From EMPA to GSMPA: Allowing for General Distributions

Mario Bravetti Marco Bernardo Roberto Gorrieri

Università di Bologna, Dipartimento di Scienze dell'Informazione
Mura Anteo Zamboni 7, 40127 Bologna, Italy
E-mail: {bravetti, bernardo, gorrieri}@cs.unibo.it

Abstract

The stochastically timed process algebra GSMPA enhances the expressiveness of the stochastically timed process algebra EMPA by allowing arbitrarily distributed durations to be directly modeled. To accomplish this, the mechanism of action identification as well as the adoption of the preselection policy for alternative actions have been introduced. A restricted version of GSMPA is presented together with the definition of its integrated semantics, from which it is possible to derive by projection a functional semantics and a performance semantics in the form of an extended generalized semi-Markov process (EGSMP). The class of EGSMPs is an extension, we have introduced, of the well-known class of GSMPs. Therefore already established theoretical results for GSMP, such as their solution through the notion of insensitivity, can be applied for performance evaluation purposes.

1 Introduction

The widely recognized desirability of integrating the functional and performance description techniques of concurrent systems (see, e.g., [16, 6]), and the advantages of doing that in the field of process algebras because of compositionality, led to the definition of EMPA [2, 3]. EMPA is a stochastically timed process algebra where only exponentially distributed as well as zero durations can be described. The reason for this restriction is twofold: the underlying performance models turn out to be homogeneous continuous time Markov chains which can be analyzed with several well known methods, and the semantics for the calculus can be defined in the *pure* interleaving style due to the memoryless property of exponential distributions. Actually, in EMPA the combined use of exponentially timed and immediate actions allows phase-type [15] distributed actions to be represented, which is quite useful since many frequently occurring distributions are or can be approximated by phase-type ones. As one might expect, the price to pay is a state space growth. In order to be able to *directly* express durations with *arbitrary distributions*, thereby avoiding approximations through combination of actions, we have developed the stochastically timed process algebra *GSMPA (Generalized Semi-Markovian Process Algebra)* which will be presented in this paper. In particular, our purpose is to *integrate deterministic and probabilistic durations*, that often are both required for the specification of even simple real systems, in order not to oblige the system designer to choose between deterministically timed algebras and stochastically timed algebras. The introduction of deterministic distributions causes new problems because: they do not have the memoryless property, their support is not $\mathbb{R}_+$ and they are not continuous (and so actions may terminate at the same instant). The problems that arise are the same as we have with completely arbitrary distributions and so we treat directly the general case. The performance model of systems described with GSMPA belongs to the class of *EGSMPs (Extended Generalized Semi-Markov Processes)*, an extension of the well known class of *GSMP (Generalized Semi-Markov Process)* [14] from which the algebra derives its name. We have introduced this extension in order to be able to express the contemporaneous termination of actions due to *non-continuous distributions* such as the deterministic ones, as well as the abort of actions. [1] As a consequence, the underlying performance models can be sometimes analyzed by exploiting well known

[1] The class of EGSMPs is also an extension of GSMPs with interruption presented in [11].

results for GSMPs, such as the notion of *insensitivity* [14, 9, 11, 10], which make it possible to compute the steady state probability of a GSMP by solving a corresponding Markov chain.

In order to allow for arbitrarily distributed durations GSMPA is basically obtained from EMPA by:

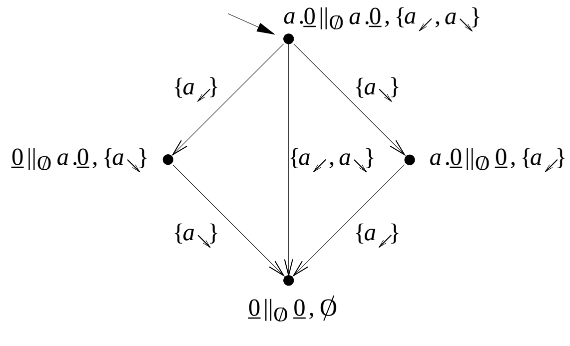
- adopting the mechanism of *action identification*, and
- adopting the *preselection policy* for *alternative* actions.

Action identification is the key concept that allows the passage to general distributions. This concept derives from the definition of GSMPs. In order to understand why action identification is necessary, we recall that in EMPA, thanks to the memoryless property of exponential distributions, it is possible to consider an action in execution in a state as beginning in that state, even if under execution when the state is entered. With general distributions this simplification is no longer possible, and it is necessary to continue the execution of an action from the accomplishment level it had previously reached. On the other hand, a given state of a concurrent system can be considered, in a realistic way, as a set of parallel activities at different accomplishment levels, each with its own age quantifying the amount of time already spent in execution. To get a description of this kind, we must *identify* the actions by giving them names (*identifiers*) which single out the different independently aged activities they represent. With each system state we then associate the set of the identifiers of the actions in execution in it. When a state is entered *each activity* of its set *is started from its current age*, i.e. from the accomplishment level the activity (singled out by the corresponding identifier) had reached in previous states. The system sojourns in the state until one activity (or more contemporaneously) is completed and the related action terminates. For instance, consider the GSMPA specification:

$$a.\underline{0} \,\|_{\emptyset}\, a.\underline{0}$$

$$PDF_{\bullet\nearrow}(a) = DET(5)$$
$$PDF_{\bullet\searrow}(a) = GEOM(.5)$$

which represents a system that executes in parallel (operator "$\_\|_{\emptyset}\_$") two actions with the same type a, where the action of the process to the left of $\|_{\emptyset}$ ($\bullet\nearrow$) has deterministic duration 5 while the action of the process to the right ($\bullet\searrow$) has geometrically distributed duration with parameter 0.5. The reason for this syntax and for this separated specification of durations will be explained in the following sections. In the underlying semantic model, the actions are identified by expressing their *location*, i.e. their position with respect to parallel composition operators. The action to the left of $\|_{\emptyset}$ is identified with $a_{\nearrow}$, the other with $a_{\searrow}$. The integrated semantics of the system under examination has the same structure as a GSMP, and is composed of a labeled transition system (LTS) and a description of durations. A simplified version of the semantics is the following:



$$PDF(a_{\nearrow}) = DET(5)$$
$$PDF(a_{\searrow}) = GEOM(.5)$$

The states are labeled with the identifiers of actions in execution, and the transitions are labeled with the identifiers of terminating actions. The transition from the initial state to the deadlocked state represents the simultaneous termination of both actions. The identifiers distinguish the two actions showing that they are different activities and when one of them terminates, the other resumes execution from the age *it* (singled out by its identifier) had reached in the previous state. It is worth noting that the durations are statically associated with action identifiers because we follow the GSMP structure. In this way in any system state the residual time to the completion of an activity can be determined by its current age and the distribution of duration statically associated with the activity.

The second modification necessary to allow for general distributions is the adoption of a suitable *execution policy* for actions. In EMPA the *race policy* is used and all the actions executable in a system state are put in execution until one of them terminates. This policy is surely correct in case of *parallel executable actions*, e.g. a and b in $a.\underline{0} \,||_{\emptyset}\, b.\underline{0}$, because it establishes only the instant of termination of the action that terminates first. In case of *alternative actions*, e.g. a and b in $a.\underline{0} + b.\underline{0}$, instead, the race policy is used to choose one of them. This leads to a model that describes the system as seen by an external observer but do not represent correctly the system resources: in the example, actions a and b are in contemporaneous execution on the same sequential process. In GSMPA, instead, the choice among alternative actions is made by means of the *preselection policy*. In order to enforce this policy, a priority and a weight are associated with each action. According to them a preliminary selection among alternative actions is done before actual execution. For instance in $<a, 2, 2>.\underline{0} + <b, 2, 3>.\underline{0} + <c, 1, 1>.\underline{0}$, *first* c is discarded since it has lower priority (1) than a and b (2) and a probabilistic selection between a and b is made according to their weights (2 and 3), and *then* the selected action is executed. In this way:

- Alternative actions are really treated as such and consequently GSMPA is a *resource aware* language correctly describing the structure of systems.
- Selections are represented directly, without e.g. the need of employing actions with a (prioritized weighted) null duration like in EMPA, thereby achieving a separation between choices and durations.

Moreover, the use of the preselection policy is very suitable (if not necessary) when *general distributions* are considered, in that it allows:

- An *effective use of identifiers*, because of the possibility to select among equally identified actions. The suitable identification mechanism based only on locations of actions can be used since alternative actions are not executed contemporaneously and two alternative actions are allowed to refer actually to the same activity.
- The *implicit treatment of the contemporaneous termination case* without the necessity of any special mechanism for selection, because the termination of each action has independent effects in the term (as can be seen in the LTS underlying the term $a.\underline{0} \,||_{\emptyset}\, a.\underline{0}$ presented previously).

Finally, GSMPA is endowed with special kinds of actions called *suspensions*. They do not represent actual resource occupying actions, but only waits that the system must perform before it can assume some behavior (they represent e.g. timers). They do not improve the language expressiveness, but allow for more elegant and simpler specifications. In this paper, due to lack of space, suspensions are not considered and a restricted version of GSMPA, called r-GSMPA, is presented. For an extensive presentation of GSMPA and the new class of EGSMPs, the reader is referred to [4].

The paper is organized as follows. In Sect. 2 we define how a system is specified with r-GSMPA and we give the syntax of its terms and the meaning of its operators. In Sect. 3 we define the integrated semantics for r-GSMPA. In Sect. 4 we briefly describe its functional semantics and its performance semantics. Finally in Sect. 5 we report some concluding remarks including comparisons with related work and directions for future research.

2 Specification of a system

The specification of a system consists of the representation of its structure and of the description of the activities it executes during its life. The execution of an activity (singled out by an identifier) is represented by the execution of an action with that identifier. Once started, an action stays in execution in every successive state until it:

- *Terminates* and the activity it represents is completed.

- Is *interrupted*. In this case the corresponding activity may:
  - *Lose its age*: when the activity will return in execution it will restart from the beginning. This modality is called *interruption with abort* (known in the literature as preemptive restart with resampling).
  - *Keep its age*: the activity is said to be *frozen*, and when it will return in execution it will continue from where it was interrupted. This modality is called *interruption with continuation* (known in the literature as preemptive resume).

In the case of interruption with continuation *the activity is continued by a following action with the same identifier*. In general, therefore, the execution of an action represents the execution of the related activity *starting from its current age*. Since the execution of an action may represent the completion of an activity started by a previous action, we can no more specify the performance aspect of the system by associating durations directly with actions within algebraic terms (as in EMPA). In GSMPA performance is specified associating statically a distribution of duration with each activity (identifier) executable by the system during its life in a compositional way. On the other hand this structure of separated specification reflects that of the semantic models of GSMPA we described in the introduction (as can be seen by the example therein presented), and hence that of GSMPs.

The specification of a system is then composed of the specification of its behavior (a term of GSMPA which describes the execution of actions) and the specification of the duration of its activities. These will be presented, respectively, in the following Sect. 2.1 and 2.2.

2.1 Specification of system behavior

2.1.1 Representation of actions

Each *action* is represented as $<a^m_{loc}, l, w>$ and consists of a *type* a, a *location* loc, a *modality* m, a *priority level* l and a *weight* w. [2]

The action *type* specifies which kind of action can be seen by an external observer (e.g. message transmission). The set of action types *AType* is ranged over by $a, b, c, \ldots$. Since we adopt the CSP [13] synchronization policy for the parallel composition operator, the actions of the whole system (hereafter called *top level actions*) can be formed by the cooperation of several *local actions* with the same type, each executed by a single sequential process.

The *location* singles out the sequential processes that cooperate in the execution of a top level action by expressing their locations with respect to parallel composition operators. The type and the location of a top level action together constitute the *identifier* of the action which establishes the corresponding activity. In order to be able to express different activities with the same type which are executed by the same sequential process, we associate with each local action a natural number and we write such numbers inside the locations.

Definition 2.1 The set Loc of *action locations* is generated by the following syntax

$$loc ::= i \mid \swarrow loc \mid \searrow loc \mid \langle loc | loc \rangle$$

where $i \in \mathbb{N}_+$. [3] ■

Definition 2.2 The set AId of (top level) *action identifiers* is defined by:

$$AId = AType \times Loc$$ ■

As regards local actions the location reduces to a simple natural number.

Definition 2.3 The set $LAId$ of *local action identifiers* is defined by:

$$LAId = AType \times \mathbb{N}_+$$ ■

[2] The location and the modality of an action are associated with its type only for representation convenience. In this way writing an action can be shortened by omitting its location and/or its modality (which are given a default value) even if the priority (and possibly the weight) must be specified.

[3] If a natural number which should appear in a location is omitted, it is intended to have the default value 1. If the location consists only of the number 1, it is completely omitted.

Note that in each sequential process local actions of the same type can be indipendently enumerated without ambiguity: e.g., in $a_2 \,||_{\{a\}}\, a_2$ the two actions correspond to the two different activities $a_{\swarrow 2}$ and $a_{\searrow 2}$.

Example 2.4 Consider term $(a.E_1 \,||_{\emptyset}\, b_2.E_2) \,||_{\{b\}}\, b_3.E_3$. Then:

- $a_{\swarrow\swarrow}$ is the identifier the action formed by the single local action identified with a (a_1) executed by the leftmost sequential process.
- $b_{(\searrow 2|3)}$ is the identifier of the action formed by the cooperation of the two local actions identified with b_2 and b_3. ■

The modality m of an action expresses its *interruption modality* as well as its *visibility*. We would like to point out that in GSMPA we cannot use a distinguished unique action type for invisible actions such as τ in EMPA, because in case some actions of a process are made hidden to the environment (by means of the hiding operator "$\_/L$" presented in Sect. 2.1.2) we must preserve the identifiers of actions (which include the action types) in order to keep the identification of actions belonging to the internal behavior of the process so that its performance is modeled correctly. As a consequence, visibility reduces to an action attribute.

Definition 2.5 The set *AMode* of *action modalities* is defined by

$$AMode = \{n, a, c, hn, ha, hc\}$$

where *interruption modality* is:

- n for *non-interruptable* (default value);
- a for *interruptable via abort*;
- c for *continuable after interruption*,

and *visibility* is:

- h for *hidden*, meaning that the action must be executed internally;
- none for *visible*, meaning that the action can co-operate with other actions (default value). ■

The *priority level* $l \in \mathbb{N}_+$ and the *weight* $w \in \mathbb{R}_+$ of an action are used, as stated in Sect. 1, for carrying out the selection among alternative actions according to the preselection policy (the default value for both priority level and weight is 1).

Definition 2.6 The set *LAct* of *local actions* is defined by

$$LAct = LAId \times AMode \times \mathbb{N}_+ \times \mathbb{R}_+$$

and the set *Act* of *actions* is defined by

$$Act = AId \times AMode \times \mathbb{N}_+ \times \mathbb{R}_+$$

■

2.1.2 Syntax of terms and informal semantics of operators

Let *Const* be a set of *constants* ranged over by $A, B, C, \ldots$, and let $ARFun = \{\varphi : AType \longrightarrow AType\}$ be a set of *action relabeling functions*.

Definition 2.7 The set $\mathcal{L}$ of *process terms* of r-GSMPA is generated by the following syntax

$$E ::= \underline{0} \mid <a_i^m, l, w>.E \mid E/L \mid E[\varphi] \mid E + E \mid E \,||_S\, E \mid A$$

where $<a_i^m, l, w> \in LAct$ and $L, S \subseteq AType$. Set $\mathcal{L}$ will be ranged over by $E, E', E'', \ldots$. We denote by $\mathcal{G}$ the set of closed and guarded terms of $\mathcal{L}$. ■

The *null term* $\underline{0}$ has the usual meaning of termination.

The *prefix operator* "$<a_i^m, l, w>.\_$" denotes the sequential composition of an action and a term: term $<a_i^m, l, w>.E$ can execute a local action with identifier a_i, modality m, priority level l and weight w, and behaves as term E after the completion of the action.

The *hiding operator* "$\_/L$" changes the visibility of the actions: term E/L behaves as term E except that actions whose type belong to L get visibility h.

The *relabeling operator* "$\_[\varphi]$" modifies the type of the actions: term $E[\varphi]$ behaves as term E except that the type of each executed action is modified according to φ. Note that also the type of hidden actions can be changed by this operator.

The *alternative composition operator* "$\_+\_$" expresses a choice between two terms: term $E_1 + E_2$ behaves as *either* term E_1 *or* term E_2 depending on whether an action of E_1 or an action of E_2 is selected for execution. The choice is made according to the preselection policy, hence the priority levels and the weights of the executable actions of the two terms are taken into account. In this way actions are executed according to only E_1 (or E_2) and the two terms cannot execute actions contemporaneously, as happens, e.g., in EMPA. Initially the term not selected is not discarded since the choice is not definitive. The choice between E_1 and E_2 is definitively resolved only when an action executed by one of them terminates. This allows for a mechanism for the interruption of actions: if, in the term not initially selected, a new action is later enabled, it may prevail through a new selection over the actions already in execution in the other term and consequently interrupt them. The interruption mechanism will be explained in detail in Sect. 2.1.3 and 3.1.

The *parallel composition operator* "$\_\,\|_S\,\_$" expresses the concurrent execution of two terms according to the CSP [13] synchronization policy, thus the two terms must co-operate when executing visible actions of the same type belonging to S, whilst they must execute independently all the other actions. Given term $E_1 \,\|_S\, E_2$, in case of cooperation of a visible action of E_1 with type $a \in S$ and location loc_1 and a visible action of E_2 with the same type a and location loc_2, the resulting action has obviously the same type a, has location $\langle loc_1 | loc_2 \rangle$, has interruption modality computed by applying function $mode : (AMode \times AMode) \longrightarrow\!\!\!\!\!\!\!\circ\;\; AMode$ defined in Table 3 to the original interruption modalities, is visible, has priority level given by the *sum* of the original priority levels, and has weight given by the *product* of the original weights. In case of an action executed independently by E_1 (E_2) which is invisible or visible with type $a \notin S$ and has location loc_1 (loc_2), the resulting action has obviously the same type, has the location $\swarrow loc_1$ ($\searrow loc_2$) and the other attributes are kept unchanged.

Finally, *constant* A behaves like the only term E such that $A \stackrel{\Delta}{=} E$.

2.1.3 Execution policy: preselection policy and race policy

The execution policy establishes what happens when, because of operators like parallel composition or alternative composition, several actions are simultaneously executable in a system state. Languages like EMPA are based on the *race policy*: in such situations all executable actions (even if representing alternative behaviors) are put in execution until the termination of one of them. In GSMPA alternative actions cannot be contemporaneously executed and the *preselection policy* is used to make a preliminary selection among them.

Definition 2.8 Two actions are said to be *conflicting* if they are executable by means of two alternative terms or share some local actions. ■

It is worth noting that conflicting actions are generated not only by the *choice among alternative behaviors* as for a and b in $a.\underline{0} + b.\underline{0}$, but also by the *choice for cooperation* as for actions identified by $a_{\langle | \swarrow \rangle}$ and $a_{\langle | \searrow \rangle}$ in $a.\underline{0} \,\|_{\{a\}} (a.\underline{0} \,\|_{\emptyset}\, a.\underline{0})$.

According to the preselection policy, a maximal subset of actions executable in parallel (i.e. conflict free) is chosen from the set of all the executable actions in a system state, and they are then executed according to the race policy until the termination of one (or more simultaneously) of them. At this point we have a new set of executable actions and a new selection-execution cycle is performed. It is worth noting that the actions still in execution at the end of the race are considered simply executable in the new state. In this way it is possible to model the interruption of such actions if they are no longer selected. As concerns selection among executable actions in a state, we must distinguish among:

- *Resumable actions*: actions under execution for which it must be decided whether to resume their execution or interrupt them.
- *New actions*: actions that are executable for the first time.
- *Discarded actions*: actions that formerly became executable, but that are not in execution in that discarded.

The selection of the actions to be executed is done by choosing one executable action at a time. At every successive stage one action, selected among a set of *selectable actions*, is put into execution. The set of selectable actions of a given stage is formed by executable actions *not in conflict* with actions already in execution (i.e. selected in previous stages). The selection phase finishes when the set of selectable actions becomes empty. In this way a maximal subset of parallel executable actions is obtained.

A generic system state thus determines:

- *Advancing actions*: actions put into execution. They are executed simultaneously according to the *race policy*.
- *Selectable actions*: resumable, new or discarded actions that *are not in conflict* with actions in execution and can be chosen according to the *preselection policy*.

Moreover, in order to obtain the described behavior, we establish that when in a state there are both kinds of actions, *preselection policy has priority over race policy* and a selectable action is put in execution causing an *immediate* state change. In this way first we make all possible action selections and then we actually execute together the chosen actions.

2.2 Specification of durations

The durations of the activities executed during system life are specified in a compositional way. The system specification must include the description of the duration of every local action of every sequential process as well as the description of the way in which the duration of an action resulting from the cooperation of several actions is computed. First let us identify sequential processes by considering their location with respect to parallel composition operators.

Definition 2.9 The set PId of *process identifiers* is generated by the following syntax

$$pid ::= \bullet \mid pid\swarrow \mid pid\searrow$$

where $\bullet$ denotes the root of the syntax tree of the term describing the whole system. ■

Example 2.10 Consider term $(E_1 \,||_S\, E_2)\,||_{S'}\, E_3$. Then $\bullet\swarrow\searrow$ denotes process E_2. ■

The specification of a system described by term $E \in \mathcal{G}$ must include:

- For every sequential process pid of E a function

  $$PDF_{pid} : LAId \twoheadrightarrow PDF$$

  that associates with every local activity executable by process pid during system life [4] the corresponding distribution of duration.
- For any action type $a \in AType$ belonging to a synchronization set occurring in E a function

  $$PDF_a : (PDF \times PDF) \longrightarrow PDF$$

  that determines the duration of an activity derived by the cooperation of two activities of type a. This operator determines the cooperation paradigm used for action type a and can be *arbitrary*. A suitable function, however, should be at least commutative and associative. For an overview of cooperation paradigms see [12].

[4] The set of local activities executed by a sequential process is easily determined from the specification of its behavior.

3 Integrated semantics for GSMPA

The integrated semantics for GSMPA has the same structure as a GSMP, i.e. it formalizes the representation of system behavior (derived from the specification of system behavior) and the description of durations of activities (derived from the specification of durations).

3.1 Representation of system behavior

We now define the formal integrated semantics for r-GSMPA according to Sect. 2.1 in the form of a labeled transition system (LTS) with two types of transitions.

3.1.1 States of the LTS

The *states* of the LTS are *labeled*. The actual state consists of a *decorated term* F, and the label is composed of: the set *IntA* of *interrupted actions* and the set *Exec* of *activities under execution*. [5]

The *decorated term* represents the *present and future behavior of the system*. The syntax of decorated terms is obtained by extending that of r-GSMPA terms with *decorated actions* in order to single out the execution stage of actions forming the present behavior so as to denote if they are advancing actions or selectable actions of the three kinds: resumable, discarded, new. Each *decorated action* is represented by $<\overset{x}{a}{}^{m}_{loc}, l, w>$ where x is the *decoration* of the action and a, loc, m, l, w have the usual meaning.

Definition 3.1 The set *Dec* of *action decorations* is defined by:

$$Dec = \{\hat{\ }, \bar{\ }, \tilde{\ }, \_\}$$

where "_" denotes the absence of decoration. ∎

As regards local actions:

- $<\hat{a}^{m}_{i}, l, w>$ denotes a *advancing* local action;
- $<\bar{a}^{m}_{i}, l, w>$ denotes a *resumable* local action;
- $<\tilde{a}^{m}_{i}, l, w>$ denotes a *discarded* local action;
- $<a^{m}_{i}, l, w>$ denotes a *new* local action.

How decorations are composed in case of cooperation and the meaning they assume for top level actions will be explained in the following.

Definition 3.2 The set *DLAct* of *decorated local actions* is defined by

$$DLAct = LAId \times Dec \times AMode \times \mathbb{N}_+ \times \mathbb{R}_+$$

and the set *DAct* of *decorated actions* is defined by

$$DAct = AId \times Dec \times AMode \times \mathbb{N}_+ \times \mathbb{R}_+$$

∎

Definition 3.3 The set $\mathcal{L}_{dec}$ of *decorated process terms* of r-GSMPA is generated by the following syntax

$$F ::= \underline{0} \mid <\overset{x}{a}{}^{m}_{i}, l, w>.E \mid F/L \mid F[\varphi] \mid F + F \mid F \parallel_S F \mid A$$

where $<\overset{x}{a}{}^{m}_{i}, l, w> \in DLAct$ and $L, S \subseteq AType$. The set $\mathcal{L}_{dec}$ will be ranged over by $F, F', F'', \ldots$. We denote by $\mathcal{G}_{dec}$ the set of closed and guarded terms of $\mathcal{L}_{dec}$. ∎

As can be seen from production $F ::= <\overset{x}{a}{}^{m}_{i}, l, w>.E$, since $E \in \mathcal{L}$ only local actions belonging to present behavior are decorated.

Let us now describe the components of the label of a state.

The set $IntA \subseteq AId$ of the *identifiers of interrupted actions* contains the identifiers of the resumable actions of the current selection phase that have not re-started execution yet, and are therefore *considered*

[5] The distinction between the actual state and the label is done because the behavior of the system must be described by the structure and the labels of the LTS abstracting from the representation of states (as for classic process algebras).

interrupted in the state. This holds even for non-interruptable actions (interruption modality n), although they will certainly re-start execution, as we will see.

The set *Exec* of *activities under execution* determines the activities with an age. It contains for each advancing activity the identifier and associated modality of the corresponding action, and for each frozen activity its identifier with associated the new modality f.

Definition 3.4 The set *EMode* of *modalities of activities under execution* is defined by

$$EMode = AMode \cup \{f\}$$ ■

After denoting by $MIds = \{mids \mid mids : AId \twoheadrightarrow EMode\}$ the family of the sets of identifiers equipped with modality, we have $Exec \in MIds$. An element of *Exec* is denoted by a^m_{loc} so that the modality can be omitted.

In conclusion the set of states of the LTS is $LabS = \mathcal{G}_{dec} \times L_S$ where $\mathcal{G}_{dec}$ is the set of actual states and $L_S = \mathcal{P}(AId) \times MIds$ is the set of state labels ranged over by $(IntA, Exec)$.

Representation of advancing actions

A top level action is advancing in a given state if its decoration is "ˆ" (i.e. each of its composing local actions is decorated with "ˆ") and its identifier with modality is in *Exec*.

It is worth noting that, since actions advancing contemporaneously in a state cannot be in *conflict* (see Sect. 2.1.3), every sequential process executes at most one single local action and it follows that GSMPA is resource aware. A trivial consequence is that every advancing action has a different identifier. When there are several advancing actions, both decorations of local actions in the term and identifiers in *Exec* are necessary to single out the local actions relative to each action.

Example 3.5 Consider the following states:

- $a.\underline{0} + \hat{a}.\underline{0}, \emptyset, \{a\}$: the identifier in *Exec* does not clarify by itself which local action of the term is advancing, hence the decoration in the term is necessary.
- $(\hat{a}.\underline{0} \,||_{\emptyset}\, \hat{a}.\underline{0}) \,||_{\{a\}} (\hat{a}.\underline{0} \,||_{\emptyset}\, \hat{a}.\underline{0}), \emptyset, \{a_{\langle\swarrow|\swarrow\rangle}, a_{\langle\searrow|\searrow\rangle}\}$: the decorations by themselves do not clarify which local actions are relative to each of the two advancing actions, hence identifiers in *Exec* are necessary. ■

Representation of selectable actions

Selectable actions are the actions executable by the term that are not in conflict with any advancing action. The decoration of a top level selectable action is determined from the decorations of its composing actions, according to the binary operator $* : Dec \twoheadrightarrow Dec$ defined in Table 3, as follows.

The decoration of a top level selectable action is:

- "¯" if it is composed of local actions all decorated with "¯";
- "˜" if it is composed of local actions, at least one of which is decorated with "˜", and the others (if any) are decorated with "¯";
- "_" (absence of decoration) if it is composed of local actions, at least one of which is not decorated at all, and the others (if any) are arbitrarily decorated.

The action is classified as:

- *Resumable* if it has decoration "¯" and its identifier is in *IntA*. [6]
- *Discarded* if:
  - either it has decoration "¯" and its identifier is not in *IntA*,
  - or it has decoration "˜".
- *New* if it has decoration "_".

[6] Resumable selectable actions are determined through decorations "¯" and the set *IntA* in the same way as advancing actions are determined through decorations "ˆ" and the set *Exec*.

3.1.2 Transitions of the LTS

The two types of transition of the LTS are related to the two execution policies employed by GSMPA: *termination transitions* to *race policy* and *choice transitions* to *preselection policy*.

Termination transitions

Termination transitions represent the contemporaneous termination of a set of *advancing actions* according to the *race policy*. The race policy is applied to a nonempty set of advancing actions and consists of the parallel execution of such actions until the termination of one (or more simultaneously) of them.

Owing to the contemporaneous termination of a set of actions, the following changes are applied to the system state:

- The decorated term is modified by:
  - Eliminating the prefix operators related to the local actions above, thus causing successive behaviors to be undertaken.
  - Eliminating the behaviors alternative to those that include the local actions forming the terminated actions.
- Set *Exec* is modified by removing the identifier with modality of every terminated action.

Upon action termination, the following additional changes, related to *previously undertaken behaviors*, are applied to the system state.

- The decorated term is modified by:
  - Decorating with “¯” all the local actions decorated with “ˆ”.
  - Decorating with “˜” all the other local actions.
- Set *IntA* is modified by inserting the identifier of every non terminated advancing action.
- Set *Exec* is modified by:
  - Removing the identifier with modality of every non terminated advancing action whose modality is n or a.
  - Changing to f the modality of every non terminated advancing action whose modality is c.

This results in:

- Transforming each advancing action into a resumable action. Such actions are all considered interrupted in the reached state.
- Considering as discarded all other actions executable according to behaviors previously undertaken by the system.

Note that local actions belonging to *new behaviors undertaken by the system* are kept without decorations. In this way actions composed by at least one local action belonging to a new behavior are new actions.

Choice transitions

Choice transitions represent the choice of a *selectable action* according to the *preselection policy*. The preselection policy is applied to a nonempty set of selectable actions and consists of the selection of one of them according to their priority levels and weights: only actions with the maximum priority level are considered, and the selection among them is carried out by giving each of them a probability proportional to its weight.

After choosing a selectable action, that action is put in execution and the system moves immediately to a new state determined as follows:

- The decorated term is modified by decorating with "^" all the local actions forming the chosen action.
- Set $IntA$ is modified by removing the identifier of the chosen action *in the case this action is resumable* because the action resumes execution and is no longer *considered* interrupted; otherwise $IntA$ remains unchanged.
- Set $Exec$ is modified by inserting the identifier with modality of the chosen action.

To be more precise, when applying the preselection policy, there are two exceptions given by resumable actions with interruption modality n and discarded actions.

The problem with resumable actions with interruption modality n is that they must certainly re-start execution and therefore must prevail over any other new or discarded action. To achieve this, they are taken to have priority level ∞ during the selection phase.

The problem with discarded actions is that of *system stability*: the interruption of an action should not be caused by another action, over which the former action previously prevailed. If this were not the case, selections already made could be re-made (possibly with a different outcome if probabilistic) just because the system is in a selection phase caused by events extraneous to the initial decision.

Example 3.6 Consider term

$$(a^a.\underline{0} + b^a.\underline{0}) \,||_{\emptyset}\, c.\underline{0}$$

It makes an initial probabilistic selection between a (action are singled out through types) and b, which have the same default priority level 1; whilst c starts certainly execution. Suppose that a is selected thereby reaching the following state

$$(\hat{a}^a.\underline{0} + b^a.\underline{0}) \,||_{\emptyset}\, \hat{c}.\underline{0}, \emptyset, \{a^a_{\swarrow}, c^a_{\searrow}\}$$

Then suppose that c terminates before a thereby reaching the following state

$$(\bar{a}^a.\underline{0} + \tilde{b}^a.\underline{0}) \,||_{\emptyset}\, \underline{0}, \{a_{\swarrow}\}, \emptyset$$

Now a and b are selectable actions: a is resumable and b is discarded. If the selection were done according to priority levels and weights of a and b, we would have a new probabilistic selection and b could be selected causing the interruption of a. This would happen because of an extraneous event: the termination of c. ■

To solve the problem above, *the priority level of discarded actions is considered as being reduced by 0.5*. This because every discarded action is in conflict with at least one resumable action due to the maximality of the set of selected actions in the previous selection phase. The reduction of the priority level of discarded actions guarantees that selections already made are re-made with the same outcome when the priority level of involved actions is the same, and that correct selections are made even when actions at different priority levels are involved (because the amount of the reduction is less than 1).

Definition of transition relations

The two types of transitions are represented as follows:

- Termination transitions are labeled with the set $TerA$ of terminating action identifiers. They are represented by relation $\longrightarrow \subseteq LabS \times L_T \times LabS$ where $L_T = \mathcal{P}(AId)$ is the set of termination transition labels ranged over by $TerA$.

- Choice transitions are labeled with the priority level l and the weight w associated with the selection of the action. [7] They are represented by relation $\longmapsto \subseteq LabS \times L_C \times LabS$ where $L_C = (\mathbb{R}_+ \cup \{\infty\}) \times \mathbb{R}_+$ is the set of choice transition labels ranged over by (l, w).

A system state including both selectable actions and advancing actions has both outgoing choice transitions and outgoing termination transitions. Because of the priority of preselection policy over race policy, the system will leave the state only through transitions of the former type (we say that *choice transitions have priority over termination transitions*). System states are then divided into:

- *Timed states*: states with no leaving choice transitions (the system sojourn in this states until the termination of an action).
- *Choice states*: states with at least one leaving choice transition (sojourn time in these states is zero).

Moreover the system may leave choice states only through choice transitions with the highest priority level.

$\longrightarrow$ is defined as the least subset of $LabS \times L_T \times LabS$ that satisfies the inference rule reported in the first part of Table 1. [8] $\longrightarrow$ is defined through the family of auxiliary transition relations $\rightarrowtail_{TALoc}$ where $TALoc \in \mathcal{P}(Loc)$ represents the locations of terminating actions. Each of them is defined as the least subset of $\mathcal{G}_{dec} \times \mathcal{G}_{dec}$ such that they satisfy the inference rules reported in the second part of Table 1. Through these relations it is established the term relative to the new state the system reaches in case of termination of a *fixed* set of actions. The inference rules must be read beginning from what has to be deduced and establishing the only way to deduce it.

$\longmapsto$ is defined as the least subset of $LabS \times L_C \times LabS$ that satisfies the inference rule reported in the first part of Table 2. The three conditions presented therein refer to the different kinds of selectable actions: resumable, discarded or new. $\longmapsto$ is defined through the auxiliary transition relation $\rightarrowtail\!\!\!\!\!\mid\;$ defined as the least subset of $\mathcal{G}_{dec} \times L_{AC} \times \mathcal{G}_{dec}$ (where $L_{AC} = DAct$ is the set of its labels ranged over by $<\hat{a}^{x\,m}_{loc}, l, w>$ which represents the selected action) that satisfies the inference rules reported in the second part of Table 2. This relation is employed to establish the attributes of the selected action and the term relative to the new state the system reaches.

In Table 3 we define the auxiliary functions employed in Tables 2 and 1. Function $mode : AMode \times AMode \longrightarrow AMode$ evaluates the modality of an action obtained through cooperation. Function $hide : AMode \longrightarrow AMode$ changes the modality of an action that becomes invisibile. Function $free : \mathcal{G}_{dec} \longrightarrow Bool$ establishes if a term includes local actions decorated with "$\hat{\ }$" and is used to detect conflicts among actions. Operator $* : Dec \times Dec \longrightarrow Dec$ establishes the decoration of an action obtained through cooperation. Functions $froz : MIds \longrightarrow \mathcal{P}(AId)$, $adv : MIds \longrightarrow \mathcal{P}(AId)$ and $cont : MIds \longrightarrow \mathcal{P}(AId)$ extract from a set of identifiers with modality those identifiers which refer to frozen, advancing and continuable (with modality c) activities, respectively. Functions $\pi_l : \mathcal{P}(Loc) \longrightarrow \mathcal{P}(Loc)$ and $\pi_r : \mathcal{P}(Loc) \longrightarrow \mathcal{P}(Loc)$ evaluate the left (right) projection of a set of locations, i.e. they extract the locations related to terms to the left (to the right) of a parallel composition operator.

3.1.3 Definition of the LTS

The LTS underlying a given term $E \in \mathcal{G}$ is then

$$(S_E, \longrightarrow_E, \longmapsto_E, r_E; \mathcal{G}_{dec}, L_S, L_C, L_T)$$

where S_E is the set of states reachable from the initial state $r_E = (E, \emptyset, \emptyset)$ via transitions in $\longrightarrow_E$, which is the restriction of $\longrightarrow$ to $S_E \times L_C \times S_E$, and in $\longmapsto_E$, which is the restrction of $\longmapsto$ to $S_E \times L_T \times S_E$.

3.2 Durations of system activities

The set of activities that may be executed during the life of a system specified by term $E \in \mathcal{G}$ is given by

[7] According to what explained in Sect. 3.1.2.

[8] We use the following notation. Operator $\rightarrow$ builds a function beginning from a set X and an element y in this way: $X \rightarrow y = \{(x, y) \mid x \in X\}$. We denote by $f|_X$ the function resulting from the restriction of the domain of f into X. We define $Bool = \{True, False\}$.

$$\frac{F \rightarrowtail_{\{loc \mid a_{loc} \in TerA\}} F'}{F, IntA, Exec \xrightarrow{TerA} F', IntA', Exec'} \qquad TerA \subseteq adv(Exec) \wedge TerA \neq \emptyset$$

$$\begin{aligned} \text{where :} \quad & IntA' = adv(Exec) - TerA \;\wedge \\ & Exec' = (froz(Exec) \cup cont(Exec \mid_{IntA'})) \rightarrow f \end{aligned}$$

$$\underline{0} \rightarrowtail_{\emptyset} \underline{0}$$

$$<\hat{a}_i^m, l, w>.E \rightarrowtail_{\{i\}} E$$

$$<\hat{a}_i^m, l, w>.E \rightarrowtail_{\emptyset} <\bar{a}_i^m, l, w>.E$$

$$<\overset{x}{a}{}_i^m, l, w>.E \rightarrowtail_{\emptyset} <\tilde{a}_i^m, l, w>.E \qquad x \in \{\bar{\ }, \tilde{\ }, \_\}$$

$$\frac{F \rightarrowtail_{TALoc} F'}{F/L \rightarrowtail_{TALoc} F'/L}$$

$$\frac{F \rightarrowtail_{TALoc} F'}{F[\varphi] \rightarrowtail_{TALoc} F'[\varphi]}$$

$$\frac{F_1 \rightarrowtail_{TALoc} F_1'}{F_1 + F_2 \rightarrowtail_{TALoc} F_1'} \quad TALoc \neq \emptyset \qquad \frac{F_2 \rightarrowtail_{TALoc} F_2'}{F_1 + F_2 \rightarrowtail_{TALoc} F_2'} \quad TALoc \neq \emptyset$$

$$\frac{F_1 \rightarrowtail_{\emptyset} F_1' \qquad F_2 \rightarrowtail_{\emptyset} F_2'}{F_1 + F_2 \rightarrowtail_{\emptyset} F_1' + F_2'}$$

$$\frac{F_1 \rightarrowtail_{\pi_l(\,TALoc)} F_1' \qquad F_2 \rightarrowtail_{\pi_r(\,TALoc)} F_2'}{F_1 \parallel_S F_2 \rightarrowtail_{TALoc} F_1' \parallel_S F_2'}$$

$$\frac{E \rightarrowtail_{\emptyset} F}{A \rightarrowtail_{\emptyset} F} \qquad A \triangleq E$$

Table 1: Rules for termination transitions

$$\frac{F \stackrel{<\overset{x}{a}^{m}_{loc},l,w>}{\rightarrowtail} F'}{F, IntA, Exec \stackrel{l',w}{\longmapsto} F', IntA', Exec|_{dom(Exec)-\{a_{loc}\}} \cup \{(a_{loc}, m)\}}$$

$$\begin{array}{llll}
\text{where} & l' = \left\{ \begin{array}{lll} l & \text{if} & m \neq n \\ \infty & \text{if} & m = n \end{array} \right. & \wedge\ IntA' = IntA - \{a_{loc}\} & \text{if} \;\; x = \text{“}{-}\text{”} \wedge a_{loc} \in IntA \\
 & l' = l - 0.5 & \wedge\ IntA' = IntA & \text{if} \;\; (x = \text{“}{-}\text{”} \wedge a_{loc} \notin IntA) \vee x = \text{“}\tilde{\ }\text{”} \\
 & l' = l & \wedge\ IntA' = IntA & \text{if} \;\; x = \text{“}_\text{”}
\end{array}$$

$$<a^{m}_{i}, l, w>.E \stackrel{<a^{m}_{i},l,w>}{\rightarrowtail} <\hat{a}^{m}_{i}, l, w>.E$$

$$<\tilde{a}^{m}_{i}, l, w>.E \stackrel{<\tilde{a}^{m}_{i},l,w>}{\rightarrowtail} <\hat{a}^{m}_{i}, l, w>.E$$

$$<\bar{a}^{m}_{i}, l, w>.E \stackrel{<\bar{a}^{m}_{i},l,w>}{\rightarrowtail} <\hat{a}^{m}_{i}, l, w>.E$$

$$\frac{F \stackrel{<\overset{x}{a}^{m}_{loc},l,w>}{\rightarrowtail} F'}{F/L \stackrel{<\overset{x}{a}^{hide(m)}_{loc},l,w>}{\rightarrowtail} F'/L} \quad a \in L \qquad\qquad \frac{F \stackrel{<\overset{x}{a}^{m}_{loc},l,w>}{\rightarrowtail} F'}{F/L \stackrel{<\overset{x}{a}^{m}_{loc},l,w>}{\rightarrowtail} F'/L} \quad a \notin L$$

$$\frac{F \stackrel{<\overset{x}{a}^{m}_{loc},l,w>}{\rightarrowtail} F'}{F[\varphi] \stackrel{<\overset{x}{\varphi(a)}^{m}_{loc},l,w>}{\rightarrowtail} F'[\varphi]}$$

$$\frac{F_1 \stackrel{<\overset{x}{a}^{m}_{loc},l,w>}{\rightarrowtail} F_1'}{F_1 + F_2 \stackrel{<\overset{x}{a}^{m}_{loc},l,w>}{\rightarrowtail} F_1' + F_2} \quad free(F_2) \qquad\qquad \frac{F_2 \stackrel{<\overset{x}{a}^{m}_{loc},l,w>}{\rightarrowtail} F_2'}{F_1 + F_2 \stackrel{<\overset{x}{a}^{m}_{loc},l,w>}{\rightarrowtail} F_1 + F_2'} \quad free(F_1)$$

$$\frac{F_1 \stackrel{<\overset{x}{a}^{m}_{loc},l,w>}{\rightarrowtail} F_1'}{F_1 \parallel_S F_2 \stackrel{<\overset{x}{a}^{m}_{\swarrow loc},l,w>}{\rightarrowtail} F_1' \parallel_S F_2} \quad a \notin S \vee m = hide(m)$$

$$\frac{F_2 \stackrel{<\overset{x}{a}^{m}_{loc},l,w>}{\rightarrowtail} F_2'}{F_1 \parallel_S F_2 \stackrel{<\overset{x}{a}^{m}_{\searrow loc},l,w>}{\rightarrowtail} F_1 \parallel_S F_2'} \quad a \notin S \vee m = hide(m)$$

$$\frac{F_1 \stackrel{<\overset{x}{a}^{m_1}_{loc_1},l_1,w_1>}{\rightarrowtail} F_1' \qquad F_2 \stackrel{<\overset{y}{a}^{m_2}_{loc_2},l_2,w_2>}{\rightarrowtail} F_2'}{F_1 \parallel_S F_2 \stackrel{<\overset{x*y}{a}^{mode(m_1,m_2)}_{\langle loc_1 | loc_2 \rangle},l_1+l_2,w_1 \cdot w_2>}{\rightarrowtail} F_1' \parallel_S F_2'} \quad a \in S \wedge m_1 \neq hide(m_1) \wedge m_2 \neq hide(m_2)$$

$$\frac{E \stackrel{<\overset{x}{a}^{m}_{loc},l,w>}{\rightarrowtail} F}{A \stackrel{<\overset{x}{a}^{m}_{loc},l,w>}{\rightarrowtail} F} \quad A \triangleq E$$

Table 2: Rules for choice transitions

$$mode(n,n) = n \qquad mode(a,a) = a \qquad mode(c,c) = c$$
$$mode(a,c) = mode(c,a) = a$$
$$mode(n,c) = mode(c,n) = mode(n,a) = mode(a,n) = n$$

$$hide(hc) = hide(c) = hc \qquad hide(ha) = hide(a) = ha \qquad hide(hn) = hide(n) = hn$$

$$free(<\hat{a}_i^m, l, w>.E) = False$$
$$free(<\overset{x}{a}_i^m, l, w>.E) = True \qquad x \in \{^-, \tilde{}\,, \_\}$$
$$free(F/L) = free(F[\varphi]) = free(F)$$
$$free(F_1 + F_2) = free(F_1 \parallel_S F_2) = free(F_1) \wedge free(F_2)$$
$$free(A) = True$$

$$\text{``}\_\text{''} * \text{``}\_\text{''} = \text{``}\_\text{''} \qquad \text{``}\sim\text{''} * \text{``}\sim\text{''} = \text{``}\sim\text{''} \qquad \text{``}-\text{''} * \text{``}-\text{''} = \text{``}-\text{''}$$
$$\text{``}\sim\text{''} * \text{``}-\text{''} = \text{``}-\text{''} * \text{``}\sim\text{''} = \text{``}\sim\text{''}$$
$$\text{``}\_\text{''} * \text{``}-\text{''} = \text{``}-\text{''} * \text{``}\_\text{''} = \text{``}\_\text{''} * \text{``}\sim\text{''} = \text{``}\sim\text{''} * \text{``}\_\text{''} = \text{``}\_\text{''}$$

$$froz(mids) = mids^{-1}(\{f\}) = \{a_{loc} \mid (a_{loc}, f) \in mids\}$$
$$adv(mids) = mids^{-1}(\{a, c, n, ha, hc, hn\})$$
$$cont(mids) = mids^{-1}(\{c, hc\})$$

$$\pi_l(locs) = \{loc \mid \swarrow loc \in locs \vee \exists loc'.\, \langle loc | loc' \rangle \in locs\}$$
$$\pi_r(locs) = \{loc \mid \searrow loc \in locs \vee \exists loc'.\, \langle loc' | loc \rangle \in locs\}$$

Table 3: Auxiliary functions

$$Ids_E = \{a_{loc} \mid \exists (F, IntA, Exec) \in S_E .\, a_{loc} \in dom(Exec)\}$$

Function $PDF_E : Ids_E \longrightarrow PDF$ that assigns to each activity of Ids_E its distribution of duration is computed, beginning from the specification of durations, as follows

$$PDF_E(a_{loc}) = PDFCalc(a_{loc}, \bullet)$$

where function $PDFCalc$ deals with functions PDF_a and PDF_{pid} in the following way.

Definition 3.7 $PDFCalc : (AId \times PId) \longrightarrow PDF$ is defined by:

$$PDFCalc(a_{\langle loc_1 | loc_2 \rangle}, pid) = PDF_a(PDFCalc(a_{loc_1}, pid\swarrow), PDFCalc(a_{loc_2}, pid\searrow))$$
$$PDFCalc(a_{\swarrow loc}, pid) = PDFCalc(a_{loc}, pid\swarrow)$$
$$PDFCalc(a_{\searrow loc}, pid) = PDFCalc(a_{loc}, pid\searrow)$$
$$PDFCalc(a_i, pid) = PDF_{pid}(a_i)$$

■

3.3 Definition of the integrated operational semantics

Definition 3.8 The *integrated operational interleaving semantics* of $E \in \mathcal{G}$ is given by the following pair composed of a LTS and a set of duration distributions

$$\mathcal{I}[\![E]\!] = (S_E, \longrightarrow_E, \longmapsto_E, r_E; PDF_E)$$ ■

4 Functional and performance semantics

Due to lack of space, the functional and performance projections of the integrated semantics are only briefly sketched here. For their full definitions as well as the presentation of EGSMPs, the reader is referred to [4].

The functional model is obtained from the integrated model by removing the quantitative information relative to duration of activities and probability of choices. It is worth noting that the functional model describes the system behavior with the same precision as the integrated model since it retains the identification of actions.

The performance model is obtained from the integrated model by abstracting from functional information, i.e. from the representation of identifiers (and hence from the location of processes that execute activities) and from action types. An EGSMP is formally derived and many reduction methods for EGSMP are introduced such as vanishing states elimination and lumping procedure. Moreover EGSMPs turn out to be an appropriate purely stochastic framework for the application of the method of extended insensitivity presented in [10]. Such method is translated into the framework of EGSMPs.

5 Conclusion

In this paper we have presented a restricted version of the stochastically timed process algebra GSMPA, developed in order to extend EMPA expressiveness with the direct representation of arbitrarily distributed durations. GSMPA has been developed in such a way as to give its terms a semantic model close to the representation of a GSMP. Since in GSMPs, thanks to activity identification, the system behavior is represented in an interleaving form (i.e. considering all possible orderings for action terminations), keeping the EMPA interleaving approach for the definition of the semantics turned out to be the right choice.

In GSMPA duration distributions are assigned to activities separately from the description of system behavior through identifiers. Therefore, for this purpose, there is no need to resort to different approaches such as, e.g., the truly concurrent one of [5]. Note that the truly concurrent models presented in [5] must be translated into an interleaving form before performance can be evaluated. Furthermore, with respect to discrete event simulation approaches, e.g. [8], GSMPA gives the possibility of deriving *finite* performance models belonging to a well-known and studied stochastic process class, thus allowing mathematical analysis through established theoretical results such as, e.g., solution by insensitivity. Should these techniques not be applicable, a simulative analysis can always be performed, since the semantics for GSMPA is defined operationally.

The development of GSMPA has been influenced by the language ET-LOTOS [1]. The main difference is that in ET-LOTOS the preselection policy is not used, and choices are resolved with the race policy. Performance models are derived from ET-LOTOS terms by applying to them a preliminary procedure that identifies each action differently. As a consequence an activity started by an action can continue only with the same action by the recursion operator. There is, therefore, no way to express different interruption modalities for the same activity (when interrupted it is either always aborted or always frozen). In GSMPA, instead, thanks to the preselection policy which allows us to equally identify alternative actions, we can employ the suitable identification mechanism based on locations that allows different actions to have the same identifier. Besides, the preselection policy: allows an implicit treatment of contemporaneous terminations which are instead managed with a selection mechanism in ET-LOTOS; makes GSMPA a resource aware language; and enables the direct representation of selections by separating them from durations, without the use of artifices such as immediate actions of ET-LOTOS. Moreover in ET-LOTOS itself the usefulness of the preselection approach is recognized by defining, through a macro, exactly a GSMPA action as the sequential composition of a (prioritized weighted) immediate action and a timed action. Another difference from ET-LOTOS is the capability of GSMPA to compositionally specify activity durations. Other algebras which consider general distributions are presented in [7, 17]. They follow the interleaving approach, but use techniques different from identification to deal with general distributions. In [7] the technique of "start reference" is employed in order to have a pointer to the system state where an action begins its execution. In [17], instead, information about causality relations among actions are exploited in order to establish the starting point of actions. The absence of identification causes two main drawbacks: the interruption of an action with following continuation cannot be expressed, and semantic models cannot be directly transformed into performance models belonging to a well-known class of stochastic processes. Moreover in [7] and [17] only continuous general distributions are considered, and therefore deterministic activities cannot be expressed.

In [4] the expressiveness of GSMPA has been studied through the specification of the alternating bit protocol. This is an interesting example of a simple system where deterministic durations (e.g. timeout period) and probabilistic durations (e.g. message transmission time) coexist. Moreover, the specification of the timer of the sender requires, in case of interruption due to the reception of an acknowledgement, that it continues if a wrong acknowledgement arrives, whilst it is aborted (for further utilization) if the right one arrives. Therefore GSMPA capability to express different interruption modalities for the same activity is needed.

Finally, the future developments of GSMPA concern, first of all, the study of the properties of its integrated equivalence presented in [4], especially the congruence property. Furthermore, we are interested in: devising other analysis techniques for EGSMPs (beside those in [4]); studying conditions under which it is possible to formally obtain an EMPA specification from a GSMPA specification; and defining a net semantics in order to implement the integrated approach of [2] in the case of arbitrary distributions.

Acknowledgements

This research has been partially funded by MURST, CNR and Esprit Project n. 8130 LOMAPS.

References

[1] M. Ajmone Marsan, A. Bianco, L. Ciminiera, R. Sisto, A. Valenzano, *"A LOTOS Extension for the Performance Analysis of Distributed Systems"*, in IEEE/ACM Trans. on Networking 2:151-164, 1994

[2] M. Bernardo, L. Donatiello, R. Gorrieri, *"Integrating Performance and Functional Analysis of Concurrent Systems with EMPA"*, Technical Report UBLCS-95-14, University of Bologna (Italy), 1996

[3] M. Bernardo, R. Gorrieri, *"A Tutorial on EMPA: A Theory of Concurrent Processes with Nondeterminism, Priorities, Probabilities and Time"*, to appear in Theoretical Computer Science, 1997

[4] M. Bravetti, M. Bernardo, R. Gorrieri, *"Generalized Semi-Markovian Process Algebra"*, Technical Report UBLCS-97-9, University of Bologna (Italy), 1997

[5] E. Brinksma, J.-P. Katoen, R. Langerak, D. Latella, *"A Stochastic Causality-Based Process Algebra"*, in Computer Journal 38:552-565, 1995

[6] D. Ferrari, *"Considerations on the Insularity of Performance Evaluation"*, in IEEE Trans. on Software Engineering 12:678-683, 1986

[7] N. Götz, U. Herzog, M. Rettelbach, *"TIPP - A Stochastic Process Algebra"*, in Proc. of PAPM '93, pp. 31-36, Edinburgh (UK), 1993

[8] P.G. Harrison, B. Strulo, *"Stochastic Process Algebra for Discrete Event Simulation"*, in *Quantitative Methods in Parallel Systems*, Springer, pp. 18-37, 1995

[9] W. Henderson, *"Insensitivity and Reversed Markov Processes"*, in Advanced Applied Probab. 15:752-768, 1983

[10] W. Henderson, D. Lucic, *"Aggregation and Disaggregation through Insensitivity in Stochastic Petri Nets"*, in Performance Evaluation 17:91-114, 1993

[11] W. Henderson, P. Taylor, *"Insensitivity of Processes with Interruptions"*, in Journal of Applied Probability 26:242-258, 1989

[12] J. Hillston, *"The Nature of Synchronisation"*, in Proc. of PAPM '94, pp. 51-70, Erlangen (Germany), 1994

[13] C.A.R. Hoare, *"Communicating Sequential Processes"*, Prentice Hall, 1985

[14] K. Matthes, *"Zur Theorie der Bedienungsprozesse"*, in Trans. of the *3rd Prague Conf. on Information Theory, Stat. Dec. Fns. and Random Processes*, pp. 513-528, 1962

[15] M.F. Neuts, *"Matrix-Geometric Solutions in Stochastic Models - An Algorithmic Approach"*, John Hopkins University Press, 1981

[16] N. Nounou, Y. Yemini, *"Algebraic Specification-Based Performance Analysis of Communication Protocols"*, in Proc. of PSTV V, North Holland, pp. 541-560, 1985

[17] C. Priami, *"Stochastic π-Calculus with General Distributions"*, in Proc. of PAPM '96, CLUT, pp. 41-57, Torino (Italy), 1996