A Stochastic Extension of a Behavioural Subset of UML Statechart Diagrams

Stefania Gnesi  
Consiglio Nazionale delle Ricerche  
Istituto di Elaborazione dell’Informazione  
Via Alfieri 1, S. Cataldo, Pisa, ITALY  
gnesi@iei.pi.cnr.it

Diego Latella  
Consiglio Nazionale delle Ricerche  
Istituto CNUCE  
Via Alfieri 1, S. Cataldo, Pisa, ITALY  
d.latella@cnuce.cnr.it

Mieke Massink  
Consiglio Nazionale delle Ricerche  
Istituto CNUCE  
Via Alfieri 1, S. Cataldo, Pisa, ITALY  
m.massink@cnuce.cnr.it

Abstract

In this paper we present a stochastically timed extension of UML Statechart Diagrams. The extension is rather simple both from a notational point of view and from a semantics point of view. In particular we enrich a state/transition formal operational semantics we proposed in [14] with random clocks for expressing time values. We do this in an “orthogonal” way, which means that the enriched semantics preserves all the properties of the untimed one. We show, by means of a simple example, how the enriched notation and its semantics can be used for performing quantitative analysis of Stochastic UML Statechart Diagrams models.

1 Introduction

The study of formal methods for the specification, design, and analysis of distributed systems has been an important research topic over the past decade. Initially, the research in this area has concentrated on the dynamic, functional aspects of such systems, like their observable behaviour, control flow, and synchronisation as properties in relative time.

More recently, formal methods for the representation and analysis of functional properties in combination with quantitative aspects of system behaviour have come into focus. They allow the specification of the delay of activities (or, actions) or the probability of actual occurrence of actions.

Nowadays, quantitative analysis, like performance analysis, has a well-recognised position in the design of complex distributed systems. Usually, quantitative models like queueing networks and Markov chains are developed by abstracting from the systems specification used for the qualitative analysis and functional design. Moreover, such an abstraction implies the use of modelling techniques and notations which are completely different from those used for the functional specification. As a matter of fact, quite often, the “abstraction” boils down to a complete redesign of the model, using the functional one as a mere “inspiration”. Obtaining quantitative models from system specifications in this way largely depends on human ingenuity and experience. The increasing complexity and magnitude of systems complicates this task considerably. Therefore, it is more and more necessary to obtain quantitative models in a way which exploits the structure of the system specification at hand.

The Unified Modelling Language (UML) is a graphical modelling language for object-oriented software and systems [5, 17, 18, 19]. It has been specifically designed for visualising, specifying, constructing and documenting several aspects of - or views on - systems. Different diagrams are used for the description of the different views. The UML is a semi-formal language, since its syntax and static semantics (the model elements, their interconnection and well-formedness) are defined precisely, but its dynamic semantics are not specified formally [18].

In our work we focus on UML Statechart Diagrams (UMLSD), which are meant for describing dynamic aspects of system behaviour. In particular we consider a subset...
of the notation for which we have provided a formal semantics in [14] which we later used for automatic verification of UMLSD [7, 13]. We call such a subset a “behavioural” one since it includes all the features of interest for reasoning about behaviour, like concurrency and nondeterminism, but does not include other important features of the notation, like those related to object-orientation. Our operational semantics is based on a hierarchical representation of UMLSD, which greatly facilitates formal reasoning about the notation and its meaning. For a discussion on all the above issues and on the reasons for our choices the interested reader is referred to [13].

In the present paper we propose Stochastic UMLSD (SUMLSD), a stochastic time extension of UMLSD inspired by Stochastic Automata [3, 4], together with their operational semantics. The extension consists in adding a set of clocks to a UMLSD. These clocks can be used as guards for transitions, which can fire only when the related clocks evaluate to zero. Moreover, clocks can be set as soon as a state is entered: the particular value to which a clock is set is given by a random variable with specified distribution function.

The semantic model we use for SUMLSD is that of Stochastic Automata, namely (finite state) automata with clocks set via random variables and tested by transitions. The reason why we chose Stochastic Automata is that they allow the use of non-exponential distributions for the random variables which characterise the time information. Most of the other stochastic extensions to formal models for concurrency allow only exponential distributions. As we shall see in this paper, non-exponential distributions not only are useful because they allow a broader range of behaviours to be modelled, but they are essential in the context of SUMLSD, due to the peculiar parallelism paradigm of statecharts.

Our stochastic semantics are a conservative, "orthogonal", extension of those proposed in [14]. By "orthogonal" here we mean that if one deletes the stochastic information from the semantics of our Stochastic UMLSD, the result is the pure (i.e. untimed) semantics for UMLSD.

Such a property is quite important. In fact it establishes a direct link between the stochastic model and its untimed version. This way confidence on the overall system design and quantitative assessment process is increased. For instance, one can formally prove that a stochastic model of a system, written using SUMLSD, has a pure semantics which is a proper abstraction of the functional model, written in UMLSD. What "a proper abstraction" means depends on the particular behavioural relation one chooses. The literature on concurrency theory provides a rich set of such relations (see for instance [20]). This way a solid and mathematically sound bridge can be established between formal validation techniques and tools on the one hand and quantitative assessment ones on the other, all of them being integrated and used in the same notational framework. We consider such a bridge extremely important for the design of high assurance systems. In particular for the assurance of their functional correctness integrated with their quantitative assessment, and specially as an argument for providing convincing evidence of their safety.

In Sect. 2 we give a quick overview of Stochastic Automata. Sect. 3 is an informal introduction to our stochastic variant of UMLSD. In Sect. 4 Stochastic Hierarchical Automata, an abstract syntax for UML Statechart Diagrams, are introduced; all relevant definitions are presented and the formal operational semantics for SUMLSD is given; moreover, the main correctness result of the paper is presented. In Sect. 5 we briefly discuss possible applications of our approach and in Sect. 6 we draw some conclusions and indicate lines for future research.

2 Stochastic Automata

Stochastic Automata (SA) [3, 4] have been developed to define the semantics of stochastic process algebras. In many stochastic process algebras such as TIPP [11], PEPA [12] and EMPA [2], the semantics are defined by associating a transition with both an action name and a distribution function that determines the stochastic timing of the transition. This association leads to complications in the definition of parallel composition of processes, which typically bring to restricting the allowed distributions to exponential ones only. The SA proposed in [3] do not suffer from such a limitation, thus allowing an unrestricted use of distributions for random variables.

SA are automata with a set of random clocks and a clock setting function that determines for each state, called location, which clocks are set to which value. The transitions of stochastic automata, called edges, are labelled by an action and a finite set of clocks. A stochastic automaton can perform a transition from location $s$ to location $s'$ labelled by action $a$ and clock set $C$ by performing action $a$ as soon as all the clocks in set $C$ have expired. A global time is assumed and all clocks are decreased at the same speed. Immediately after the transition takes place all clocks associated to $s'$ by the clock setting function are randomly set according to their probability distributions. The above informal description is captured by a formal semantics of SA given in terms of Probabilistic Transitions Systems. For the sake of simplicity, here we will refrain from describing such semantics, referring the interested reader to [3].

3 Stochastic UML Statechart Diagrams

UML Statechart Diagrams are a (object-oriented) variant of classical Harel statecharts [8, 9]. The statecharts for-
In our example, we assume an exponential distribution with rate $\lambda$ for the arrivals. Incoming requests are sampled by the system exactly every $N$ time units. Finally, the processing time of a request is exponentially distributed, with rate $\mu$.

In Fig. 1 the above situation is modelled by a composite state, namely System. This state is refined into two concurrent states, namely Arrivals, which models both the sensing and the incoming requests, and Controller. These states are further refined into distinct sequential automata, the initial states of which are $A_1$ and $OFF$ respectively. The distribution functions are associated to the clocks in the “clock definition” part of the diagram - $\exp$ (resp. $K$) stand for exponential (resp. constant) distribution. Their definitions reflect our assumptions on the arrivals, the sampling rate and the processing time. As in the case of SA, when entering a state, the clocks listed in the state are set to the values which are determined by random variables associated to the clocks. Also for SUMLSD, as for SA global time is assumed and all clocks are decreased at the same speed. Finally, the sub-states of a concurrent state are entered at exactly the same time at which such a state is entered, so that all the clocks involved are set at that time.

In our example, both $C_a$ and $C_n$ are given a value when state $A_1$ of Arrivals is entered, while only $C_n$ (resp. $C_a$) gets a value when $A_2$ (resp. $A_3$) is entered. Similarly, $C_s$ is set whenever state $ONa$ of Controller is entered.

“System states” are modelled by configurations, which are sets of states. For instance, the following are possible configurations of our system\footnote{For simplicity, we will often skip mentioning states System, Arrivals and Controller explicitly.}: \{A_1, OFF\}, the initial configuration, and \{A_2, ONa\}.

A transition connects a source to a target state. The transition is labelled by a trigger event, a corresponding action and an optional set of clocks. A transition is enabled and can fire if and only if its source state is in the current configuration, its trigger is offered by the external environment and all clocks which label it evaluate to zero. In this case, if the transition fires, the source state is left, the actions are executed, and the target state is entered. As in UMLSD, the actual states which are entered are sub-states of the target state, when the latter is not basic (i.e. is refined). Symmetrically, also the source state can be a composite one so that the real origin of a transition is indeed a set of sub-states of such a state. In both cases we speak of inter-level transitions. In our running example there are no inter-level transitions, for simplicity.

In general, more than one event may be available in the environment. The UML semantics assumes a dispatcher which selects one event at a time from the environment, modelled as a queue, and offers it to the state machine. In our example, the environment is assumed to be initialised with event $sw$, the trigger of (every transition of) Arrivals, and is used by this state for its communication with the Controller as well as its self-triggering. The environment will always contain one event, so that no assumption needs to be made on the dispatching policy. Starting from the ini-
tial configuration and the initial environment, if no arrival takes place before the first sampling (i.e. as soon as \( C_n \) reaches zero) \( \text{Arrivals} \) moves to state \( A_2 \), sending none to \( \text{Controller} \) and resetting clock \( C_n \), for keeping the sampling rate; the \( \text{Controller} \) reacts by sending a \( \text{sw} \) back for reactivating \( \text{Arrivals} \) and remains in its initial state. If, on the other hand, there are requests (i.e. \( C_a \) reaches zero) before the first sampling, then \( \text{Arrivals} \) moves to state \( A_3 \) and remains there until clock \( C_n \) expires, in case it moves to \( A_2 \) sending \( \text{on} \) to \( \text{Controller} \) and resetting clock \( C_n \). While \( \text{Arrivals} \) is in state \( A_3 \) clock \( C_a \) may expire, thus modelling new incoming requests. Each time \( \text{Arrivals} \) moves (back) to \( A_3 \) clock \( C_a \) is obviously reset and \( \text{sw} \) is sent for self-reactivation. While in state \( A_2 \), a new request makes \( \text{Arrivals} \) move to \( A_3 \) for waiting for the next sampling time. The behaviour of the \( \text{Controller} \) should be clear. Notice that on entering state \( \text{ONa} \), clock \( C_s \) is set in order to model the processing time of the sampled request as well as pre-emption of the current request by the newly sampled one. State \( \text{ONb} \) is necessary for keeping the correct synchronisation with \( \text{Arrivals} \). This completes the description of our example.

In the general case, more than one transition can be enabled once the dispatcher has provided the selected event to the state machine. Some of them can be in conflict: this happens when the intersection of the sets of states left by the transitions is not empty. Some conflicts can be resolved using priorities. Roughly speaking, as in UMLSD, a transition has higher priority than another transition if its source state is a sub-state of the source of the other one. If the conflict cannot be resolved using priorities, then any of the conflicting enabled transitions may be fired, possibly non-deterministically; this is for example the case for the transition from \( A_1 \) to \( A_2 \) and that from \( A_1 \) to \( A_3 \) in \( \text{Arrivals} \) when \( A_1 \) is in the current configuration and \( \text{sw} \) is offered by the environment. The choice between the two transitions, in a particular run of the system, can be dynamically resolved by means of a race condition: the transition whose clock is zero will be fired. Of course, it may happen that both clocks are zero, in which case non-determinism pops up again. Although non-determinism is easily dealt with for formal correctness, it must be resolved by external means when one is interested in stochastic analysis of the system.

Due to concurrent states, it may happen that more than a single transition is fired as a reaction to a given event. In particular the set of transitions that will fire is a maximal set of enabled, non-conflicting transitions, such that no enabled transition outside the set has higher priority than a transition in the set. When the effects of all such transitions and related actions are complete we say that the current step is completed. At this point, a new event is selected by the dispatcher and a new step is started. In this sense the UML semantics does not allow “chain reactions” within the same step: events generated as a consequence of firing a step are not available to the machine during the same step, but they are available for being dispatched to the machine only from the next step on.

As a consequence of the above interpretation of the step, the time for completing the step is determined by the conjunction of all the clocks associated to all the transitions fired in the step: the step completes when all such clocks go to zero. This suggests the fact that our semantic model must be able to cope with non-exponential (time) distributions. In fact, even if we would restrict all clocks to be exponentially distributed, yet the above conjunction operation amounts to a \( \text{max} \) operation on their random variables, and exponential distributions are not closed under \( \text{max} \).

4. Stochastic Hierarchical Automata

In order to define the stochastic extension of our operational semantics, we first proceed with a purely syntactic translation of \( \text{SUMLSD} \) into Stochastic Hierarchical Automata (SHA) in a similar way as we did for Statechart Diagrams [14]. SHA can be seen as an abstract syntax for \( \text{SUMLSD} \). They are composed of simple sequential stochastic automata related by a refinement function. A location is mapped via the refinement function into the set of (parallel) automata which refine it. Our sample \( \text{SUMLSD} \) is mapped into the SHA of Fig. 2 completed with the edge information shown in Table 1, as discussed in the sequel. There is a clear correspondence between states and locations. Initial states correspond to locations represented by thick boxes. The refinement of a state into one or more sub-states in the statechart is properly represented by the
refinement function \( \rho \); in our example we have \( \rho \, S = \{ S_A1, S_A2 \} \) and \( \rho \, x = \emptyset \) for any other location \( x \). In the figure \( \rho \) is represented by dotted arrows.

Non-inter-level transitions are represented in the obvious way. Each inter-level transition is represented as in the non-stochastic case [14], namely by an edge from the highest ancestor of the real “origin” of the inter-level transition to the highest ancestor of its real target(s). The indication of the real origin of the transition is encoded in the label of the edge. In particular, it is encoded in what is called the source restriction (SR) of the edge, namely the (set of) source state(s) of the original (join) transition. The label also contains the event (EV) which triggers the edge and the corresponding actions (AC) to be performed when the edge is fired. Furthermore, the label of an edge contains the so called target determinator (TD). The target determinator explicitly lists all the basic (i.e. non-refined) locations which must be reached when an edge is fired and is used for inter-level/fork transitions. For the sake of simplicity, and without loss of generality, in this paper we do not allow guards in the edges.

In the following, we will formalize SHA by extending the notion of Hierarchical Automata as defined in [15, 14] and their UML operational semantics given in [14] in order to cope with clocks and their random variables.

In the context of SHA, a stochastic (sequential) automaton (SSA) is defined as follows:

**Def. 1 (Stochastic Sequential Automata)** A SSA \( A \) is a 6-tuple \((\sigma_A, s^0_A, \gamma_A, \lambda_A, \delta_A, k_A)\) where \( \sigma_A \) is a finite set of locations with \( s^0_A \in \sigma_A \) the initial location, \( \gamma_A \) is a set of random clocks - each \( c \in \gamma_A \) is a random variable with distribution function \( F_c \), \( \lambda_A \) is a finite set of edge labels, \( \delta_A \subseteq \sigma_A \times \lambda_A \times 2^{\gamma_A} \times \sigma_A \) is the set of edges, and \( k_A : \sigma_A \rightarrow 2^{\gamma_A} \) is the clock setting function. We assume that the sets of clocks labelling the edges as well as those belonging to \( \text{rng} \, k_A \) be finite.

As mentioned in the previous section, in the context of SHA, the labels in \( \lambda_A \) have a particular structure. Morever, we assume that all edge labels are unique. This can be achieved by assigning them arbitrary unique names. For SSA \( A \) let functions \( SRC, TGT : \delta_A \rightarrow \sigma_A \) and \( CL : \delta_A \rightarrow 2^{\gamma_A} \) be defined as \( SRC(s, l, c, s') = s \), \( TGT(s, l, c, s') = s' \), \( CL(s, l, c, s') = c \). SHA are defined as follows:

**Def. 2 (Stochastic Hierarchical Automata)** A SHA \( H \) is a triple \((F, E, \rho)\), where \( F \) is a finite set of SSA with mutually disjoint sets of locations, i.e. \( \forall A_1, A_2 \in F. \quad A_1 \neq A_2 \Rightarrow \sigma_{A_1} \cap \sigma_{A_2} = \emptyset \) and \( E \) is a finite set of events; the refinement function \( \rho : \bigcup_{A \in F} \sigma_A \rightarrow 2^F \) imposes a tree structure to \( F \). i.e. (i) there exists a unique root automaton \( A_{\text{root}} \in F \) such that \( A_{\text{root}} \notin \bigcup \text{rng} \, \rho \), (ii) every non-root automaton has exactly one ancestor state: \( \forall A \in F \setminus \{ A_{\text{root}} \} \exists ! s \in \bigcup_{A' \in F \setminus \{ A \}} \sigma_{A'} \; A \in (\rho \, s) \) and (iii) there are no cycles: \( \forall S \subseteq \bigcup_{A \in F} \sigma_A \exists ! s \in S. \quad S \cap \bigcup_{A \in \rho \, s} \sigma_A = \emptyset \).

We say that a location \( s \) for which \( \rho \, s = \emptyset \) holds is basic.

In the sequel we shall implicitly make reference to a generic SHA \( H = (F, E, \rho) \).

Every SSA \( A \in F \) characterises a SHA in its turn: intuitively, such a SHA is composed by all those SSA which lay below \( A \), including \( A \) itself, and has a refinement function \( \rho_A \) which is a proper restriction of \( \rho \).

**Def. 3** For \( A \in F \) the automata, locations, clocks and edges under \( A \) are defined respectively as

\[
A = \{ A \} \cup \bigcup_{A' \in F} \left( \bigcup_{s \in A'} (\rho_A \, s) \right) (A' \sigma_A'),
\]

\[
S = \bigcup_{A' \in A} A' \sigma_{A'},
\]

\[
O_A = \bigcup_{A' \in A} A' \gamma_{A'}, \quad \text{and}
\]

\[
T_A = \bigcup_{A' \in A} \delta_{A'}.
\]

The definition of sub-hierarchical automaton follows:

**Def. 4 (Sub-SHA)** For \( A \in F, (F_A, E, \rho_A) \), where \( F_A = (A \, A), \quad \text{and} \quad \rho_A = \rho |_{(S \, A)} \), is the SHA characterised by \( A \).

In the sequel for \( A \in F \) we shall refer to \( A \) both as a SSA and as the sub-SHA of \( H \) it characterises, the role being clear from the context. \( H \) will be identified with \( A_{\text{root}} \). SSA will be considered a degenerate case of SHA.

**Def. 5 (Location Precedence)** For \( s, s' \in S, \quad s \prec s' \text{ iff } s' \in S (\rho \, s) \). Let also \( \preceq \text{ denote the reflexive closure of } \prec \).
The notion of conflict between edges needs to be extended in order to deal with location hierarchy. This is done in exactly the same way as for untimed Hierarchical Automata; the interested reader is referred to [14, 13]. When edges $t$ and $t'$ are in conflict we write $t \# t'$. Priorities are assigned to edges via a function $\pi$ and a partial order $\subseteq$ based on location precedence is defined on them. So, we say that $t$ has lower priority than (the same priority as) $t'$ iff $rt \subseteq pt'$. A configuration denotes a global state of a SHA composed of local states of component SSA:

**Def. 6 (Configurations)** A configuration of $H$ is a set $C \subseteq (S, H)$ such that (i) $\exists s \in \sigma_{\text{next}}, s \in C$ and (ii) $\forall s, A, s \in C \land A \in s \Rightarrow \exists i, s' \in A, s' \in C$.

For $A \in F$ the set of all configurations of $A$ is denoted by $\text{Conf}_A$. Moreover, for configuration $C$ we let $\mathcal{K} C = \bigcup_{A \in F} \left( \bigcup_{s \in C \cap \sigma_A} k_A \right)$.

The operational semantics of an SHA is defined as a SA. In the context of Statechart Diagrams, locations are called statuses and the edges form the STEP-edge relation. The STEP-edges are labelled by the set of the (labels of those) edges of the SSA which have been fired in the SHA. Each status is composed of a configuration, the current environment with which the SHA is supposed to interact, and the set of clocks which are set when entering the status. Also in this paper, as in [14, 13, 7] we model the environment as a parametric abstract data type, since in the definition of UMLSD the particular nature of the environment is left unspecified. In the following we recall the notation we use for the sequel and (vi)

**Def. 8 (STEP-edge Relation)**

\[
\frac{(\text{Sel } E \in E'')}{H \uparrow \emptyset \vdash (C, \{e\}, \emptyset) \xrightarrow{\text{LE}_A} (C', \{e'\}, \emptyset)}
\]

The deduction system for $\text{LE}_A$ is a simple orthogonal extension of the one for the non-stochastic semantics proposed in [14] and is shown in Fig. 3 where the following auxiliary functions are used:

**Def. 9 (Enabled Edges)** For $A \in F$, set of locations $\mathcal{C}$ and environment $E$.

(i) the set of all the enabled local edges of $A$ in $(C, E)$, $\text{LE}_A C E$ is defined as the set

\[
\{t \in \delta_A \mid \{(\text{SRC } t) \cup (\text{SR } t) \subseteq C \land (EV ) t \in E\}
\]

(ii) the set of all enabled edges of $A$ in $(C, E)$ considered as a SHA, i.e. including those of descendents of $A$, $\text{EA}_A C E$ is defined as $\bigcup_{A'' \subseteq (A, A)} \text{LE}_{A''} C E$.

Moreover, for edge $t$, its destination configuration $DTN t$ is the set $\{s' \mid \exists s'' \in TD t, TGT t \preceq s' \preceq s''\}$.
Progress rule

\[
\begin{align*}
  t \in LE_A & \subseteq \mathcal{E} \\
  \forall t' \in P \cup \mathcal{E} & \subseteq \mathcal{E}, \pi t \sqsubseteq \pi t' \\
  A \uparrow P : ((C, \mathcal{E}, \text{cal}G), \{t\} \xrightarrow{\mathcal{E}(t)} (DTN, t, \text{new}(ACt), K(DTN, t)))
\end{align*}
\]

Composition Rule

\[
\begin{align*}
  \{s\} & = C \cap \sigma_A \\
  \rho_A s & = \{A_1, \ldots, A_n\} \neq \emptyset \\
  \bigwedge_{j=1}^{n} A_j \uparrow P & \cup \mathcal{E} \subseteq \mathcal{E} : (C_j, \mathcal{E}_j, \mathcal{G}_j) \xrightarrow{L_jL_jj} (C_j, \mathcal{E}_j, \mathcal{G}_j) \land is_{\text{join}}^{n} E_j \quad \mathcal{F} \\
  \bigcup_{j=1}^{n} L_j = \emptyset & \Rightarrow (\forall t \in \mathcal{E}, \exists t' \in P, \pi t \sqsubseteq \pi t') \\
  A \uparrow P : (C, \mathcal{E}, \bigcup_{j=1}^{n} \mathcal{G}_j) & \xrightarrow{\bigcup_{j=1}^{n} L_j, \bigcup_{j=1}^{n} C_j} (\{s\} \cup \bigcup_{j=1}^{n} C_j, \mathcal{F}, \bigcup_{j=1}^{n} \mathcal{G}_j)
\end{align*}
\]

Stuttering Rule

\[
\begin{align*}
  \{s\} & = C \cap \sigma_A \\
  \rho_A s & = \emptyset \\
  \forall t \in \mathcal{E} & \subseteq \mathcal{E}, \exists t' \in P, \pi t \sqsubseteq \pi t' \\
  A \uparrow P : (C, \mathcal{E}, \mathcal{G}) & \xrightarrow{\emptyset, \emptyset} (\{s\}, \text{nil}, \emptyset)
\end{align*}
\]

Figure 3. Operational Semantics of Stochastic UML Statechart Diagrams

5 Analysis

In this section we shortly discuss how our SUMLSD models can be used for analysis. A SUMLSD model contains both functional and quantitative information. Here we focus on the quantitative information; we shall shortly discuss functional analysis at the end of this section. Typical measures we might be interested in are throughput, response time, failure rate etc. Since general distributions are allowed in SUMLSD, analytical techniques as well as quantitative model checking (see e.g. [1, 10]) can be used only in particular, restricted cases, namely when all step-edges of the operational semantics are labelled by a single clock which is ruled by an exponential distribution. In our example this is not the case: its operational semantics is given in Fig. 4 and the above requirement is violated. In fact, we have edges without clocks and clock \( C \) is not exponential (notice that in this example there is no edge with more than one clock, which may occur in the general case).

In such cases (discrete event) simulation can be used. In the following we shall show how this can be done for our sample model using the simulation tool available for the stochastic process algebra SPADEx [3, 4]. To that purpose we express the stochastic automaton of Fig. 4 as a SPADEx set of process definitions. We use a standard and mechanizable technique for representing the automaton in a so called “state oriented” process algebra specification accordingly. The Composition Rule stipulates how automaton \( A \) delegates the execution of edges to its sub-automata and these edges are propagated upwards. Finally, if there is no edge of \( A \) enabled with priority “high enough” and moreover no sub-automata exist to which the execution of edges can be delegated, then \( A \) has to “stutter”, as enforced by the Stuttering Rule. Notice that stuttering preserves the clock values.

The following theorem, which is easily proven by derivation induction [16], shows that the stochastic extension of the operational semantics is orthogonal in the sense that the automaton of the basic, untimed, operational semantics is the same as that of the stochastic semantics, once clocks and clock settings are removed. For SHA \( A \), let \((\phi_A)\) denote the HA obtained from \( A \) by just removing all clock setting operations in any location and all clock testing in edges.

Theorem 1 For all \( A, P, C, \mathcal{E}, L, \mathcal{C}, \mathcal{E}', \) the following holds: \((\phi_A) \uparrow P : (C, \mathcal{E}) \xrightarrow{L \mathcal{E}} (C', \mathcal{E}')\) if and only if there exists \( \mathcal{G}, \mathcal{C}, \mathcal{G}' \) such that \( A \uparrow P : (C, \mathcal{E}, \mathcal{G}) \xrightarrow{L \mathcal{E}} (C', \mathcal{E}', \mathcal{G}')\).

Proof By derivation induction. \( \square \)
[21], with the obvious extensions for dealing with clocks. Each location is associated with a process definition, so we have 14 processes, namely AS1, …, AS14. Moreover, for purely notational reasons we name the edges as Tx where x is a1 … a6, s1 … s8. Clock setting (resp. expiration) is denoted by \{ clocks \} (resp. \{ clocks \} \rightarrow). Finally, action prefix and choice are denoted by semicolon and + respectively. The SPADES expression for the SA of Fig. 4 is given in Fig. 5.

The SPADES simulator takes as input the process specification together with the clock distribution definitions and a directive specifying what kind of adversary should be used in the simulation. The role of the adversary is to resolve the possible non-determinism present in the stochastic automaton. In fact, although there is wide consensus on the usefulness of non-determinism as an abstraction tool in stepwise design, non-deterministic models are not well suited for discrete event simulation. For our model we choose a equi-probable/prioritised adversary. Since we want the controller to be released as soon as as the current processing is complete, we require that whenever Cs is zero, Ts d has priority over Ts a and similarly, Ts d over Ts a. All other non-deterministic situations are resolved using equal probabilities. Notice that here we are concerned with dynamic priorities, which take into consideration also the values of the clocks as opposed to UML static transition priorities, which are only related to the hierarchical structure of Statechart Diagrams.

Let’s suppose we are interested in the number of requests the processing of which is aborted once started. This knowledge might be useful when trying to keep pre-emption rate low, operating on the sampling rate and/or the required processing time. Pre-emption corresponds to the execution of edge Ts6 or Ts7. We performed a batch-means analysis with 10 simulations runs of 1000 time units each, taking the mean. The result is summarised in Fig. 6, where the number of times Ts6 or Ts7 is executed is shown as a function of the processing rate (i.e. the rate of the exponential distribution for Cs), for different values of the sampling period (i.e. clock Cn), namely 0.1, 0.5 and 0.9 time units. As one would expect, the number of pre-empted requests decreases when the processing rate increases. On the other hand, we see that it decreases also when the sampling rate increases. This phenomenon is due to the (clock-)synchronous nature of our model, which in turn is inherent in (UML) Statechart Diagrams. In fact, whenever Cs expires one would expect the Controller to move to location OFF immediately. Unfortunately, this is not the case in our model since, for synchronisation reasons, it has to wait for a signal from Arrivals. Suppose for instance that Controller is in location ONa when Cs expires. It can happen that the new signal from Arrivals is a on, so that Ts6 is executed but this does not correspond to pre-emption. A similar reasoning can be done for Ts7 in similar circumstances. It is of course clear that the lower the sampling rate is, the more serious this
phenomenon is, the less accurate our measure gets.

From the same simulation runs, by focusing on the ratio between the detected requests ($T_{a1}, T_{a3}$ and $T_{a6}$) and the actual arrivals ($T_{a4}$), we get that for a sampling period of 0.1, in the average, no arrivals are missed, while for a sampling period of 0.5 and 0.9 we have 0.98 and 0.97 respectively.

Notice that although in this paper we performed the mapping of SUMLSD transitions to SPADEx ones manually, also such a mapping can be automatized. More specifically, starting from the SUMLSD transitions one is interested in, one can automatically identify the related STEPs edges of the semantics stochastic automaton in a similar way as shown in [7].

We close this section by emphasising once more that for SUMLSD $M$ all functional analysis techniques can still be used on $\phi M$, i.e. the pure version of $M$ obtained by just removing time/stochastic information, since Theorem 1 guarantees that adding or deleting quantitative information does not change the state/transition structure of the semantics. So the semantics of $M$ can be formally related to other, more concrete, automata representing detailed behaviour of the model or it can be formally checked against requirement properties, expressed for instance in temporal logics. These formal checks can also be done automatically by means of model-checking tools [7]. Notice here that although the potential state space is not affected by quantitative information, the states which can be actually reached, due to time constraints, can be a proper subset of the potential state-space. This may happen for instance when one uses FIFO queues for the environment (or any other data structure which poses an order on the events it stores) and distributions with empty intersections, like for instance different constants. In such cases proper restrictions are useful, like those imposed by Stochastic Petri Nets (see e.g. [6]), that is, the set of clocks labelling any edge should not contain more than one clock with constant distribution, or in general two or more clocks with non-intersected distributions. Alternatively, special care is needed for the verification of liveness properties, since the “good states” they may prove to exist might indeed be states which will never be reached due to the time constraints. No such problems instead exist when dealing with safety properties like, e.g., showing the absence of deadlock.

6 Conclusions and further work

In this paper we presented a stochastic extension of UML Statechart Diagrams. The formal operational semantics have been provided and its correctness with respect to the untimed semantics has been shown. A small example of use of the proposed notation for quantitative analysis via simulation has been shown. The paper shows that simple extensions can prove quite useful and can be performed in a safe way when the correct conceptual and technical tools are used. In particular, our original definition of the semantics for a subset of UML Statechart Diagrams given in [14] has proven very well suited for orthogonal extensions like that presented in this paper. The reason for that lays on the particular form of definition we used for it, namely hierarchical and recursive. All relevant definitions in the present paper boil down to simple extensions of the analogous ones for the untimed semantics given in [14], yet the added value is the possibility of quantitative analysis directly from the enriched UMLSD models.

All the steps presented in the paper are mechanizable, so a tool for automatic analysis of SUMLSD based on simulation can be implemented and we indeed plan to build such tool.

As we have seen in our example, there are situations in which the clock-synchronous approach of Statechart Diagrams may pose modelling problems. One way to tackle these problems is to generalise our semantics to a set of statecharts asynchronously communicating via queues. This is actually the framework proposed in the UML. Although we have some methodological perplexity on such an approach (see [13] for a discussion of this issue), our semantics can be adjusted in order to deal with more than one statechart and we plan to do this in the near future. The modification affects only Def.8, while the rest of the operational semantics remain unchanged.

The semantics proposed in this paper are based on the assumption that a step is kind of atomic: no information is given on how things happen within a step. It might sometimes be useful to “look inside” the step instead. To that purpose, we are planning to develop an alternative semantic model where we do not reduce the complete SHA to a SA, but we rather map each SSA of the SHA into a distinct SA and we analyse the behaviour of the network of such automata. This way we could also exploit one of the most
beneficial features of SA and their presentation languages like SPADES, namely compositionality.

Finally, UMLSD allow the use of non-deterministic environments. Similarly, we can tune our semantics in order to deal with probabilistic environments, again by acting on Def.8.

7 Acknowledgments

We would like to thank Istvan Majzik and Pedro D’Argenio for fruitful discussions on some issues related to the present work.

References


