Component-Oriented Specification of Performance Measures

Alessandro Aldini [1] Marco Bernardo [2]

Università di Urbino "Carlo Bo", Istituto STI, Italy

Abstract

Formal notations for system performance modeling need to be equipped with suitable notations for specifying performance measures. These companion notations have been traditionally based on reward structures and, more recently, on temporal logic. In this paper we propose a mixed approach, which aims at facilitating the specification of performance measures by allowing the designer to express them in a component-oriented way. The resulting Measure Specification Language MSL, which is being integrated in Æmilia/TwoTowers, is interpreted both on action-labeled continuous-time Markov chains and on stochastic process algebras. The latter interpretation provides a compositional framework for performance-sensitive model manipulations and emphasizes the increased expressiveness with respect to traditional reward structures for modeling notations in which the concept of state is implicit.

1 Introduction

The need for assessing the quantitative characteristics of a system during the early stages of its design has fostered within the academic community the development of formal methods integrating the traditionally addressed functional aspects with the performance aspects. This has resulted in different system modeling notations, with complementary strengths and weaknesses, among which we mention stochastic process algebras (SPA: see, e.g., [14,13,7] and the references therein) and stochastic Petri nets (SPN: see, e.g., [2] and the references therein). Both SPAs and SPNs are equipped with precisely defined semantics as well as analysis techniques, which – in the performance evaluation case – require the solution of the underlying stochastic process in the form of a continuous-time Markov chain (CTMC [19]).

[1] Email: aldini@sti.uniurb.it
[2] Email: bernardo@sti.uniurb.it

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

From the usability viewpoint, the modeling notations above force the system designer to be familiar with their technicalities, some of which are not so easy to learn. Moreover, such notations do not support a fully elucidated component-oriented way of modeling systems, which is especially desirable when dealing with complex systems made out of numerous interacting parts.

This usability issue has been recently tackled with the development of Æmilia [8,5], an architectural description language based on EMPA_{gr} [7] for the textual and graphical representation of system families. Æmilia clearly separates the specification of the system behavior from the specification of the system topology, thus hiding many of the technicalities of the static operators of SPA. This is achieved by dividing an Æmilia specification into two sections. In the first section, the designer defines – through SPA equations in which only the easier dynamic operators can occur – the behavior of the types of components that form the system, together with their interactions with the rest of the system. In the second section, the designer declares the instances of the previously defined types of components that are present in the system, as well as the way in which their interactions are attached to each other in order to make the components communicate.

For performance evaluation purposes, the modeling notations mentioned before have been endowed with companion notations for the specification of the performance measures of interest. According to the classifications proposed in [18,12], we have instant-of-time measures, expressing the gain/loss received at a particular time instant, and interval-of-time (or cumulative) measures, expressing the overall gain/loss received over some time interval. Both kinds of measures can refer to stationary or transient state. Most of the approaches that have appeared in the literature for expressing various kinds of performance measures are based on the definition of reward structures [15] for the CTMCs underlying the system models.

In the framework of modeling notations like SPA and SPN, the idea is that the reward structures should not be defined at the level of the CTMC states and transitions, but at the level of the system models and then automatically inherited by their underlying CTMCs. In the SPN case, the rewards can naturally be associated with the net markings and the net transitions/activities [9,17]. In the SPA case, the reward association is harder because the modeling notation is action-based, hence the concept of state is implicit. In [10,11] the CTMC states to which certain rewards have to be attached are singled out by means of suitable modal logic formulas, whereas in [7,6] the rewards are directly specified within the actions occurring in the system specifications and are then transferred to the CTMC states and transitions during the CTMC construction. In [20], instead, temporal reward formulas have been introduced, which are able to express accumulated atomic rewards over sequences of CTMC states and allow performance measures to be evaluated through techniques for computing long-run averages. Finally, a different, non-reward-based approach relies on the branching-time temporal

logic CSL [3], which is used to directly specify performance measures and to reduce performance evaluation to model checking. Based on the observation that the progress of time can be regarded as the earning of reward, a variant of CSL called CRL has been subsequently proposed in [4], where rewards are assumed to be already attached to the CTMC states.

The usability issue for the performance modeling notations obviously extends to the companion notations for expressing performance measures. In particular, we observe that none of the proposals surveyed above allows the designer to specify the performance measures in a component-oriented way, which once again would be highly desirable.

From the designer viewpoint, even the use of a component-oriented modeling notation like Æmilia may be insufficient if accompanied by an auxiliary notation in which the specification of performance measures is not easy. This was the outcome of a usability-related experiment conducted with some graduate and undergraduate students at the University of L'Aquila. Such students, who are familiar with software engineering concepts and methodologies, but not with formal methods like SPA, were previously exposed to SPA together with the reward-based companion notation proposed in [6], then they were exposed to Æmilia together with the same companion notation. At the end of this process, on the modeling side the students felt more confident about the correctness of the communications they wanted to establish – thanks to the separation of concerns between behavior specification and topology specification – and found very beneficial the higher degree of parametricity (hence the increased potential for specification reuse). On the other hand, they still complained about the difficulties with a notation to specify performance measures that forced them to reason in terms of states and transitions rather than components. Most importantly, they perceived the definition of the measures as a task for performance experts, because for them it was not trivial at all to decide which kinds and values of rewards to use in order to derive even simple indicators like system throughput or resource utilization.

Although the difficulty with choosing adequate values for the rewards is an intrinsic limitation of the reward-based approach to the specification of performance measures, in this paper we claim that a remarkable improvement of the usability of such an approach can be obtained by combining ideas from action-based methods and logic-based methods in a component-oriented flavor. More specifically, we shall propose a Measure Specification Language (MSL) that builds on a simple first-order logic by means of which the rewards are attached to the states and the transitions of the CTMCs underlying component-oriented system models, like e.g. Æmilia specifications. On the one hand, such an integrated approach relying on both rewards and logical constructs turns out to be more expressive than classical reward-based methods when using modeling notations like SPA in which the concept of state is implicit. On the other hand, component-orientation is achieved within MSL by means of a mechanism to define measures that are parameterized with re-

spect to the activities that individual components or specific parts of their behavior can carry out. Another contribution of this paper is to provide an interpretation for the core logic of MSL based on SPA, which allows for performance-sensitive compositional reasoning.

The rest of the paper is organized as follows. In Sect. 2 we recall some background about component-oriented system modeling, action-labeled CTMCs, and reward structures. In Sect. 3 we present MSL by defining its core logic together with its CTMC interpretation. In Sect. 4 we present the measure definition mechanism associated with MSL. In Sect. 5 we provide the SPA-based interpretation for the core logic of MSL. Finally, in Sect. 6 we conclude by reporting some perspectives on future work.

2 Setting the Context

The formal approach to the specification of performance measures we present in this paper is conceived for component-oriented system models whose underlying stochastic processes are action-labeled CTMCs.

2.1 Component-Oriented System Models

Following the guidelines proposed in [1], the formal model of a component-oriented system should comprise at least two parts: the description of the individual system component types and the description of the overall system topology.

The description of a system component type should be provided by specifying at least its name, its (data-related and performance-related) parameters, its behavior, and its interactions. The behavior should express all the alternative sequences of activities that the component type can carry out [3], while the interactions are those activities occurring in the behavior that are used by the component type to communicate with the rest of the system. The interactions can be annotated with qualifiers expressing their attributes concerning e.g. the direction (input vs. output) or the form (point-to-point, broadcast, client-server, etc.) of the communication they can be involved in.

The description of the system topology should be provided by declaring the instances of component types that form the system, together with the specification of the way in which their interactions should be attached to each other in order to make the components communicate. If the interactions are annotated with qualifiers, the attachments should be consistent with them. The description of the topology should then be completed by the possible indication of component interactions that act as interfaces for the overall system, which is useful to support hierarchical modeling.

[3] This general framework allows for both branching-time and linear-time models and includes different formalisms like process algebras and Petri nets.

In the following we consider as an illustrative example a queueing system $M/M/2$ with arrival rate $\lambda \in \mathbb{R}_{>0}$, no buffer, and service rates $\mu_1, \mu_2 \in \mathbb{R}_{>0}$ [16]. This system represents a service center equipped with two servers processing requests at rate μ_1 and μ_2, respectively. Service is provided to an unbounded population of customers, which arrive at the service center according to a Poisson process of rate λ. Whenever both servers are idle, an incoming customer has the same probability to be served by the two servers.

The overall system thus comprises two component types: the arrival process and the server. In the framework of the architectural description language Æmilia [8] such component types would be modeled as follows:

```
ARCHI_TYPE QS_M_M_2(rate lambda, rate mu1, rate mu2)
  ARCHI_ELEM_TYPES
    ELEM_TYPE Arrivals_Type(rate arrival_rate)
      BEHAVIOR
        Arrivals(void) =
          <arrive, arrival_rate> . Arrivals()
      INPUT  INTERACTIONS void
      OUTPUT INTERACTIONS OR arrive
    ELEM_TYPE Server_Type(rate service_rate)
      BEHAVIOR
        Server_Idle(void) =
          <arrive, _> . Server_Busy();
        Server_Busy(void) =
          <serve, service_rate> . Server_Idle()
      INPUT  INTERACTIONS UNI arrive
      OUTPUT INTERACTIONS void
```

The system topology comprises one instance of `Arrivals_Type` and two instances of `Server_Type`, suitably connected to each other as modeled below in Æmilia:

```
  ARCHI_TOPOLOGY
    ARCHI_ELEM_INSTANCES
      Arr : Arrivals_Type(lambda);
      S1  : Server_Type(mu1);
      S2  : Server_Type(mu2);
    ARCHI_INTERACTIONS
      void
    ARCHI_ATTACHMENTS
      FROM Arr.arrive TO S1.arrive;
      FROM Arr.arrive TO S2.arrive
END
```

2.2 ACTMCs and Reward Structures

For performance evaluation purposes, we assume that from the considered component-oriented system models it is possible to extract finite-state, finitely-branching, action-labeled CTMCs.

Definition 2.1 A finite action-labeled CTMC (ACTMC) is a quadruple

$$\mathcal{M} = (S, Act, \longrightarrow_{\mathcal{M}}, s_0)$$

where S is a finite set of states, $s_0 \in S$ is the initial state, Act is a non-empty set of activities, and $\longrightarrow_{\mathcal{M}} \subseteq S \times (Act \times \mathbb{R}_{>0}) \times S$ is a finite transition relation. ■

Each state of an ACTMC obtained from a component-oriented system model is actually a global state representing a system configuration that can be viewed as a vector of local states, which are the current behaviors of the individual components. Each transition corresponds instead to either an activity performed by a single component in isolation, or a set of attached interactions executed simultaneously by several communicating components. As an example, Fig. 1 shows the ACTMC underlying the queueing system modeled with Æmilia in Sect. 2.1.

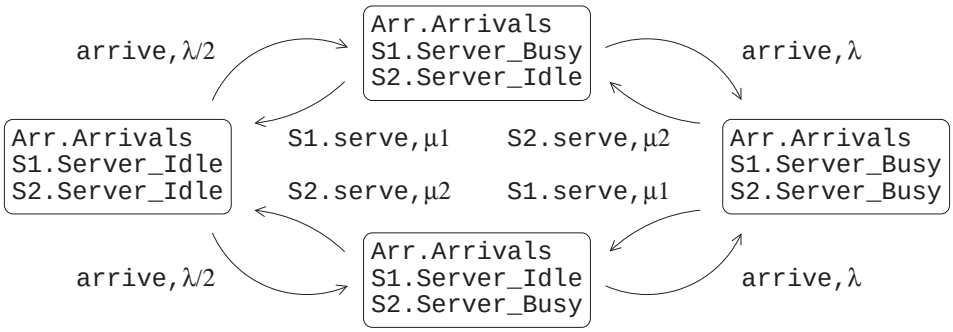

Fig. 1. ACTMC model of the queueing system example

As far as the analysis of ACTMC-based, component-oriented models is concerned, the typical approach to performance measure specification relies on reward structures [15]. This requires associating real numbers with system behaviors and activities, which are then transferred to the proper states (rate rewards) and transitions (instantaneous rewards) of the ACTMC, respectively.

A rate reward expresses the rate at which a gain (or a loss, if the number is negative) is accumulated while sojourning in the related state. By contrast, an instantaneous reward specifies the instantaneous gain (or loss) implied by the execution of the related transition.

The instant-of-time value of a performance measure specified through a reward structure is computed for an ACTMC $\mathcal{M} = (S, Act, \longrightarrow_{\mathcal{M}}, s_0)$ through the following equation:

$$(1) \qquad \sum_{s \in S} R_{\mathrm{r}}(s) \cdot \pi(s) + \sum_{(s,a,\lambda,s') \in \longrightarrow_{\mathcal{M}}} R_{\mathrm{i}}(s,a,\lambda,s') \cdot \phi(s,a,\lambda,s')$$

where:

- $R_{\mathrm{r}}(s)$ is the rate reward associated with s.
- $\pi(s)$ is the probability of being in s at the considered instant of time.
- $R_{\mathrm{i}}(s, a, \lambda, s')$ is the instantaneous reward associated with the transition (s, a, λ, s').
- $\phi(s, a, \lambda, s')$ is the frequency of the transition (s, a, λ, s') at the considered instant of time, which is given by $\pi(s) \cdot \lambda$.

Suppose e.g. that, in the queueing system example, we are interested in computing throughput and utilization. The system throughput is defined as the mean number of customers that are served per time unit. In order to compute it, we should set:

$$R_{\mathrm{r}}(s) = \begin{cases} \mu_1 & \text{if } s = (\texttt{Arr.Arrivals}, \texttt{S1.Server\_Busy}, \texttt{S2.Server\_Idle}) \\ \mu_2 & \text{if } s = (\texttt{Arr.Arrivals}, \texttt{S1.Server\_Idle}, \texttt{S2.Server\_Busy}) \\ \mu_1 + \mu_2 & \text{if } s = (\texttt{Arr.Arrivals}, \texttt{S1.Server\_Busy}, \texttt{S2.Server\_Busy}) \\ 0 & \text{if } s = (\texttt{Arr.Arrivals}, \texttt{S1.Server\_Idle}, \texttt{S2.Server\_Idle}) \end{cases}$$

Equivalently, we may set $R_{\mathrm{i}}(\_, a, \_, \_) = 1$ for $a \in \{\texttt{S1.serve}, \texttt{S2.serve}\}$ and $R_{\mathrm{i}}(\_, a, \_, \_) = 0$ for $a \in \{\texttt{arrive}\}$.

The system utilization, instead, is defined as the percentage of time during which at least one server is busy. In order to compute it, we should set $R_{\mathrm{r}}(s) = 1$ if s contains as local state at least one between `S1.Server_Busy` and `S2.Server_Busy`, $R_{\mathrm{r}}(s) = 0$ otherwise.

3 MSL: Core Logic and ACTMC Interpretation

MSL is based on a core logic for associating rewards with the ACTMCs underlying component-oriented system models. The core logic is in turn based on a set of first-order predicates, which we shall interpret on an ACTMC $\mathcal{M} = (S, Act, \longrightarrow_{\mathcal{M}}, s_0)$, that can be in the scope of quantifications with respect to an activity set $A \subseteq Act$.

In order to achieve a satisfactory degree of expressiveness, at least four formula schemas are needed in the core logic. On the one hand, the designer has to be allowed to decide whether state rewards or transition rewards are needed to define a certain performance measure. On the other hand, in an action-based setting the designer has to be allowed to decide whether all the activities in a given set contribute to the value of a certain performance measure, or only one of them does. The combination of the two sets of two options results in four possibilities that are made available to the designer.

Definition 3.1 The core logic of MSL is a first-order logic composed of the following four formula schemas:

(i) $\forall a \in A(is\_trans(s, a, \lambda, s') \Rightarrow eq(partial\_contrib(s, a, \lambda, s'), rew(a, \lambda))) \Rightarrow$
$eq(state\_rew(s), sum\_partial\_contrib(s, A))$

(ii) $\forall a \in A(is\_trans(s, a, \lambda, s') \Rightarrow eq(trans\_rew(s, a, \lambda, s'), rew(a, \lambda)))$

(iii) $\exists a \in A(is\_trans(s, a, \lambda, s')) \Rightarrow$
$eq(state\_rew(s), choose\_state\_rew(s, A, cf))$

(iv) $\exists a \in A(is\_trans(s, a, \lambda, s')) \Rightarrow$
$eq(trans\_rew(choose\_trans(s, A, cf)), choose\_trans\_rew(s, A, cf))$ ■

Because of their initial quantification, we call universal the first two formula schemas and existential the last two formula schemas. Intuitively, the first universal formula schema establishes that all the transitions labeled with an activity $a \in A$ that depart from the current state of $\mathcal{M}$ provide a contribution of value $rew(a, \lambda)$ to the rate at which the reward is gained while staying in that state. Since several contributing transitions may depart from the current state, we assume that all their partial contributions have to be summed up (partial contribution additivity assumption). The second universal formula schema establishes that all the transitions labeled with an activity $a \in A$ gain an instantaneous reward of value $rew(a, \lambda)$ whenever they are executed.

In the queueing system example, the system throughput can be specified through a formula of type (i) where:

$A = \{\texttt{S1.serve}, \texttt{S2.serve}\}, rew(\texttt{S1.serve}, \_) = \mu_1, rew(\texttt{S2.serve}, \_) = \mu_2$

The throughput of `S1` alone can still be specified through a formula of type (i) where:

$$A = \{\texttt{S1.serve}\}, rew(\texttt{S1.serve}, \_) = \mu_1$$

Finally, the system throughput can alternatively be specified through a formula of type (ii) where:

$$A = \{\texttt{S1.serve}, \texttt{S2.serve}\}, rew(\texttt{S1.serve}, \_) = rew(\texttt{S2.serve}, \_) = 1$$

The first existential formula schema establishes that the current state of $\mathcal{M}$ gains a contribution to the rate at which the reward is accumulated while staying there if it can execute at least one transition labeled with an activity $a \in A$. The value of such a contribution will have to be selected by applying a choice function cf to the rewards $rew(a, \lambda)$ associated with all the transitions labeled with an activity $a \in A$ that depart from the current state. By choice function we mean a function that simply returns one of its arguments, like e.g. max and min. The second existential formula schema establishes that only one of the transitions labeled with an activity $a \in A$ that depart from the current state of $\mathcal{M}$ gains an instantaneous reward upon execution. Such a transition is selected by means of a choice function cf, which takes into account the rewards $rew(a, \lambda)$ of the activities $a \in A$ labeling the transitions that depart from the current state multiplied by the frequencies of the transitions themselves.

In the queueing system example, the system utilization can be specified through a formula of type (iii) where:

$A = \{\texttt{S1.serve}, \texttt{S2.serve}\}, rew(\texttt{S1.serve}, \_) = rew(\texttt{S2.serve}, \_) = 1, cf = \min$

The utilization of S1 alone can still be specified through a formula of type (iii) where:

$$A = \{\texttt{S1.serve}\}, rew(\texttt{S1.serve}, \_) = 1, cf = \min$$

Finally, the actual arrival rate can be specified through a formula of type (iv) where:

$$A = \{\texttt{arrive}\}, rew(\texttt{arrive}, \_) = 1, cf = \min$$

In order to formalize the semantics of the core logic of MSL, we now provide the following ACTMC-based interpretation of the syntactical predicates and functions occurring in Def. 3.1:

- $is\_trans \subseteq S \times Act \times \mathbb{R}_{>0} \times S$ such that:

$$is\_trans(s, a, \lambda, s') = \begin{cases} 1 & \text{if } (s, a, \lambda, s') \in \longrightarrow_{\mathcal{M}} \\ 0 & \text{otherwise} \end{cases}$$

- $eq \subseteq \mathbb{R} \times \mathbb{R}$ such that:

$$eq(x, y) = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{otherwise} \end{cases}$$

- $rew : A \times \mathbb{R}_{>0} \to \mathbb{R}$ such that $rew(a, \lambda)$ is the reward contribution given by activity $a \in A$ when labeling a transition with rate $\lambda \in \mathbb{R}_{>0}$.
- $state\_rew : S \to \mathbb{R}$ such that $state\_rew(s)$ is the rate at which the reward is gained while staying in state s.
- $trans\_rew : \longrightarrow_{\mathcal{M}} \to \mathbb{R}$ such that $trans\_rew(s, a, \lambda, s')$ is the instantaneous reward that the transition (s, a, λ, s') gains whenever it is executed.
- $partial\_contrib : \longrightarrow_{\mathcal{M}} \to \mathbb{R}$ such that $partial\_contrib(s, a, \lambda, s')$ is the partial contribution given by the transition (s, a, λ, s') to the rate at which the state reward is gained at s.
- $sum\_partial\_contrib : S \times \{A\} \to \mathbb{R}$ such that:

$$sum\_partial\_contrib(s, A) = \sum_{a \in A} \sum_{(s,a,\lambda,s') \in \longrightarrow_{\mathcal{M}}} partial\_contrib(s, a, \lambda, s')$$

  where the sum is 0 whenever no transition labeled with $a \in A$ can be executed by s.
- $choose\_state\_rew : S \times \{A\} \times CF \to \mathbb{R}$ such that:

$$choose\_state\_rew(s, A, cf) = cf\{\!|\, rew(a, \lambda) \mid a \in A \wedge \exists s' \in S.\, is\_trans(s, a, \lambda, s') \,|\!\}$$

  where $CF = \{f : 2^{\mathbb{R}} \to \mathbb{R} \mid f(\emptyset) = 0 \wedge \forall n \in \mathbb{N}_{>0}.\, f(\{x_1, \ldots, x_n\}) \in \{x_1, \ldots, x_n\}\}$ is the set of the choice functions.
- $choose\_trans : S \times \{A\} \times CF \to \longrightarrow_{\mathcal{M}}$ such that:

$$choose\_trans(s, A, cf) = (s, a, \lambda, s')$$

iff:

$$rew(a,\lambda)\cdot\phi(s,a,\lambda,s') = cf(\{\!|\, rew(b,\mu)\cdot\phi(z,b,\mu,z') \mid b\in A \wedge is\_trans(z,b,\mu,z') \,|\!\})$$

- $choose\_trans\_rew : S \times \{A\} \times CF \not\to \mathbb{R}$ such that:

$$choose\_trans\_rew(s,A,cf) = rew(a,\lambda)$$

iff:

$$choose\_trans(s,A,cf) = (s,a,\lambda,s')$$

for some $s' \in S$.

In the light of the above ACTMC interpretation of the core logic of MSL, we observe that equation (1) is reformulated as follows with respect to an activity set A and a choice function cf:

$$\sum_{s\in S} UR_{\rm r}(s,A)\cdot\pi(s) + \sum_{a\in A}\sum_{(s,a,\lambda,s')\in\longrightarrow_{\mathcal{M}}} UR_{\rm i}(s,a,\lambda,s')\cdot\phi(s,a,\lambda,s') + \sum_{s\in S} ER_{\rm r}(s,A,cf)\cdot\pi(s) + \sum_{s\in S} ER_{\rm i}(s,A,cf)\cdot\phi(s,A,cf) \tag{2}$$

Each reward element of equation (2) maps to a corresponding MSL formula schema of Def. 3.1 as follows:

(i) $UR_{\rm r}(s,A)$ is the universal state reward with respect to A that is accumulated while staying in s, which is given by $sum\_partial\_contrib(s,A)$.

(ii) $UR_{\rm i}(s,a,\lambda,s')$ is the universal transition reward that is gained when executing the transition (s,a,λ,s') such that $a \in A$, which is given by $trans\_rew(s,a,\lambda,s')$.

(iii) $ER_{\rm r}(s,A,cf)$ is the existential state reward with respect to A and cf that is accumulated while staying in s, which is given by $choose\_state\_rew(s,A,cf)$.

(iv) $ER_{\rm i}(s,A,cf)$ is the existential transition reward with respect to A and cf that is gained when executing the transition returned by $choose\_trans(s,A,cf)$, which is given by $choose\_trans\_rew(s,A,cf)$. Similarly, $\phi(s,A,cf)$ is the frequency of such a transition, which is given by $\phi(choose\_trans(s,A,cf))$.

4 The Measure Definition Mechanism of MSL

MSL is equipped with a component-oriented measure definition mechanism built on top of its core logic. The purpose of this mechanism is related to the usability issue. First, the mechanism allows a performance metric to be given a mnemonic name whenever it is derived from a reward structure specified through a set of formula schemas of the MSL core logic. Second, it allows a performance metric to be parameterized with respect to component activities and component behaviors. Third, assumed that the identifier of a performance metric denotes the value of the metric computed on a certain ACTMC,

it allows metric identifiers to be combined through the usual arithmetical operators and mathematical functions.

The syntax for defining a performance measure in MSL, possibly parameterized with respect to a set of component-oriented arguments, is the following:

$$MEASURE \ \triangleleft name \triangleright (\triangleleft parameters \triangleright) \ IS \ \triangleleft body \triangleright$$

In practice, we can envision to deal with libraries of basic measure definitions and derived measure definitions. The body of a basic measure definition is a set of formula schemas of the MSL core logic. By contrast, the body of a derived measure definition is an expression involving identifiers of previously defined metrics (each denoting the value of the corresponding measure computed on a given ACTMC), arithmetical operators, and mathematical functions.

The parameters of the metric identifier can comprise component activities as well as component behaviors together with possibly associated real numbers. The component activities and the component behaviors result in the activity sets occurring in the quantifications of the MSL formula schemas, whereas the real numbers express the reward contributions of the activities within the MSL formula schemas (i.e. they are used in the definition of the *rew* function).

Using this mechanism, with MSL it is possible to define typical instant-of-time performance measures in a component-oriented way. The idea is that the difficulties with measure specification should be hidden inside the definition body, so that the designer has only to provide component-oriented actual parameters when using the metric identifier. To illustrate this point, we now consider the following four classes of performance measures frequently recurring both in queueing theory and in practice: system throughput, resource utilization, mean queue length, and mean response time.

A definition for the system throughput that is easy to use should only request the designer to specify the component activities contributing to the throughput, while a unitary transition reward is transparently associated in the definition body with each of such activities. Using the dot notation for expressing the component activities in the form $C.a$, we have the following definition for the throughput:

$$MEASURE \ throughput(C_1.a_1, \ldots, C_n.a_n) \ IS$$
$$\forall a \in \{C_1.a_1, \ldots, C_n.a_n\}(is\_trans(s, a, \lambda, s') \Rightarrow eq(trans\_rew(s, a, \lambda, s'), 1))$$

According to the ACTMC interpretation of the MSL core logic, the definition above means that each transition labeled with an activity in $\{C_1.a_1, \ldots, C_n.a_n\}$ must be given a unitary instantaneous reward. An equivalent way to define the same measure is to specify that the rate at which each state accumulates reward is the sum of the rates of the activities contributing to the throughput that are enabled at that state:

$$\begin{array}{l} \mathit{MEASURE}\ \mathit{throughput}(C_1.a_1, \ldots, C_n.a_n)\ \mathit{IS} \\ \quad \forall a \in \{C_1.a_1, \ldots, C_n.a_n\} \\ \qquad (\mathit{is\_trans}(s, a, \lambda, s') \Rightarrow \mathit{eq}(\mathit{partial\_contrib}(s, a, \lambda, s'), \mathit{rew}(a, \lambda))) \Rightarrow \\ \qquad\quad \mathit{eq}(\mathit{state\_rew}(s), \mathit{sum\_partial\_contrib}(s, \{C_1.a_1, \ldots, C_n.a_n\})) \end{array}$$

where $\mathit{rew}(a, \lambda) = \lambda$ whenever $a = C_i.a_i$ for some $i = 1, \ldots, n$.

In the case of the utilization of a resource, it should be enough for the designer to specify the component activities modeling the utilization of that resource, while a unitary reward is transparently associated in the definition body with each state in which at least one of such activities is enabled:

$$\begin{array}{l} \mathit{MEASURE}\ \mathit{utilization}(C.a_1, \ldots, C.a_n)\ \mathit{IS} \\ \quad \exists a \in \{C.a_1, \ldots, C.a_n\}(\mathit{is\_trans}(s, a, \lambda, s')) \Rightarrow \\ \qquad \mathit{eq}(\mathit{state\_rew}(s), \mathit{choose\_state\_rew}(s, \{C.a_1, \ldots, C.a_n\}, \min)) \end{array}$$

where $\mathit{rew}(a, \_) = 1$ whenever $a = C.a_i$ for some $i = 1, \ldots, n$. According to the ACTMC interpretation of the MSL core logic, the definition above means that each state enabling at least one activity in $\{C.a_1, \ldots, C.a_n\}$ must be given a unitary rate reward. Note that resource utilization is a special case of performance measure representing the probability of being in a state in which some activities can be carried out.

The mean queue length, which represents the mean number of customers waiting for service, should only require the designer to specify the number of customers in each part of the behavior of the component managing the customer queueing. Using the dot notation for expressing the component behavior parts in the form $C.B$, we have the following definition:

$$\begin{array}{l} \mathit{MEASURE}\ \mathit{mean\_queue\_length}(C.B_1(k_1), \ldots, C.B_n(k_n))\ \mathit{IS} \\ \quad \exists a \in \mathit{activities}(C.B_1)(\mathit{is\_trans}(s, a, \lambda, s')) \Rightarrow \\ \qquad \mathit{eq}(\mathit{state\_rew}(s), \mathit{choose\_state\_rew}(s, \mathit{activities}(C.B_1), \min)) \\ \quad \vdots \\ \quad \exists a \in \mathit{activities}(C.B_n)(\mathit{is\_trans}(s, a, \lambda, s')) \Rightarrow \\ \qquad \mathit{eq}(\mathit{state\_rew}(s), \mathit{choose\_state\_rew}(s, \mathit{activities}(C.B_n), \min)) \end{array}$$

where *activities* is a built-in function that retrieves the set of the activities that a behavior part can perform, and $\mathit{rew}(a, \_) = k_i$ whenever $a \in \mathit{activities}(C.B_i)$ for some $i = 1, \ldots, n$. According to the ACTMC interpretation of the MSL core logic, the definition above means that each state comprising one of the considered behavior parts, from which at least one transition departs that is labeled with one of the activities occurring in the behavior, must be given as rate reward the number specified for that behavior.

The mean response time can be defined similarly to *mean_queue_length*

thanks to Little's law by taking into account the arrival rate λ of the customers. This is done by replacing k_i with k_i/λ for $i = 1, \ldots, n$.

All the examples shown so far illustrate basic measure definitions. An example of a derived metric is given by the mean queue length for a system that has m queueing components $C_1, C_2, \ldots, C_m$, which is defined as follows:

$$\begin{aligned} &\mathit{MEASURE}\ \mathit{total\_mean\_queue\_length}(C_1.B_{1,1}(k_{1,1}), \ldots, C_1.B_{1,n_1}(k_{1,n_1}),\\ &\qquad\qquad C_2.B_{2,1}(k_{2,1}), \ldots, C_2.B_{2,n_2}(k_{2,n_2}),\\ &\qquad\qquad \vdots\\ &\qquad\qquad C_m.B_{m,1}(k_{m,1}), \ldots, C_m.B_{m,n_m}(k_{m,n_m}))\ \mathit{IS}\\ &\quad \mathit{mean\_queue\_length}(C_1.B_{1,1}(k_{1,1}), \ldots, C_1.B_{1,n_1}(k_{1,n_1})) +\\ &\quad \mathit{mean\_queue\_length}(C_2.B_{2,1}(k_{2,1}), \ldots, C_2.B_{2,n_2}(k_{2,n_2})) +\\ &\quad \vdots\\ &\quad \mathit{mean\_queue\_length}(C_m.B_{m,1}(k_{m,1}), \ldots, C_m.B_{m,n_m}(k_{m,n_m})) \end{aligned}$$

5 SPA Interpretation of MSL

In this section we provide an interpretation of the core logic of MSL based on SPA, aiming at verifying whether a framework can be developed in which system models can be compositionally manipulated without altering the value of instant-of-time performance measures specified with MSL. We shall also address some issues concerned with the enhanced expressiveness of MSL with respect to traditional reward structures when dealing with modeling notations like SPA, in which the concept of state is implicit.

5.1 SPA with Universal and Existential Rewards

In order to achieve compositionality, we adopt a subcalculus of $\mathrm{EMPA}_{\mathrm{gr}_1}$ [7] extended with universal and existential rewards. In this calculus every action is a tuple like $<a, \lambda, (uy, ub, ey, eb)>$ or $<a, *_w, (*, *, *, *)>$, where:

- $a \in Act$ is the action type (τ if invisible).
- $\lambda \in \mathbb{R}_{>0}$ expresses the rate of an exponentially timed action.
- $*_w$ denotes a passive action with reactive weight $w \in \mathbb{R}_{>0}$.
- (uy, ub, ey, eb) is a reward 4-tuple, where every reward belongs to $\mathbb{R}$ if the action is exponentially timed, or is $*$ if the action is passive. In the case of an exponentially timed action, such rewards express the action contribution $rew(a, \lambda)$ occurring in the four MSL formula schemas of Def. 3.1, where the universal yield reward uy is related to (i), the universal bonus reward ub is related to (ii), the existential yield reward ey is related to (iii), and the existential bonus reward eb is related to (iv).

Hence, a performance measure defined through a MSL formula schema quantified with respect to an activity set A is rendered by inserting the rewards $rew(a, \lambda)$ occurring in the MSL formula schema into the appropriate position of the reward 4-tuple of the exponentially timed actions whose type is in A.

In order to compute the instant-of-time value of a performance measure defined in MSL, in accordance with the ACTMC interpretation of the core logic of MSL the universal yield rewards are governed by the partial contribution additivity assumption. This means that the overall rate at which reward is accumulated while staying in a certain state is the sum of the universal yield rewards associated with the exponentially timed actions whose type is in A that are enabled at that state. By contrast, the existential yield rewards of the actions simultaneously enabled at a given state cannot be summed up, as this would conflict with the intuition behind the existential quantification. Instead a choice function is applied to the existential yield rewards of the exponentially timed actions whose type is in A that are enabled at that state. Similarly, in the case of the universal and existential bonus rewards we can argue in accordance with the ACTMC interpretation of the core logic of MSL.

The formal syntax and semantics for this calculus with universal and existential rewards is the natural extension of the syntax and the semantics presented in [7]. As usual, we restrict ourselves to the set $\mathcal{G}$ of the closed and guarded terms and we denote by $\mathcal{E}$ the set of the performance closed terms of $\mathcal{G}$.

5.2 Congruence Result

We now show that it is possible to define a performance-measure-sensitive congruence for a SPA extended with universal and existential rewards. This means that we can provide a formal framework for the compositional manipulation of system models that does not alter the value of the performance measures expressed in MSL.

The reward-based Markovian behavioral equivalence that we are going to introduce is an extension of the bisimulation-based one of [6]. In essence, this equivalence aggregates the transitions labeled with the same type and departing from the same state that reach states of the same equivalence class. More precisely, the rates and the universal yield rewards of such transitions are summed up, while their universal bonus rewards are multiplied by the probability of executing the corresponding transitions before being summed up. The existential yield rewards and the existential bonus rewards are subject to the application of a choice function instead of the addition. By so doing, we are consistent with the ACTMC interpretation summarized through equation (2).

Definition 5.1 We define partial function aggregated rate-reward $RR : \mathcal{G} \times Act \times \{\exp, *\} \times 2^{\mathcal{G}} \not\to \mathbb{R}_{>0} \times (\mathbb{R} \cup \{*\})^4$ by:

$$RR(E, a, l, C) = (Rate(E, a, l, C), UY(E, a, l, C), UB(E, a, l, C), EY(E, a, l, C), EB(E, a, l, C))$$

where:

$$Rate(E,a,\exp,C) = \sum\{\!|\, \lambda \mid \exists uy, ub, ey, eb.\, \exists E' \in C.\, E \xrightarrow{a,\lambda,(uy,ub,ey,eb)} E' \,|\!\}$$

$$Rate(E,a,*,C) = \sum\{\!|\, w \mid \exists E' \in C.\, E \xrightarrow{a,*_w,(*,*,*,*)} E' \,|\!\}$$

$$UY(E,a,\exp,C) = \sum\{\!|\, uy \mid \exists \lambda, ub, ey, eb.\, \exists E' \in C.\, E \xrightarrow{a,\lambda,(uy,ub,ey,eb)} E' \,|\!\}$$

$$UB(E,a,\exp,C) = \sum\{\!|\, \tfrac{\lambda}{Rate(E,a,\exp,C)} \cdot ub \mid \exists uy, ey, eb.\, \exists E' \in C.\, E \xrightarrow{a,\lambda,(uy,ub,ey,eb)} E' \,|\!\}$$

$$EY(E,a,\exp,C) = cf\{\!|\, ey \mid \exists \lambda, uy, ub, eb.\, \exists E' \in C.\, E \xrightarrow{a,\lambda,(uy,ub,ey,eb)} E' \,|\!\}$$

$$EB(E,a,\exp,C) = cf\{\!|\, \tfrac{\lambda}{Rate(E,a,\exp,C)} \cdot eb \mid \exists uy, ub, ey.\, \exists E' \in C.\, E \xrightarrow{a,\lambda,(uy,ub,ey,eb)} E' \,|\!\}$$

$$UY(E,a,*,C) = UB(E,a,*,C) = EY(E,a,*,C) = EB(E,a,*,C) = *$$

where cf is the choice function and $RR(E,a,l,C) = \bot$ whenever the multisets above are empty. ■

Definition 5.2 An equivalence relation $\mathcal{B} \subseteq \mathcal{G} \times \mathcal{G}$ is a reward-based Markovian bisimulation iff, whenever $(E_1, E_2) \in \mathcal{B}$, then for all action types $a \in Act$, levels $l \in \{\exp, *\}$, and equivalence classes $C \in \mathcal{G}/\mathcal{B}$:

$$RR(E_1,a,l,C) = RR(E_2,a,l,C)$$

■

It is easy to see that the union of all the reward-based Markovian bisimulations is the largest reward-based Markovian bisimulation. Such a union, denoted $\sim_{\rm RMB}$, is called reward-based Markovian bisimilarity.

Theorem 5.3 *Whenever cf is commutative, associative, and distributive with respect to the multiplication by non-negative numbers, then* $\sim_{\rm RMB}$ *is a congruence with respect to all the process algebraic operators as well as recursion.*

Proof. *As far as the rates and the universal rewards are concerned, the proof is the same as that of the corresponding theorem of [6,7]. In the case of the existential rewards, it is sufficient to observe that the properties required about cf are exactly the same as the ones used in the case of the universal rewards when working with the addition operator.* □

For instance, min and max are choice functions that satisfy the hypothesis of the congruence theorem above.

Theorem 5.4 *Let* $E_1, E_2 \in \mathcal{E}$*. If* $E_1 \sim_{\rm RMB} E_2$ *then the value of the reward-based performance measure is the same for* E_1 *and* E_2*.*

Proof. *We can argue similarly as done in the proof of Thm. 5.3.* □

5.3 Axiomatization

We now provide a sound and complete axiomatization of $\sim_{\mathrm{RMB}}$. Let $\mathcal{A}^{\mathrm{RMB}}$ be the set of axioms of [6] extended with universal and existential rewards, where the axioms expressing the aggregation of rates and rewards are recast as follows:

$$<a,\lambda_1,(uy_1,ub_1,ey_1,eb_1)>.E + <a,\lambda_2,(uy_2,ub_2,ey_2,eb_2)>.E =$$
$$<a,\lambda_1+\lambda_2,(uy_1+uy_2,\tfrac{\lambda_1}{\lambda_1+\lambda_2}\cdot ub_1+\tfrac{\lambda_2}{\lambda_1+\lambda_2}\cdot ub_2,$$
$$cf(ey_1,ey_2),cf(\tfrac{\lambda_1}{\lambda_1+\lambda_2}\cdot eb_1,\tfrac{\lambda_2}{\lambda_1+\lambda_2}\cdot eb_2))>.E$$
$$<a,*_{w_1},(*,*,*,*)>.E + <a,*_{w_1},(*,*,*,*)>.E =$$
$$<a,*_{w_1+w_2},(*,*,*,*)>.E$$

Note that such a pair of axioms reflects the definition of the aggregated rate-reward function RR (see Def. 5.1).

Theorem 5.5 *Whenever cf satisfies the same constraints as Thm. 5.3, then the deductive system $Ded(\mathcal{A}^{\mathrm{RMB}})$ is sound and complete for $\sim_{\mathrm{RMB}}$ over the set of the non-recursive terms of $\mathcal{G}$.*

Proof. *We can argue similarly as done in the proof of Thm. 5.3.* □

In order to augment the aggregation power of $\sim_{\mathrm{RMB}}$ without losing the congruence property, as shown in [6] it is possible to jointly consider universal yield rewards and universal bonus rewards, thus resulting in a normal form in which only universal yield rewards are used. Indeed, an axiom like $<a,\lambda_1,(uy_1,ub_1,0,0)>.E + <a,\lambda_2,(uy_2,ub_2,0,0)>.E = <a,\lambda_1+\lambda_2,(uy_1+uy_2+\lambda_1\cdot ub_1+\lambda_2\cdot ub_2,0,0,0)>.E$ would be correct. Instead, in the case of the existential rewards, a similar axiom would cause a loss of compositionality. Intuitively, applying in an interleaved way the addition and the choice function does not preserve the value of the performance measures, as shown below in the case the choice function is max.

Example 5.6 Let $E \triangleq <a,\lambda_1,(0,0,ey_1,eb_1)>.E + <a,\lambda_2,(0,0,ey_2,eb_2)>.E$, whose underlying ACTMC has a single state with a single self-loop transition labeled with a whose rate is $\lambda_1+\lambda_2$. Then consider a performance measure for E that is existentially quantified with respect to $\{a\}$. The instant-of-time value of such a performance measure is given by $ER_{\mathrm{r}}(E,\{a\},\max) + ER_{\mathrm{i}}(E,\{a\},\max) = \max(ey_1,ey_2)+\max(\lambda_1\cdot eb_1,\lambda_2\cdot eb_2)$. By contrast, if we express the existential bonus rewards in terms of existential yield rewards, we obtain $\max(ey_1+\lambda_1\cdot eb_1, ey_2+\lambda_2\cdot eb_2)$. Now assume $ey_1=1$, $ey_2=2$, and $\lambda_1\cdot eb_1=2$, $\lambda_2\cdot eb_2=1$. In the former case we obtain the value $\max(1,2)+\max(2,1)=2+2=4$. On the other hand, in the latter case we obtain a different value, which is $\max(1+2,2+1)=3$. ■

5.4 Expressiveness

We conclude by observing that the introduction of existential rewards enhances the expressiveness with respect to [6,7]. For instance let us consider again the queueing system example of Sect. 2.2. Suppose we wish to measure the overall system utilization, i.e. the percentage of time during which at least one server is busy. To do that, we try to extend the Æmilia specification of Sect. 2.1 by inserting rewards into the actions occurring in the specification. We soon realize that the only way to carry out this task correctly is to associate a unitary existential yield reward with any `serve` action, which corresponds to using a MSL formula of type (iii) as done in Sect. 3. This is because the only state of Fig. 1 in which two `serve` transitions can be executed must be counted only once. Note that this would not be possible if we had at our disposal only universal rewards.

6 Conclusion

In this paper we have addressed the problem of making the specification of performance measures a task that can be carried out in a component-oriented fashion. As a step towards the solution of this usability-related problem, we have proposed MSL, a formal notation for specifying performance measures. MSL relies on a simple first-order logic, which we have interpreted both on ACTMCs and on SPA. The second interpretation has been useful to show that the core logic of MSL supports performance-sensitive compositional reasoning.

As far as usability is concerned, MSL is equipped with a measure definition mechanism, through which it is possible to associate mnemonic names with performance metrics derived from a reward structure specified through a set of MSL core logic formulas, as well as to parameterize them with respect to component activities and behaviors. The objective of this component-oriented measure definition mechanism is to manage the numeric values of the rewards as transparently as possible. In this way, while the definition of a basic metric may be a task for a performance expert, the definition of derived metrics and the use of any metric definition should be affordable by non-specialists. MSL has been exemplified on a number of typical performance measures.

Due to the introduction of the existential rewards, in the case of modeling notations in which the concept of state is implicit MSL is able to express an increased number of performance measures with respect to previous reward-based notations. However, it is still difficult (if not impossible) to define reachability-like performance measures. To this purpose, we plan to investigate a way to integrate MSL and CSL [3] in order to achieve an enhanced expressiveness while retaining a satisfactory degree of usability.

References

[1] A. Aldini and M. Bernardo, *"On the Usability of Process Algebra: An Architectural View"*, in *Theoretical Computer Science* 335:281–329, Elsevier, 2005.

[2] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis, *"Modelling with Generalized Stochastic Petri Nets"*, John Wiley & Sons, 1995.

[3] C. Baier, J.-P. Katoen, and H. Hermanns, *"Approximate Symbolic Model Checking of Continuous Time Markov Chains"*, in Proc. of the *10th Int. Conf. on Concurrency Theory (CONCUR 1999)*, LNCS 1664:146-162, 1999.

[4] C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen, *"On the Logical Characterisation of Performability Properties"*, in Proc. of the *27th Int. Coll. on Automata, Languages and Programming (ICALP 2000)*, LNCS 1853:780-792, 2000.

[5] S. Balsamo, M. Bernardo, and M. Simeoni, *"Performance Evaluation at the Software Architecture Level"*, in *Formal Methods for Software Architectures*, LNCS 2804:207-258, 2003.

[6] M. Bernardo and M. Bravetti, *"Reward Based Congruences: Can We Aggregate More?"*, in Proc. of the Joint Int. Workshop on Process Algebra and Performance Modelling and Probabilistic Methods in Verification (PAPM/PROBMIV 2001), LNCS 2165:136-151, 2001.

[7] M. Bernardo and M. Bravetti, *"Performance Measure Sensitive Congruences for Markovian Process Algebras"*, in Theoretical Computer Science 290:117-160, 2003.

[8] M. Bernardo, L. Donatiello, and P. Ciancarini, *"Stochastic Process Algebra: From an Algebraic Formalism to an Architectural Description Language"*, in *Performance Evaluation of Complex Systems: Techniques and Tools*, LNCS 2459:236-260, 2002.

[9] G. Ciardo, J. Muppala, and K.S. Trivedi, *"On the Solution of GSPN Reward Models"*, in Performance Evaluation 12:237-253, 1991.

[10] G. Clark, *"Formalising the Specification of Rewards with PEPA"*, in Proc. of the *4th Workshop on Process Algebras and Performance Modelling (PAPM 1996)*, CLUT, pp. 139-160, 1996.

[11] G. Clark, S. Gilmore, and J. Hillston, *"Specifying Performance Measures for PEPA"*, in Proc. of the *5th AMAST Int. Workshop on Formal Methods for Real Time and Probabilistic Systems (ARTS 1999)*, LNCS 1601:211-227, 1999.

[12] B.R. Haverkort and K.S. Trivedi, *"Specification Techniques for Markov Reward Models"*, in Discrete Event Dynamic Systems: Theory and Applications 3:219-247, 1993.

[13] H. Hermanns, *"Interactive Markov Chains"*, LNCS 2428, 2002.

[14] J. Hillston, *"A Compositional Approach to Performance Modelling"*, Cambridge University Press, 1996.

[15] R.A. Howard, *"Dynamic Probabilistic Systems"*, John Wiley & Sons, 1971.

[16] L. Kleinrock, *"Queueing Systems"*, John Wiley & Sons, 1975.

[17] M.A. Qureshi and W.H. Sanders, *"Reward Model Solution Methods with Impulse and Rate Rewards: An Algorithm and Numerical Results"*, in Performance Evaluation 20:413-436, 1994.

[18] W.H. Sanders and J.F. Meyer, *"A Unified Approach for Specifying Measures of Performance, Dependability, and Performability"*, in Dependable Computing and Fault Tolerant Systems 4:215-237, 1991.

[19] W.J. Stewart, *"Introduction to the Numerical Solution of Markov Chains"*, Princeton University Press, 1994.

[20] J.E. Voeten, *"Temporal Rewards for Performance Evaluation"*, in Proc. of the *8th Int. Workshop on Process Algebra and Performance Modelling (PAPM 2000)*, Carleton Scientific, pp. 511-522, 2000.