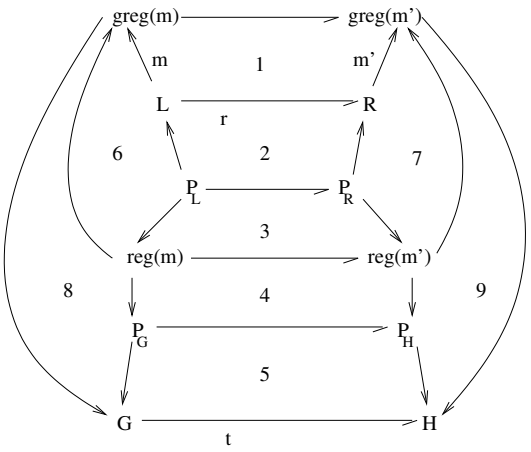


Figure 6: hierarchical transformation

The behaviour of a GSGTS can be described as a stochastic process over continuous time, where reachable graphs form the state space and the application of transformation rules defines state transitions as instantaneous events. More precisely, a rule enabled by a match defines an event associated with an independent random variable (*timer*, or scheduling time) which represents the time expected to elapse (*scheduled delay*) before the rule is applied to the match. As the simulation is executed, the timer is randomly set according to the static specification provided by the *cdf* of the corresponding event — in the implementation, this involves a call to a pseudo-random number generator (RNG).

We intend to extend the definition of stochastic graph transformation in order to make it possible to express the dependency of the probability distribution of an event e on other events, and more precisely on properties of the graph, such as the number of events of a certain type in a certain region associated to e, or the average scheduled delay for lower-level events of a certain type — e.g. the dependency of the *cdf* of *Reconfiguration* events on the number and the scheduled delay of *Add* and *SwitchOn* events in the relevant distribution area, given in the running example. Syntactically, we also allow for rule constructors over finite domains, i.e. $c : X \to R$ with X finite, representing a set of rules $r_1 = c(x_1), \ldots, r_k = c(x_k)$, where k is the cardinality of X — e.g. *Add(X)* in the example.

We now consider stochastic simulation from the point of view of a HGTS $\mathbf{G}$. For each $G \in \mathscr{L}_{\mathbf{G}}$, we can assume to have a scheduling function

$$sched_G : \Pi e \in \mathscr{E}_G.Dist(e) \to Time(e)$$

which assigns a value to each timer, given its *cdf* — where $Time(e) \subseteq Real$ is a subtype of the reals that represents a random variable. We can also assume to have an event counting function

$$count_G : \Pi p \in Reg_G.\Pi r \in R.Num(r,p)$$

where $Num(r,p) \subseteq Nat$ is a subtype of the naturals that represents the number of matches of rule r in region p.

Once we assume that for each state G, the scheduling of delays follows the granularity order $\sqsubset_G$, we can allow for the *cdf* of an event $e \in \mathscr{E}_G$ to depend
(1) on the general number of events in G, i.e. on $\{count_G(r,p) | r \in R, p \in Reg_G\}$, and as a special case, on the number of events located in the same region, i.e. $\{count_G(r, reg_G(e)) | r \in R\}$ (*local density*);
(2) on the scheduled delays of lower-level events, i.e. on $\{sched_G(e') | e' \sqsubset e\}$ (*local activity*).

We now show how the distribution assignment can be functionally defined, in order to make the above possible. We can associate local density to *local event counting*, specified as function of type

$$LDens_G =_{df} \Pi e \in \mathscr{E}_G.\Pi r \in R.Num(r, reg_G(e))$$

We can associate local activity to a notion of *local delay scheduling*, specified as a function of type

$$LAct_G =_{df} \Pi e_1 e_2 \in \mathscr{E}_G.e_2 \sqsubset_G e_1 \rightarrow Time(e_2)$$

We can then introduce a notion of *abstract distribution assignment* δ that depends on both, specified as follows

$$\delta : \Pi G \in \mathscr{L}_{\mathbf{G}}.\Pi e \in \mathscr{E}_G.(LDens_G(e) \times LAct_G(e)) \rightarrow Dist(e)$$

Partial scheduling $\pi_G : \Pi e \in \mathscr{E}_G.LDens_G(e) \times LAct_G(e)$ can now be defined by recursion (assuming δ is given)

$$\pi_G =_{df} \mu f.\lambda e_1.(\lambda r.count_G(r, reg_G(e_1)),\ \lambda e_2.\ \text{if } e_2 \sqsubset_G e_1 \text{ then } (sched_G(e_2)(\delta(G)(e_2)(f(e_2)))))$$

Finally, the distribution assignment $\Delta : \Pi G \in \mathscr{L}_{\mathbf{G}}.\Pi e \in \mathscr{E}_G.Dist(e)$ can be defined as

$$\Delta =_{df} \lambda G.\lambda e.\delta(G)(e)(\pi_G(e))$$

This shows that Δ can be defined recursively, given $sched, count, \delta$, and relying on the well-foundedness of $\sqsubset$.

Def. 7 A *stochastic hierarchical graph transformation system* (SHGTS) is a structure $\mathbf{H} = \langle \mathbf{G}, \delta \rangle$, where $\mathbf{G}$ is a HGTS and δ is a function specified as above, so that $\Delta : \Pi G.\Pi e.Dist(e)$ can be defined as the function that assigns a continuous *cdf* to each event for each reachable graph.

4 Stochastic Simulation

We define an operational interpretation of SHGTS in terms of semi-Markov processes, following an approach already used in [THR10, KTH09, KL07]. We rely on a representation of stochastic processes as discrete event systems [CL08], and define a translation of SHGTS into them.

Semi-Markov processes are continuous-time stochastic processes in which the embedded jump chain is a Markov chain and the inter-event times are associated with general distribution functions. This means that events are independent of past states, but may depend on the time spent in the current one. They are a generalisation of continuous-time Markov processes, which allows only for exponential distributions. More formally, a semi-Markov process can be defined as a process generated by a *generalised semi-Markov scheme* (GSMS) [DK05]. Here we need to define a structure that is syntactically more general than a GSMS, insofar as we need to keep hierarchy and delay scheduling order into account.

Def. 8 A *hierarchical semi-Markov scheme* (HSMS) is here a structure

$$\begin{aligned}\mathscr{P} \;=\; & \langle\, Z,\, E,\, enabled : Z \to \wp(E),\, \sqsubseteq^S : Z \to (E \times E) \to Boolean, \\ & new : Z \times E \to Z,\, cdfAsg : Z \to E \to Real \to [0,1],\, s_0 : Z \,\rangle\end{aligned}$$

where Z is a set of system states; E is a set of timed events; *enabled* is the activation function, so that $enabled(s)$ is the finite set of active events associated with state s; *new* is a partial function depending on states and events that represent transitions; *cdfAsg* is the distribution assignment, so that $cdfAsg(s)(e)$ gives a *cdf* of the scheduled delay of e at state s; $\sqsubseteq^S(s)$ is a well-founded order (*schedule-making order*) on the enabled events at state s; and s_0 is the initial state.

Def. 9 Given a SHGTS $\mathbf{H} = \langle R, G_0, \delta \rangle$, we define its translation to an HSMS $\mathscr{P}_{\mathbf{H}}$ as follows:

1. $Z = \mathscr{L}(\langle R, G_0 \rangle)$ i.e. set of graphs reachable from G_0 by rules in R
2. $E = \mathscr{E}_{\mathbf{G}}$ i.e. the set of possible events for $\mathbf{G} = \langle R, G_0 \rangle$
3. $enabled(G) = E_G$ i.e. set of all events enabled in graph G
4. $new(G,e) = H$ iff $G \Rightarrow_e H$ i.e. transition is defined as transformation
5. $cdfAsg(G)(e) \;=\; \Delta(G)(e)$
6. $\sqsubseteq^S(G) \;= \sqsubseteq_G$
7. $s_0 \;=\; G_0$

This embedding can be used as static framework for the definition of a simulation algorithm that is adequate with respect to system runs, in the sense that there is a one-to-one correspondence between the runs of the original SHGTS and those of the resulting HSMS, and therefore correct and complete with respect to reachability. The algorithm, based on the general scheduling scheme given in [CL08], can be described as follows.

1. Initial step
   (a) The simulation time is initialised to 0.
   (b) The set of the enabled events $A = enabled(s_0)$ is computed.
   (c) The schedule-making order $\sqsubseteq^S(s_0)$ is computed.

(d) For each event $e \in A$, a scheduling time t_e is determined by RNG depending on the probability distribution function $cdfAsg(e)(s_0)$;

(e) The enabled events with their scheduled times are collected in the event list $l_{s_0} = \{(e,t_e)|e \in A\}$ ordered by time values.

2. For each successive step — given the current state $s \in Z$ and the associated event list $l_s = \{(e,t)|e \in active(s)\}$

(a) the first element $k = (e,t)$ is removed from l_s;

(b) the simulation time t_S is updated by increasing it to t;

(c) the new state s' is computed as $s' = new(s,e)$;

(d) the list $m_{s'}$ of the surviving events is computed, by removing from l_s all the disabled elements, i.e. all the elements (z,x) of l_s such that $z \notin enabled(s')$;

(e) The schedule-making order $\sqsubseteq^S(s')$ is computed.

(f) a list $n_{s'}$ of the newly enabled events is built, containing a single element (z,t_z) for each event z such that $z \in enabled(s') \backslash enabled(s)$ and is scheduled at time $t_z = t_S + d_z$, where d_z is the random delay value given by RNG depending on the distribution function $cdfAsg(s')(z)$;

(g) the new event list $l_{s'}$ is obtained by reordering the concatenation of $m_{s'}$ and $n_{s'}$ with respect to time values

Part of the complexity of the algorithm is hidden in the recursive definition of *cdfAsg*, which means that time scheduling takes place following the schedule-making order $\sqsubset^S$, associated to $\sqsubset_G$ by the translation. The translation and the assumptions on the hierarchy $\leq$ also guarantee that $\sqsubset^S(s)$ is well-founded. Essentially, the algorithm relies on graph transformation with persistent matches, on functions to count events, and on ordered delay scheduling implemented as calls to RNG.

5 Further work

Expressing stochastic dependencies associated with density and activity measures — as in the example — can be useful to model situations in which specific events depend on large numbers of co-located ones, as in describing biochemical processes at different levels of detail (e.g. molecular and cellular). Graph transformation can be good at tracking individual processes — however, there are aspects that can be modelled more efficiently in terms of mass effect and differential equations [Car08]. Therefore, the general capability of expressing presence of reactants and reactions in a region can be useful.

From the point of view of an implementation, applying few high-level rules rather than many low-level ones could ease the cost of updating incremental data-structures. As presently implemented in GRASS [THR10], scheduling is carried out independently of the RETE network. This might make hierarchical scheduling comparatively expensive. Hierarchical information could be handled more efficiently by introducing a further level of incrementality based on a *live transformation* approach [RBOV08], i.e. by continuously maintaining spatial information as part of

the transformation context, so that changes to the spatial structure can be instantly mapped to the underlying graph.

6 Conclusion

Stochastic simulation is a promising field of application for graph transformation techniques. We have argued that structured graphs can be useful for stochastic simulation. We have focussed on probabilistic dependency of events on global and local properties, in order to make stochastic modelling more flexible. In particular, we have shown that hierarchy can be used to define a topological order on events, allowing for the introduction of a weak notion of modularity and an increased expressiveness with respect to stochastic simulation. This extension can be embedded in discrete event models of stochastic processes. We think that an implementation of stochastic simulation along this lines can take benefit from an appropriate use of incremental pattern-matching.

References

[Bar92] H. P. Barendregt. Lambda Calculi with Types. In Abramsky et al. (eds.), *Handbook of Logic in Computer Science*. Volume 2, pp. 117–309. Oxford – Clarendon Press, 1992.

[BHRV08] G. Bergmann, A. Horváth, I. Rath, D. Varró. A benchmark evaluation of incremental pattern matching in graph transformation. In *International Conference on Graph Transformation*. LNCS 5214, pp. 396–410. 2008.

[BKK05] G. Busatto, H.-J. Kreowski, S. Kuske. Abstract hierarchical graph transformation. *Mathematical Structures in Computer Science* 15(4):773–819, 2005.

[BL09] R. Bruni, A. Lluch Lafuente. Ten Virtues of Structured Graphs. In *GT-VMT'09*. 2009.

[Car08] L. Cardelli. Artificial Biochemistry. In Springer (ed.), *Algorithmic Bioprocesses*. LNCS, 2008.

[CL08] C. G. Cassandras, S. Lafortune. *Introduction to discrete event systems*. Kluwer, 2008.

[DK05] P. R. D'Argenio, J.-P. Katoen. A theory of stochastic systems part I: Stochastic automata. *Inf. Comput.* 203(1):1–38, 2005.

[EEPT06] H. Ehrig, K. Ehrig, U. Prange, G. Taentzer. *Fundamentals of Algebraic Graph Transformation (Monographs in Theoretical Computer Science. An EATCS Series)*. Springer, 2006.

[Hec05] R. Heckel. Stochastic Analysis of Graph Transformation Systems: A Case Study in P2P Networks. In *Proc. Intl. Colloquium on Theoretical Aspects of Computing (ICTAC'05)*. LNCS 3722, pp. 53–69. Springer-Verlag, 2005.

[HLM06] R. Heckel, G. Lajios, S. Menge. Stochastic graph transformation systems. *Fundamenta Informaticae* 72:1–22, 2006.

[HT10] R. Heckel, P. Torrini. Stochastic Modelling and Simulation of Mobile Systems. 2010. to be published.

[KK96] H.-J. Kreowski, S. Kuske. On the Interleaving Semantics of Transformation Units - A Step into GRACE. In Cuny (ed.), *Graph Grammars and their Application to Computer Science*. Pp. 89 – 106. Springer, 1996.

[KL07] P. Kosiuczenko, G. Lajios. Simulation of generalised semi-Markov processes based on graph transformation systems. *Electronic Notes in Theoretical Computer Science* 175:73–86, 2007.

[KTH09] A. Khan, P. Torrini, R. Heckel. Model-based Simulation of VoIP Network Reconfigurations using Graph Transformation Systems. In Corradini and Tuosto (eds.), *Intl. Conf. on Graph Transformation (ICGT) 2008 - Doctoral Symposium*. Electronic Communications of the EASST 16. 2009.

[LMV02] P. L. L'Ecuyer, L. Meliani, J. Vaucher. SSJ: a framework for stochastic simulation in Java. In *Proceedings of the 2002 Winter Simulation Conference*. Pp. 234–242. 2002.

[Mil08] R. Milner. Bigraphs and Their Algebra. *Electr. Notes Theor. Comput. Sci.* 209:5–19, 2008.

[RBOV08] I. Rath, G. Bergmann, A. Okrős, D. Varró. Live model transformations driven by incremental pattern matching. In *ICMT'08*. LNCS 5063, pp. 107–121. Springer, 2008.

[THR10] P. Torrini, R. Heckel, I. Ráth. Stochastic Simulation of Graph Transformation Systems. In *FASE'10*. Pp. 154–157. 2010.

[TSB02] P. Torrini, J. G. Stell, B. Bennett. Mereotopology in second-order and modal extensions of intuitionistic propositional logic. *Journal of Applied Non-Classical Logic* 12(3–4):490–525, 2002.

[Vic89] S. Vickers. *Topology via logic*. Cambridge University Press, 1989.