

# A QoS-oriented Extension of UML Statecharts<sup>\*</sup>

David N. Jansen<sup>1</sup>, Holger Hermanns<sup>2,1</sup>, and Joost-Pieter Katoen<sup>1</sup>

<sup>1</sup> Universiteit Twente, Postbus 217, 7500 AE Enschede, The Netherlands

<sup>2</sup> Universität des Saarlandes, Im Stadtwald, 66123 Saarbrücken, Germany  
{dnjansen,hermanns,katoen}@cs.utwente.nl

**Abstract.** Performance, dependability and quality of service (QoS) are prime aspects of the UML modeling domain. To capture these aspects effectively in a modeling language requires easy-to-use support for the specification and analysis of randomly varying behaviors. This paper introduces an extension of UML statecharts with randomly varying durations, by enriching a specific syntactic construct: The “after” operator is equipped with (discrete or continuous) probability distributions, determining the duration of the delay caused by this operator. The semantics of this extension is given in terms of a variant of stochastic automata. It is shown how existing model-checking tools can be used to calculate model-inherent QoS characteristics automatically. We study a UML model of an automatic teller machine scenario using this approach.

## 1 Introduction

*Background and motivation.* The UML is pervading many challenging engineering areas including real-time and embedded system design. Embedded systems designers are usually facing various challenges if they strive for systems with *predictable quality of service* (QoS). Most QoS aspects of current embedded systems are time-related features and properties. They are usually referred to as soft real-time constraints, and are of stochastic nature. To incorporate these constraints in the embedded systems design process is a challenging issue [22]:

- System dynamics is becoming ever more complex, making it more and more difficult to properly predict the QoS.
- The trend to networked embedded systems raises issues like message buffering, inter-dependencies due to media sharing, and communication characteristics, all influencing the system QoS.
- Applications involve more and more non-functional features in the form of multimodal interfaces and multimedia support, having impact on the QoS.

A workable modeling and analysis approach to embedded system QoS is based on the observation that networks, interfaces, and even circuits on chips [9, 33] can be understood and modeled as discrete systems exhibiting some form of

---

<sup>\*</sup> Parts of this work was achieved during the Dagstuhl seminar 03201 ‘Probabilistic Methods in Verification and Planning’ held at IBFI Schloss Dagstuhl in April 2003.

stochastic behavior, such as error rates, response time distributions, or message queue lengths.

Mathematically speaking, the QoS characteristics of a given embedded system induce families of stochastic decision processes, e. g. Markov chains or semi-Markov decision processes. However, these mathematical objects are too fine grained to be directly specifiable by an embedded systems designer. Therefore, one must rely on modeling techniques and tools for stochastic processes. While in principle the UML provides the right ingredients to model discrete event dynamic systems, it lacks support for stochastic process modeling. This issue has been addressed in both the UML profile for schedulability, performance and time [31], and in the draft UML profile for modeling QoS and fault tolerance [30]. These profiles suggest annotational extensions of UML providing means to specify performance, dependability and QoS characteristics at various levels. However, the vague semantics of the UML and of its annotational extensions drastically hampers QoS analysis: It is simply impossible to distill a faithful performance or QoS quantity from a stochastic (decision) process that is only partially defined. This means that model-based QoS *prediction* is only possible for UML fragments with a rigid formal semantics.

In this paper, we attack this problem for UML statecharts. We provide a formal semantics of an extension of UML statecharts, the extension being both simple and easy to understand, yet powerful enough to model a sufficiently rich class of stochastic decision processes. The extension is twofold. One extension, first introduced in [20], allows state transitions to select probabilistically out of different effects, much like the rolling of a die can have one out of six effects, determined probabilistically. The second extension is novel, yet simple and conservative: The “after” operator of statecharts is given a stochastic interpretation, allowing the use of arbitrary probability distributions for modeling, such as  $\text{EXP}[10 \text{ min}]$  for a negative exponential distribution with a mean of 10 min, or  $\text{UNIF}[10 \text{ h}, 15 \text{ h}]$  for a uniform distribution in the interval from 10 to 15 hours. The resulting statecharts dialect is called `STOCHARTS`, and contains UML statecharts (up to some minor features such as deferred events) as a subset.

*Example 1.* Consider the following workflow model of a car damage assessment by an insurance company. In the damage assessment it is decided whether a damaged car should be repaired and whether the garage offers an acceptable price for the repair. To assure customer satisfaction, the insurance company imposes the following QoS requirement: “In at least 95 % of the cases, an entire damage assessment takes less than 4 days.” The damage assessment is split in several mandatory and optional phases which take varying times, and in summary delay the repair of the damaged car. In Fig. 1, the damage assessment is succinctly modeled by a `STOCHART`. Once a workflow starts, the damage assessor contacts a garage. The time this step takes may vary, and it is modeled by an “after” operator parameterized with a negative exponential distribution having a mean duration of one minute. After completion of this delay, the conversation may lead to one out of two outcomes, which are modeled probabilistically. Either a physical assessment date is negotiated (with probability 0.3), or a phone assessment is

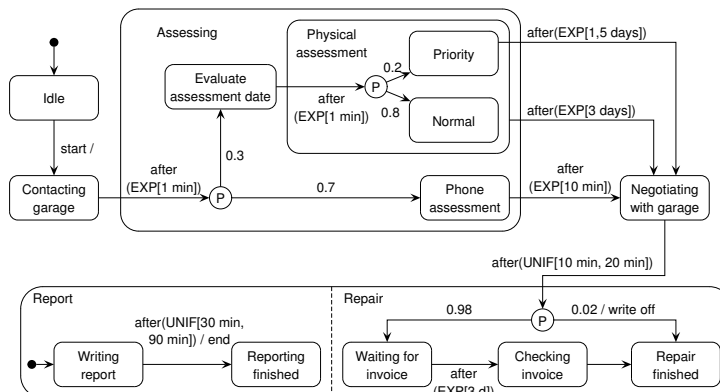


Fig. 1. Damage assessing process

carried out (with probability 0.7). This behavior is modeled graphically by using a  $\textcircled{P}$ -pseudonode, a drawing aid inspired by the  $\textcircled{C}$ -pseudonode of [15]. With this probabilistic choice we abstract from a detailed analysis of the reasons that lead to the decision. Dependent on the decision taken, the assessment continues. With the above explanation, the subsequent intuitive behavior should be self-explanatory.

*Technical merits.* To make STOCHARTS a useful tool in QoS modeling and prediction, requires more than a simple extension of UML with an intuitive interpretation. In order to support model-based QoS prediction, a formal mathematical interpretation is indispensable. Otherwise, model-based calculations are not trustworthy. Concretely, we needed to solve various challenges which we report in this paper: (i) The semantic model associated with STOCHARTS needed to be defined together with semantic mapping which conservatively extends the standard semantics. As in [20], we take as a representative the requirements-level semantics of Eshuis and Wieringa [12], which is based on the Statemate semantics [16] and its formalisation by Damm et al. [10]. We have chosen the requirements-level semantics because it is simple and matches our intended use of the UML closely. (ii) In order to associate a stochastic interpretation to collaborative collections of statecharts embedded in arbitrary environments, we provide a compositional semantics, which uses concepts from Input/Output (I/O) automata [26]. (iii) Our semantic embedding, though rather parsimonious in its scope (just one operator is touched), exploits lessons learned in a decade of research in formal specification of stochastic processes, rooted in the seminal works on stochastic Petri nets [1, 2], stochastic process algebra [14, 18] and probabilistic automata [32]. (iv) We provide experimental evidence that this approach can be used for modeling and model-based prediction of interesting QoS quantities. The QoS characteristics of an automatic teller machine scenario are modelled in the UML extension and the user-perceived QoS is predicted using model-checking. Other model-based

techniques can equally well be applied for QoS prediction, such as simulation or numerical analysis.

*Related work.* Quite some research has been devoted in recent years to enable QoS analysis from UML. Several authors have suggested mappings of statechart fragments onto stochastic Petri nets variants [6, 19, 23, 25], of which the syntax suggested by Lindemann et al. [25] is most similar to ours. This approach does not consider nondeterministic phenomena, while we do. Others have linked to process algebra [28, 8]. Closest to our work is the work reported in [13], since both approaches use variations of stochastic automata as semantic models. In their approach, however, clocks appear as syntactic entities, and are explicitly referenced in the syntax. Our approach instead keeps the syntax unchanged, while timers are used under the hood. Furthermore, we do not limit standard UML statecharts in any way.

*Organization of the paper.* Section 2 introduces our statechart dialect. Section 3 introduces the semantic model, and Section 4 describes a mapping from STOCHARTS to this model. A larger example of using STOCHARTS in modeling and analysis is presented in Section 5. Section 6 concludes the paper. In the remainder of this paper, we assume familiarity with basic probability theory [29].

## 2 StoCharts

This section describes our syntactic extensions to UML statecharts together with their informal interpretation.

*UML statecharts.* A (simple) UML statechart consists of

- a set of *Nodes*, organized as a simple typed tree where nodes are of type ‘basic’ (leafs), ‘and’, or ‘or’. The root node and children of ‘and’ nodes are of type ‘or’. Each ‘or’ node has a distinguished, *initial* child node.
- a set of *Events* with the distinguished element  $\perp$  denoting “no event required”.
- a set *Vars* of (typed) variables or attributes together with an initial valuation  $V_0 : Vars \rightarrow \mathcal{D}$ , assigning initial values to the variables. Here  $\mathcal{D}$  subsumes the domains of all variables.
- a set *Edges* of edges. Each edge connects a set of source nodes to a set of target nodes, and is labeled with an event, a guard (a Boolean expression), and a (possibly empty) set of actions (assignments to variables or sending messages to other objects).

For analysis purposes, we assume that all these sets are finite and that variable domains are restricted to bounded integers only.

*Our enrichment.* We propose to extend statecharts as follows: an “after” operator is considered to indicate random delays, and the notion of edges is refined into probabilistic edges (P-edges) to handle discrete probabilistic branching. The latter concept has recently been introduced by us in [20].

- Like an event, the after-operator acts as a trigger of an edge (or a P-edge) and has as parameter a cumulative distribution function  $F$  where  $F(t)$  determines the probability to wait at most  $t$  time units.
- A P-edge is a tuple  $(X, e, g, P)$  where  $X$  is a non-empty set of source nodes,  $e$  is either an event or an after operator,  $g$  is a guard, and  $P$  is a function assigning probabilities to pairs consisting of a set  $A$  of actions, and a non-empty set  $Y$  of target nodes<sup>3</sup>.  $P(A, Y)$  denotes the probability of selecting target  $(A, Y)$ .

To simplify matters, P-edges are indicated by numbers, i. e., we use a bijective index function  $\iota$  to identify P-edges. When  $\iota(j) = (X_0, e_0, g_0, P_0)$ , we write  $\iota(j).X$  for  $X_0$ ,  $\iota(j).e$  for  $e_0$  etc. We also number STOC HARTs. A STOC HART numbered  $i$  is indicated by the tuple  $(Nodes_i, Events_i, Vars_i, PEdges_i)$ . A P-edge can be projected to a *set* of ordinary edges in the following way: an edge is a triple  $(j, A, Y)$ , where  $j$  identifies a P-edge such that  $(A, Y)$  is assigned a positive probability, i. e.,  $P(A, Y) > 0$ .

The *scope* of an edge  $(j, A, Y)$  is the smallest (in the parent–child-hierarchy) node that is not affected when the edge is executed. That is, it is the smallest node of type ‘or’ that contains both the source nodes  $\iota(j).X$  and the target nodes  $Y$ . We require all edges (with probability  $> 0$ ) belonging to a P-edge to have the same scope. This may be understood as: “arrows from a P-pseudonode that cross node borders – like inter-level transitions – should be avoided”. This restriction slightly limits the expressiveness, but eases the semantics considerably. See [20] for an extensive discussion of this issue as well as an alternative, unrestricted semantics.

*Drawing P-edges.* A P-edge consists of two parts: an arrow labeled with an event and a guard directed to a P-pseudonode (denoted  $\textcircled{P}$ ) from which several arrows to target nodes emanate, each labeled with a probability and an action set. For example, the statechart in Fig. 1 contains a P-edge that starts in node **Contacting garage** and allows to choose between the targets **Phone assessment** and **Evaluate assessment date**. P-pseudonodes are omitted in case the P-edge consists of a single edge that occurs with probability one.

*Intuitive interpretation.* The STOC HART is always in some state which consists of one or several nodes. A P-edge may be taken (i. e., is enabled) if all its source nodes are part of the current state, its guard holds, and if either its event happens or the delay associated with its after operation expires. Nondeterministically an enabled P-edge is selected (or a number of non-conflicting ones), and its discrete

<sup>3</sup> Formally,  $P$  is a probability measure on the discrete probability space  $(\mathbf{P}(Actions_i) \times (\mathbf{P}(Nodes_i) \setminus \{\emptyset\}), P)$ .

probabilistic choice will be resolved. Once the selected edge is taken, its actions are executed, and its target nodes will be entered. On entering a node with an outgoing P-edge labeled with an  $\text{after}(F)$  operation, a sample is taken from distribution  $F$  and a timer is set accordingly. The corresponding outgoing edge becomes enabled once the timer expires.

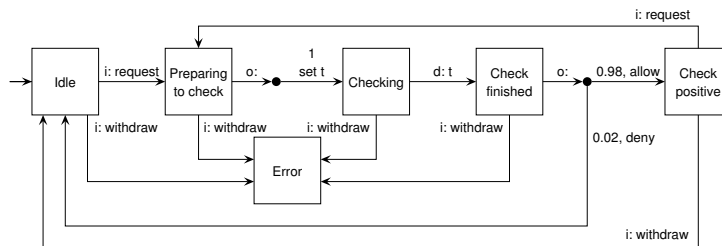
### 3 Underlying transition model

The formal semantics of STOCARTS is defined in terms of an extension of labeled transition systems, basically automata with *locations* representing the possible configurations of the system, and transitions between locations representing the system's evolution. These transition systems are equipped with *timers* to model probabilistic delays, and with a set of *actions* to model system activities. The use of timers in transition systems is very much in line with the use of clocks in, e.g., timed automata [4, 27]. While clocks in timed automata run forward at the same pace and are always reset to 0, our timers are initialized by sampling a probability distribution and run backwards, all at the same pace. On the other hand, timers are always checked for expiration (i.e., is the timer equal to zero?), while clocks can be checked in complex conditions.

Input and output actions are distinguished to allow for the composition of transition systems, like in I/O-automata [26]. Three types of transition relations are used: input transitions, output transitions, and delay transitions, the latter being enabled once a timer expires. Whereas input and delay transitions are standard ternary relations (input is even a function), the output transition relation is probabilistic. Like in I/O-automata we assume input-enabledness, i.e., in each location any input can be accepted. The resulting model, a *stochastic I/O-automaton* (IOSA, for short), is a tuple  $(L, l_0, \mathcal{T}, \mathcal{A}, I, O, \Delta)$  consisting of:

- a set  $L$  of locations with initial location  $l_0 \in L$ .
- a set  $\mathcal{T}$  of timers such that each timer  $t$  has an associated cumulative distribution function (cdf)  $F_t$ .
- a set  $\mathcal{A}$  of actions, partitioned into inputs  $\mathcal{A}^{\text{in}}$  and outputs  $\mathcal{A}^{\text{out}}$ .
- an input transition relation, described by the function  $I : L \times \mathcal{A}^{\text{in}} \rightarrow L$ .
- a probabilistic output transition relation  $O \subseteq L \times \mathbf{Prob}(\mathcal{A}^{\text{out}} \times \mathbf{P}(\mathcal{T}) \times L)$ .
- a delay transition relation  $\Delta \subseteq L \times \mathcal{T} \times L$ .

*Example 2.* Figure 2 depicts an IOSA modeling the behavior of a bank from the perspective of an automatic teller machine (ATM). Output, input and delay transitions are labeled  $\circ$ .,  $i$ ., and  $d$ ., respectively. The ATM sends a request (action **request**) to check whether a certain withdrawal is allowed; from the bank's point of view, this is an input. It needs some time to check the allowance; this is modeled by setting a timer  $\mathbf{t}$  and waiting until it expires. If the outcome of the check is positive, the ATM may request to actually withdraw the money from the account (**withdraw**). If the ATM sends a withdrawal request without allowance, the bank moves to an error state.



**Fig. 2.** Stochastic I/O automaton of the bank behavior

*Informal semantics.* The interpretation of IOSA uses similar ingredients as the semantics of timed automata [4]. In fact, IOSA are symbolic representations of infinite (probabilistic timed) transition systems [11] where a *state* consists of a location and a timer valuation, recording the current value of each timer. We only present the intuition behind this interpretation; its formalization can be found in [21]. The behavior of the IOSA begins in its initial location, with every timer being set to a random sample drawn according to its associated distribution. In each state, one enabled transition may be selected for execution. Only if no transition is enabled, time may pass until some transition will be enabled (if any). An output transition is always enabled, an input transition is enabled only if its input action is offered by the environment, and a delay transition is enabled only if its timer has expired (i. e., has reached value zero). If input action  $a$  is offered in location  $l$ , the unique next location is  $I(l, a)$ . If timer  $t$  expires in location  $l$ , delay transition  $(l, t, l')$  may be taken and results in moving to  $l'$ . Output transitions are slightly more involved. To take an output transition emanating from location  $l$ , first a probability space  $\mathcal{P}$  with  $(l, \mathcal{P}) \in O$  is nondeterministically selected. Subsequently, one of the possible targets  $(a, T, l')$  in  $\mathcal{P}$ , is probabilistically chosen. On taking this transition, action  $a$  is generated, all timers in set  $T$  are reset and the system evolves to location  $l'$ .

*Composition.* Compositionality is of utmost importance to enable the use of IOSA in arbitrary embedding contexts. That is to say, each of our extended statecharts can be embedded in any environment, and the semantics of their composite behavior can simply be determined in a modular way. This approach even allows the embedding environment to be specified using a different specification technique provided its interpretation can be given in terms of stochastic I/O-automata. This differs from our earlier approach [20] where we only considered system randomness for which a closed system approach suffices.

In a network of IOSA one automaton is selected (nondeterministically) to take an output transition. This output action immediately triggers corresponding input transitions in the other automata. (By input-enabledness, each automaton can react to the output.) The composite automaton accepts input action  $a$  if all its components do so. Conversely, the composite automaton takes a delay transition, if one of its components does so. Composition is defined as a bi-

nary operator; as this operator is associative, it can easily be generalized to finite collections of IOSA. For  $\text{IOSA}_i = (L_i, l_{0,i}, \mathcal{T}_i, \mathcal{A}_i, I_i, O_i, \Delta_i)$  (with  $i = 1, 2$ ) with  $\mathcal{A}_1^{\text{out}} \cap \mathcal{A}_2^{\text{out}} = \emptyset$ , their composite automaton, denoted  $\text{IOSA}_1 \otimes \text{IOSA}_2 = (L, l_0, \mathcal{T}, \mathcal{A}, I, O, \Delta)$  is defined by:<sup>4</sup>

- $L = L_1 \times L_2$  with initial location  $l_0 = (l_{0,1}, l_{0,2})$
- $\mathcal{T} = \mathcal{T}_1 \cup \mathcal{T}_2$
- $\mathcal{A} = \mathcal{A}_1 \cup \mathcal{A}_2$ , where  $\mathcal{A}^{\text{out}} = \mathcal{A}_1^{\text{out}} \cup \mathcal{A}_2^{\text{out}}$  and  $\mathcal{A}^{\text{in}} = (\mathcal{A}_1^{\text{in}} \cup \mathcal{A}_2^{\text{in}}) \setminus \mathcal{A}^{\text{out}}$ .
- $I$  is the smallest relation defined by the rule:

$$\frac{a \in \mathcal{A}^{\text{in}}}{I((l_1, l_2), a) = (I_1(l_1, a), I_2(l_2, a))}$$

- $O$  is the smallest relation defined by two symmetric rules. We only give the rule where  $\text{IOSA}_1$  takes the initiative for output:

$$\frac{(l_1, \mathcal{P}_1) = (l_1, (\mathcal{A}_1^{\text{out}} \times \mathbf{P}(\mathcal{T}_1) \times L_1, P_1)) \in O_1}{((l_1, l_2), \mathcal{P}) = ((l_1, l_2), (\mathcal{A}^{\text{out}} \times \mathbf{P}(\mathcal{T}) \times L, P)) \in O}$$

where  $P$  is the probability measure defined by:

$$P(\{(a, T, (l_1, I_2(l_2, a)))\}) = P_1(\{(a, T, l_1)\}) \quad \text{if } a \in \mathcal{A}_1^{\text{out}} \text{ and } T \subseteq \mathcal{T}_1$$

and  $P(\{\omega\}) = 0$  otherwise.

- $\Delta$  is the smallest relation defined by the two rules:

$$\frac{(l_1, t, l'_1) \in \Delta_1}{((l_1, l_2), t, (l'_1, l_2)) \in \Delta} \quad \frac{(l_2, t, l'_2) \in \Delta_2}{((l_1, l_2), t, (l_1, l'_2)) \in \Delta}$$

*Related models.* As already mentioned, IOSA are inspired by timed (I/O) automata [4, 27] and probabilistic automata [32]. From another perspective, IOSA extend stochastic automata [11] with discrete probabilistic branching and input-output communication. Another strongly related model is probabilistic I/O automata (PIOA) [34]. The main differences between IOSA and PIOA are: All clocks in the latter are governed by negative exponential distributions, while IOSA allow arbitrary distributions. (Despite their name PIOA include stochastic timing.) All choices in PIOA are probabilistic, while IOSA also allow non-deterministic choice. Consequently, a composite PIOA chooses the automaton to produce an output in a probabilistic manner rather than in a non-deterministic manner. Each probabilistic I/O automaton with deterministic input (and no internal actions) can be mapped to an equivalent IOSA; the reverse, however, is not true.

## 4 Formal semantics

We first describe how for an individual STOCHART, steps are selected and executed, and then consider the semantics of a collection of STOCHARTS.

<sup>4</sup> To simplify the notation, we extend  $I_1$  and  $I_2$  by  $I_i(l_i, a) = l_i$  if  $a \notin \mathcal{A}_i$ .

*How to construct a step for a single STOCHART.* A *configuration* describes the most important part of a STOCHART. Configuration  $C_i$  of STOCHART  $SC_i$  is a set of nodes containing: the root node, one child for each ‘or’-node in  $C_i$ , and all children of all ‘and’-nodes in  $C_i$ . The set of configurations of  $SC_i$  is denoted  $Conf_i$ . A *location* of  $SC_i$  is a triple  $(C_i, I_i, V_i)$  with configuration  $C_i$ , event-set  $I_i$  (to which still has to be reacted) and valuation  $V_i : Vars_i \rightarrow \mathcal{D}$ . The set of all valuations of  $SC_i$  is denoted  $Val_i$ . The validity of guard  $g$  in a location depends on the configurations  $C_1, \dots, C_n$  and the valuations  $V_1, \dots, V_n$ . We denote by  $(C_{1\dots n}, V_{1\dots n}) \models g$  that  $g$  holds.

The set  $PT$  of P-edges to be executed has to obey a couple of requirements: all P-edges in  $PT$  must be enabled, they must be pairwise consistent, obey the priority constraints, and, finally,  $PT$  must be maximal. For a more detailed description of these requirements, see [12, 20]. The following algorithm constructs  $PT$ , and probabilistically selects a step from it: Assume that the current location of  $SC_i$  is  $(C_i, I_i, V_i)$ .

1. Calculate the set  $EnP(C_i, I_i, V_i)$  of enabled P-edges. P-edge  $j$  is enabled iff:

$$\iota_i(j).X \subseteq C_i \wedge \iota_i(j).e \in I_i \cup \{\perp\} \wedge (C_{1\dots n}, V_{1\dots n}) \models \iota_i(j).g$$

2. Calculate  $PSteps(EnP(C_i, I_i, V_i))$ , where  $PSteps(E)$  (for  $E \subseteq PEdges_i$ ) contains all maximal, prioritized, consistent sets of P-edges  $\subseteq E$ .
3. Select nondeterministically a set of P-edges  $PT$  of  $PSteps(EnP(C_i, I_i, V_i))$ .
4. Draw samples from the probability spaces of the P-edges in  $PT$ , resulting in a set of edges  $T$  (called a step).

The second and third phases are similar to the nextstep-algorithm in [12]. The last phase can be described by a discrete probability space over  $\mathbf{P}(Edges_i)$ . Its probability measure  $P$  is a function on sets of sets defined as follows. For any selection of  $A_j$  and  $Y_j$  (for  $j \in PT$ ),

$$P(\{(j, A_j, Y_j) \mid j \in PT\}) = \prod_{j \in PT} (\iota_i(j).P)(\{(A_j, Y_j)\})$$

and  $P(\{\omega\}) = 0$  otherwise. Subsequently, the selected steps (in the last phase) are executed. For a single STOCHART this amounts to update the variables and events that occur in the activities and to determine the new state.

*How to construct a step for a collection of STOCHARTs.* Actions of one STOCHART may influence events of other STOCHARTs. This is reflected in the step execution of a collection of STOCHARTs. The collection of STOCHARTs  $(Nodes_i, Events_i, Vars_i, PEdges_i)_{i=1}^n$  is mapped to a single IOSA  $(L, l_0, \mathcal{C}, \mathcal{A}, I, O, \Delta)$  as follows:

- $L = \prod_{i=1}^n Conf_i \times \mathbf{P}(Events_i) \times Val_i$
- $l_0 = (s_{0,1}, \dots, s_{0,n})$ , where  $s_{0,i} = (C_{0,i}, \emptyset, V_{0,i})$  is the initial location of  $SC_i$

- For each P-edge  $\iota_i(j)$  with label  $\text{after}(F_{ij})$ , there is a timer  $t_{ij} \in \mathcal{T}$  with cdf  $F_{ij}$
- $\mathcal{A}^{\text{in}} = \mathbf{P} \left( \bigcup_{i=1}^n \text{Events}_i \right)$  and
- $\mathcal{A}^{\text{out}} = \mathbf{P} \left( \bigcup_{k=1}^n \{i.e \mid \exists(j, A, Y) \in \text{Edges}_k : \text{send } i.e \in A \wedge i \notin \{1, \dots, n\}\} \right)$
- $I$  is the smallest relation defined by the rule:

$$\frac{E \in \mathcal{A}}{(C_i, I_i, V_i)_{i=1}^n \xrightarrow{E} (C_i, I_i \cup (E \cap \text{Events}_i), V_i)_{i=1}^n}$$

- If  $PT_i = (\mathbf{P}(\text{Edges}_i), P_i) \in P\text{Steps}(EnP(C_i, I_i, V_i))$  for each  $i$ , then  $((C_i, I_i, V_i)_{i=1}^n, (\mathcal{A} \times \mathbf{P}(\mathcal{C}) \times L, P)) \in O$ , where  $P$  is the probability measure defined by:

$$P(\{(E, T, \text{Execute}(C_{1\dots n}, T_{1\dots n}, V_{1\dots n}))\}) = \prod_{i=1}^n P_i(\{T_i\})$$

where  $E$  is the set of events that are sent to the environment:

$$E = \bigcup_{k=1}^n \{i.e \mid \exists(j, A, Y) \in T_k : \text{send } i.e \in A \wedge i \notin \{1, \dots, n\}\}$$

and  $T = \{t_{ij} \mid \iota_i(j) \text{ becomes enabled}\}$ . Let  $P(\{(E, T, \omega)\}) = 0$  otherwise.

- $\Delta$  is defined by the rule:

$$\frac{\begin{array}{l} X \subseteq C_{i_0} \quad \iota_{i_0}(j) = (X, \text{after}(F_{i_0,j}), g, P) \in P\text{Edges}_{i_0} \\ I'_{i_0} = I_{i_0} \cup \{\text{after}(F_{i_0,j})\} \quad \forall i \neq i_0 : I'_i = I_i \end{array}}{(C_i, I_i, V_i)_{i=1}^n \xrightarrow{t_{i_0,j}} (C_i, I'_i, V_i)_{i=1}^n}$$

for some  $i_0 \in \{1, \dots, n\}$ .

Note that the communication mechanism between STOCHARTS differs from the communication between IOSA. Whereas all STOCHARTS may take their steps and produce output at once, only one component IOSA of some composite IOSA is allowed to take an output edge at a time. In our context, the composition of IOSA is only used to combine the embedded system under consideration with the embedding environment.

## 5 StoCharts in action

As an example illustrating how STOCHARTS can be used for modeling and model-based QoS prediction, we develop a model of an Automatic Teller Machine (ATM). An ATM (often called “cash dispenser”) distributes money to clients who identify themselves with a personal chip-card and a PIN (personal identification number). We measure whether the ATM satisfies a customer satisfaction requirement. The distributions and probabilities used in this examples should in practice be obtained from statistical analysis of observed behavior, while for this example we decided to use plausible, but ad-hoc chosen, parameters.

*Model.* We model a system consisting of three components, (i) the ATM subsystem, (ii) the bank, and (iii) the client. Figure 3 shows a typical interaction of the ATM with its environment (client and bank). The ATM is described by an UML-statechart in Fig. 4. This statechart contains neither  $\textcircled{P}$ -nodes nor after-delays.<sup>5</sup> Since we assume familiarity with standard statecharts, we do not comment on the modelled behavior.

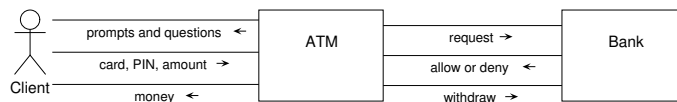
We have modelled the bank’s behavior as viewed by the ATM in terms of the IOSA in Fig. 2. To check an account balance takes some time (given by the distribution associated with timer  $t$ ). With a certain probability (0.98) withdrawing is permitted, and the bank processes withdrawals, otherwise it sends a denial back to the ATM. We abstract from the internal structure of the bank and only model the delays to handle the ATM’s requests.

The client’s behavior is also modelled as a STOCHART, depicted in Fig. 5. Each action of the client is assumed to take some non-negligible time, denoted in the figure. The client first inserts the card. Then the client is ready to key in the PIN and also the desired amount, the order depending on the type of machine confronted with. There is a probability (0.01) that the user enters the wrong PIN, and similarly the amount of money the client desires to withdraw varies probabilistically. Afterwards, the client awaits the money and the card back (also in either order). If the ATM reports a denial of the bank, the client doesn’t expect any money, but still awaits the chip-card.

*Model analysis.* Model analysis of a collection of STOCHARTS is based on the analysis of the associated IOSA. This model can be analysed using simulation, or using model-checking. In the most general case model-checking algorithms in the style of [3] are needed. If nondeterminism is absent (which is the case in our example), one can deal with simpler algorithms, such as the ones implemented in PROVER [35]. PROVER estimates probabilities of system requirements up to a user-specified confidence by means of discrete event simulation. If all distributions are exponential, Markov-chain model-checking tools such as ETMCC [17] or PRISM [24] are available. These tools calculate probabilities with which certain system requirements are satisfied by the model. As a requirement specification language, all the above tools rely on the stochastic temporal logic CSL [5].

*Requirement.* We focused on a probabilistic requirement of the following form: “Once the card has been entered, the probability that the client obtains the money within time  $t$  is at least  $p$ .” This can be considered as a QoS requirement

<sup>5</sup> Note that any standard UML statechart is a trivial STOCHART.



**Fig. 3.** A typical interaction of the ATM with its environment

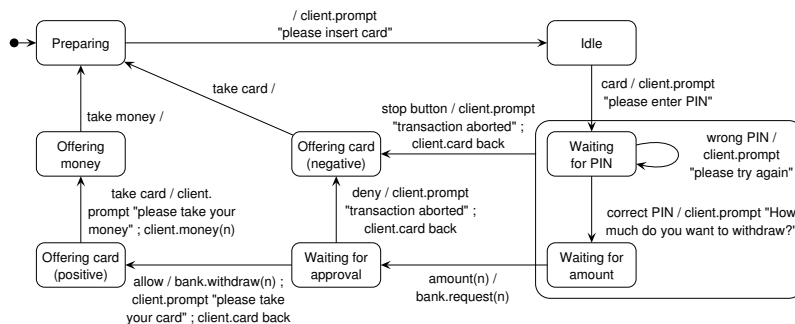


Fig. 4. The automatic teller machine behavior

imposed by the bank to ensure customer satisfaction. In the temporal logic CSL, this requirement can be formalized as

$$\text{prompt "please insert card"} \implies \mathcal{P}_{\geq p}(\diamond^{\leq t} \text{take money})$$

and can be fed into the CSL model checkers PROVER and ETMCC, which were the tools we used in our studies. The IOSA models associated with the STO-CHART specifications were generated in a semi-automatic way, and manually fed into the respective model checkers.

*Experiments.* In our experiments we considered two scenarios, a simple one, consisting of one client, one ATM and the bank, and a more complicated one with multiple clients, two ATMs and the bank. In the latter scenario, a mutual exclusion module (not shown in the figures) sequentializes the critical communication between bank (which is a shared resource) and ATMs. With respect to stochastic timing, our basic model assumed that all after-delays are given by exponential distributions with mean durations as listed in the respective figures. If only mean durations are available from domain analysis, the stochastic behavior is known to be reflected best by exponential distributions (since this class of distribution has maximal entropy). But we also experimented with other distributions in the context of PROVER.

*Results.* Some results of our experiments are shown in Fig. 6, where we evaluated the above CSL-requirement for increasing time points  $t$  and calculated the boundary probabilities  $p$  at which the requirement turns from being TRUE to being FALSE. For a pair  $(t, p)$  above a plot the requirement is FALSE, while for pairs below it is TRUE.

On the left we see the results obtained with ETMCC for both scenarios (one/two ATMs). We observe that in the second scenario, the probabilities are slightly lower than in the first one, with the maximum difference around 40 seconds (0.58 vs. 0.53). The reason is that although the throughput of the bank is higher with more ATMs, the individual customer perceived delay increases, with the bank being the bottleneck.

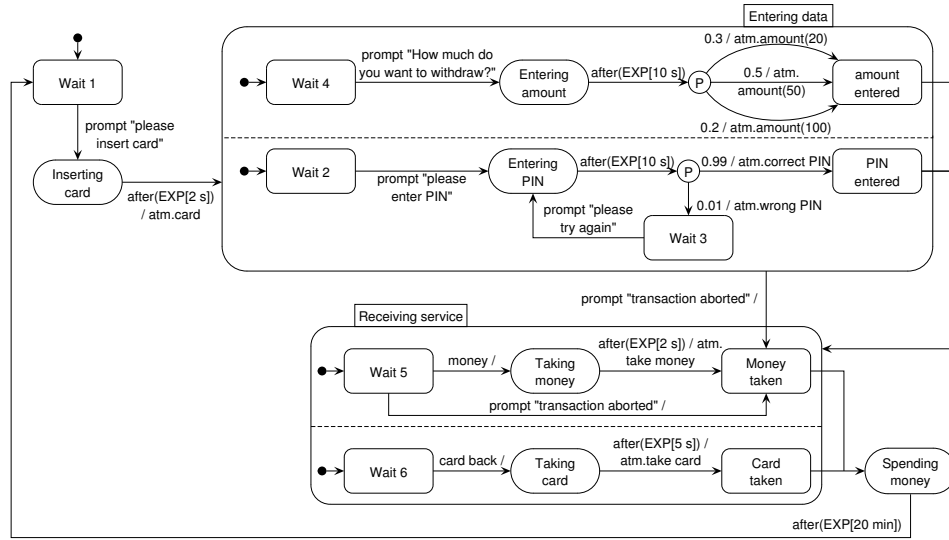


Fig. 5. The client's behavior

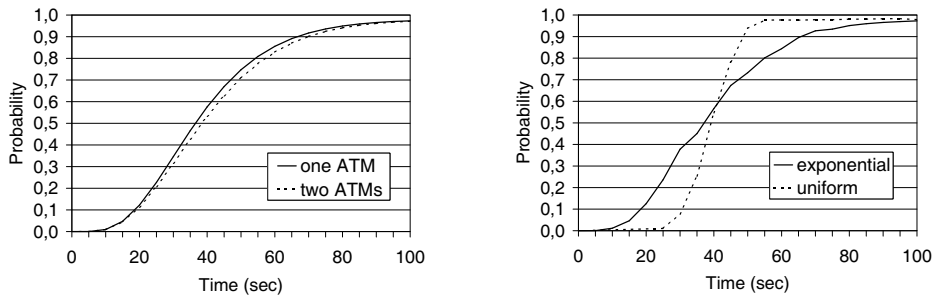


Fig. 6. The probability to withdraw money within  $t$  time units

On the right of Fig. 6 we plotted two graphs generated with PROVER, where the confidence was set to 0.999. The solid curve corresponds to the second scenario (two ATMs) studied with ETMCC, and the numerical results obtained with both tools are very close. This can be considered as a simple sanity check, indicating that the currently manual steps in the tool chain were performed correctly and that PROVER produces rather accurate results. The dashed curve is obtained when replacing all exponential distributions with uniform distributions (over an interval around the mean durations). More precise, each  $\text{EXP}[t]$  occurring in Fig. 6 is replaced by  $\text{UNIF}[t - \frac{1}{2}t, t + \frac{1}{2}t]$ . Since there is now some minimum time before significant behavior can take place (with nonzero probability), the calculated probability stays below the other plot for small times. For longer times we are on the other end of the uniform intervals involved, and it

is almost sure (except when the bank refuses the transfer) that the money will have been received.

## 6 Conclusion

This paper has introduced STOCCHARTS, a simple twofold extension of UML statecharts tailored to QoS modeling and prediction. We have laid out a formal mapping from STOCCHARTS to the IOSA model, a model based on timed, stochastic and probabilistic automata. This mapping provides the basis for faithful model-based QoS prediction.

Care has been taken to define a compositional semantics, making use of I/O automata. Most of the complexity of our semantic mapping is inherited from standard UML statecharts, while the extensions map rather smoothly on the IOSA model.

We have exemplified how QoS requirements on a STOCCHART can be model checked. The tool chain used in the model checking example is incomplete, and a few steps have been performed manually, namely the construction of an IOSA given a set of STOCCHART and parts of the translations from an IOSA to the input languages of the model checkers, in particular PROVER. We are currently closing the gaps, by integrating the STOCCHART formalism into the MODEST tool environment [7].

## References

1. M. Ajmone Marsan, G. Conte, and G. Balbo. A Class of Generalised Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems. *ACM Transactions on Computer Systems*, 2(2):93–122, 1984.
2. M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modeling with Generalized Stochastic Petri Nets*. John Wiley & Sons, 1995.
3. L. de Alfaro. *Formal Verification of Probabilistic Systems*. Ph.D dissertation, Stanford University, 1997.
4. R. Alur and D. Dill. A theory of timed automata. *Th. Comp. Sc.*, 126: 183–235, 1994.
5. C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen. Model-checking algorithms for continuous-time Markov chains. *IEEE Transactions on Software Engineering*, 29(6):524–545, 2003.
6. S. Bernardi, S. Donatelli, and J. Merseguer. From UML sequence diagrams and statecharts to analysable Petri net models. In *Workshop on Software and Performance (WOSP)*, pp. 35–45, ACM Press, 2002.
7. H. Bohnenkamp, H. Hermanns, J.-P. Katoen, and R. Klaren. The Modest modeling tool and its implementation. In P. Kemper and W.H. Sanders, editors, *Computer Performance Evaluation*, LNCS 2794, 2003.
8. C. Cavenet, S. Gilmore, J. Hillston, and P. Stevens. Performance modelling with UML and stochastic process algebra. In: *UK Performance Engineering Workshop (UKPEW)*, pp. 16 Univ. of Glasgow, UK, 2002.
9. C. Constantinescu. Impact of deep submicron technology on dependability of VLSI circuits. In *Dependable Systems and Networks*, pp. 205–209. IEEE CS Press, 2002.
10. W. Damm, B. Josko, H. Hungar, and A. Pnueli. A compositional real-time semantics of state machine designs. In *Compositionality : the Significant Difference (COMPOS)*, LNCS 1536: 186–238, 1998.

11. P. R. D'Argenio, J.-P. Katoen and E. Brinksma. An algebraic approach to the specification of stochastic systems (extended abstract). In *Programming Concepts and Methods*, pp. 126–147, Chapman & Hall, 1998.
12. R. Eshuis and R. J. Wieringa. Requirements-level semantics for UML statecharts. In *Formal Methods for Open Object-Based Distributed Systems (FMOODS)*, pp. 121–140, Kluwer, 2000.
13. S. Gnesi, D. Latella, and M. Massink. A stochastic extension of a behavioural subset of UML statechart diagrams. In *Symposium on High-Assurance Systems Engineering (HASE)*, pp. 55–64. IEEE CS Press, 2000.
14. N. Götz, U. Herzog and M. Rettelbach. Multiprocessor and Distributed System Design: The Integration of Functional Specification and Performance Analysis Using Stochastic Process Algebras. In *Tutorial Proc. PERFORMANCE '93*, LNCS 729: 121–146, 1993.
15. D. Harel and E. Gery. Executable object modeling with statecharts. *IEEE Computer*, 30(7):31–42, 1997.
16. D. Harel and A. Naamad. The STATEMATE semantics of statecharts. *ACM Transactions on Software Engineering and Methodology*, 5(4):293–333, 1996.
17. H. Hermanns, J.-P. Katoen, J. Meyer-Kayser and M. Siegle. A Markov chain model checker. *J. on Software Tools and Technology Transfer*, 4(2):153–173, 2003.
18. J. Hillston. *A Compositional Approach to Performance Modelling*. Cambridge Univ. Press, 1996.
19. G. Huszerl, I. Majzik, A. Pataricza, K. Kosmidis, and M. Dal Cin. Quantitative analysis of UML statechart models of dependable systems. *The Computer J.*, 45(3):260–277, 2002.
20. D. N. Jansen, H. Hermanns, and J.-P. Katoen. A probabilistic extension of UML statecharts : specification and verification. In *Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT)*, LNCS 2469: 355–374, 2002.
21. D. N. Jansen. Extensions of Statecharts with Time, Probability, and Stochastic Timing. Ph.D thesis, Univ. of Twente, October 2003.
22. B. Jonsson *et al.*. Component-based design and integration platforms : Draft roadmap of IST-2001-34820. *Advanced Real-Time Systems*, 2003.
23. P. King and R. Pooley. Derivation of Petri net performance models from UML specifications of communications software. In *Computer Performance Evaluation (TOOLS)*, LNCS 1786: 262–276, 2000.
24. M. Z. Kwiatkowska, G. Norman, and D. Parker. Probabilistic symbolic model checking with PRISM: A hybrid approach. In *Tools and Algorithms for the Construction and Analysis of Algorithms (TACAS)*, LNCS 2280: 52–66, 2002.
25. C. Lindemann, A. Thümmler, A. Klemm, M. Lohmann, and O. P. Waldhorst. Performance analysis of time-enhanced UML diagrams based on stochastic processes. In *Workshop on Software and Performance (WOSP)*, pp. 25–34, ACM Press, 2002.
26. N. Lynch and M. Tuttle. An introduction to input/output automata. *CWI Quarterly*, 2(3): 216–246, 1989.
27. N. Lynch and F. Vaandrager. Forward and Backward Simulations : II. Timing-Based Systems. *Information and Computation*, 128: 1–25, 1996.
28. P. Mitton and R. Holton. PEPA performability modelling using UML statecharts. In *UK Performance Engineering Workshop (UKPEW)*, 15 pp. Univ. of Durham, UK, 2000.
29. A. N. Shiryaev. *Probability*, vol. 95 of *Graduate Texts in Mathematics*. Springer, New York, 1996.
30. Object Management Group. *UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms : Request for Proposal*. 2002.
31. Object Management Group. *UML Profile for Schedulability, Performance, and Time Specification*. 2002.
32. R. Segala and N.A. Lynch. Probabilistic simulations for probabilistic processes. *Nordic J. of Computing*, 2(2): 250–273, 1995.
33. D. Tennenhouse. Proactive computing. *Comm. of the ACM*, 43(5): 43–50, 2000.
34. S.-H. Wu, S. A. Smolka and E. W. Stark. Composition and behaviors of probabilistic I/O automata. *Th. Comp. Sc.*, 176(1/2): 1–38, 1997.
35. H. Younes and R. Simmons. Probabilistic verification of discrete event systems using acceptance sampling. In: *Computer-Aided Verification (CAV)*, LNCS 2404: 223–235, 2002.