Available online at www.sciencedirect.com
ScienceDirect
Theoretical Computer Science 382 (2007) 3–23
Theoretical Computer Science
www.elsevier.com/locate/tcs

Mixing logics and rewards for the component-oriented specification of performance measures

Alessandro Aldini*, Marco Bernardo

Università di Urbino "Carlo Bo", Istituto di Scienze e Tecnologie dell'Informazione, 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 logics. In this paper we propose an approach that combines logics and rewards, together with a definition mechanism that allows performance measures to be specified in a component-oriented way, thus facilitating the task for non-experts. The resulting Measure Specification Language (MSL) 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 implicit-state modeling notations.

Keywords: Continuous-time Markov chains; Stochastic process algebra; Reward structures; Component oriented modeling; Measure specification language

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., [20,19,11] 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 [28]).

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 comprising of numerous interacting parts.

This usability issue has been tackled with the development of Æmilia [12,8], an architectural description language based on EMPA_{gr} [11] for the textual and graphical representation of system families. Æmilia clearly separates

* Corresponding author. Fax: +39 0722 4475.
E-mail address: aldini@sti.uniurb.it (A. Aldini).

0304-3975/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2007.05.006

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 above have been endowed with companion notations for the specification of the performance measures of interest. According to the classifications proposed in [27,18], 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 states. 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 [21] 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 [14,26]. In the SPA case, the reward association is harder because the modeling notation is action-based; hence the concept of state is implicit. In [15,16], the CTMC states to which certain rewards have to be attached are singled out by means of suitable modal logic formulas, whereas in [11,10] the rewards are directly written into the actions occurring in the system specifications, and are then transferred to the CTMC states and transitions during the CTMC construction. In [29], 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 [6], 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 [7], 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 [10]; 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 from 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. Such a mixed 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. Component-orientation is then achieved in MSL by means of a mechanism to define measures that are parameterized with respect to component activities and component behaviors. In particular, such a mechanism allows performance metrics to be defined in a transparent way in terms of the activities that individual components or parts of their behavior can carry out, or in terms of specific local behaviors that describe the components of interest. 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 improved usability and expressiveness of MSL is shown through a case study originally conducted in [1] with Æmilia and action-based rewards.

The rest of the paper, which is a full and revised version of [4], is organized as follows. In Section 2, we recall some background about component-oriented system modeling, action-labeled CTMCs, and reward structures. In Section 3, we present MSL by defining its core logic together with its action-labeled CTMC interpretation. In Section 4, we present the measure definition mechanism associated with MSL. In Section 5, we provide the SPA-based interpretation for the core logic of MSL. In Section 6, we reconsider a case study about the analysis of the energy consumption for a battery-powered device employing a dynamic power manager. Finally, in Section 7, we conclude by reporting comparisons with related work and perspectives on future developments.

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 [3], the 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,[1] 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 e.g. the direction (input vs. output) or the form (point-to-point, broadcast, server-clients, etc.) of the communication they can be involved in.

The description of the system topology should be provided by declaring the instances of the component types that form the system, together with a 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.

In the following, we consider as an illustrative example a queuing system $M/M/2$ with arrival rate $\lambda \in \mathbb{R}_{>0}$ and service rates $\mu_1, \mu_2 \in \mathbb{R}_{>0}$ [25]. This system represents a service center with no buffer equipped with two servers processing requests at rates 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 with rate λ. Whenever both servers are idle, an incoming customer has the same probability of being 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, such component types would be modeled as follows:

```
ARCHI_TYPE QS_M_M_2(rate lambda, rate mu1, rate mu2)

  ARCHI_BEHAVIOR

    ARCHI_ELEM_TYPE Arrivals_Type(rate arrival_rate)
      BEHAVIOR
```

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

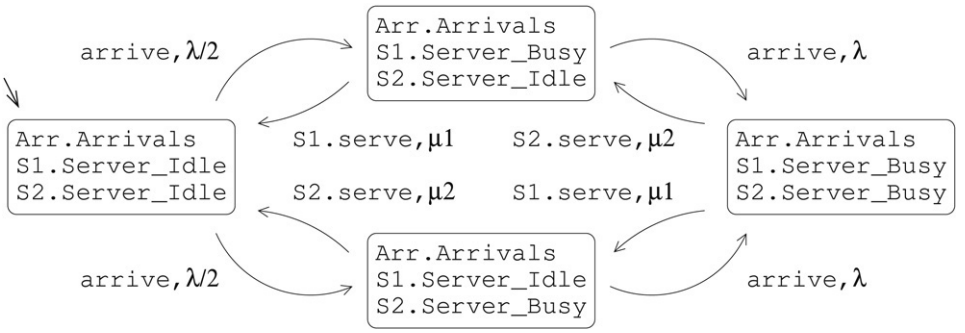

Fig. 1. ACTMC model of the queueing system example.

```
    Arrivals(void) = <arrive, exp(arrival_rate)> . Arrivals()
  INPUT  INTERACTIONS void
  OUTPUT INTERACTIONS OR arrive

ARCHI_ELEM_TYPE Server_Type(rate service_rate)
  BEHAVIOR
    Server_Idle(void) =
      <arrive, _> . Server_Busy();
    Server_Busy(void) =
      <serve, exp(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. Action-labeled CTMCs 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 s 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 $[z_1, z_2, \ldots, z_n]$, which are the current behaviors of the individual components. We denote by S_{local} the set of local states of $\mathcal{M}$. Each transition corresponds instead either to an activity performed by a single component in isolation, or to a set of attached interactions executed simultaneously by several communicating components. As an example, Fig. 1 shows the ACTMC underlying the queuing system modeled with Æmilia in Section 2.1.

As far as the analysis of ACTMC-based component-oriented models is concerned, the typical approach to performance measure specification relying on reward structures can be followed. This requires associating real

numbers with system behaviors and activities, which are then transferred to the proper states (as rate rewards) and transitions (as 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:

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

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, for instance, that in the queuing 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 $\mathcal{M} = (S, Act, \longrightarrow_{\mathcal{M}}, s_0)$. In order to achieve a satisfactory degree of expressiveness, at least six formula schemas have to be present in the core logic, for reasons that we are going to elucidate.

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. As far as state rewards are concerned, while it is straightforward to define them for state-based modeling notations by means of rewards directly associated with the local states, in the case of an action-based modeling notation they can only be expressed indirectly, i.e. by means of rewards associated with the activities enabled in the states. Thus, there are three options: direct state rewards, indirect state rewards, and transition rewards.

On the other hand, the designer has to be allowed to decide whether all the local states in a given set $Z \subseteq S_{\mathrm{local}}$ or all the activities in a given set $A \subseteq Act$ contribute to the value of a certain performance measure, or only one element of the set does. Therefore, there are two options: universal quantification and existential quantification. In conclusion, the combination of the two sets of options results in six alternatives to be made available to the designer.

Definition 3.1. The core logic of MSL is a first-order logic composed of the universal closure with respect to S of the following six formula schemas:

(i) $\forall z \in Z(is\_local(z, s) \Rightarrow eq(lstate\_contrib(z, s), lstate\_rew(z))) \Rightarrow$
$eq(state\_rew(s), sum\_lstate\_contrib(s, Z))$

(ii) $\forall a \in A(is\_trans(s, a, \lambda, s') \Rightarrow eq(act\_contrib(s, a, \lambda, s'), act\_rew(a, \lambda))) \Rightarrow$
$eq(state\_rew(s), sum\_act\_contrib(s, A))$

(iii) $\forall a \in A(is\_trans(s, a, \lambda, s') \Rightarrow eq(trans\_rew(s, a, \lambda, s'), act\_rew(a, \lambda)))$
(iv) $\exists z \in Z(is\_local(z, s)) \Rightarrow eq(state\_rew(s), choose\_lstate\_rew(s, Z, cf))$
(v) $\exists a \in A(is\_trans(s, a, \lambda, s')) \Rightarrow eq(state\_rew(s), choose\_act\_rew(s, A, cf))$
(vi) $\exists a \in A(is\_trans(s, a, \lambda, s')) \Rightarrow eq(trans\_rew(choose\_trans(s, A, cf)), choose\_trans\_rew(s, A, cf))$. ■

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

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

$$Z = \{\texttt{S1.Server\_Busy}, \texttt{S2.Server\_Busy}\}$$
$$lstate\_rew(\texttt{S1.Server\_Busy}) = \mu_1,\ lstate\_rew(\texttt{S2.Server\_Busy}) = \mu_2$$

or, equivalently, through a formula of type (ii) such that:

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

or, equivalently, through a formula of type (iii) where:

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

Similarly, the throughput of S1 alone can be specified through a formula of type (i) where:

$$Z = \{\texttt{S1.Server\_Busy}\}$$
$$lstate\_rew(\texttt{S1.Server\_Busy}) = \mu_1$$

or through a formula of type (ii) such that:

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

or through a formula of type (iii) where:

$$A = \{\texttt{S1.serve}\}$$
$$act\_rew(\texttt{S1.serve}, \_) = 1$$

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 at least one of its local states is in Z. The value of the contribution will have to be selected by applying a choice function cf to the direct state rewards $lstate\_rew(z)$ associated with the local states in Z that are part of the current state. By choice function, we mean a function that simply returns one of its arguments, like e.g. max and min. Similarly, the second 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 the contribution is determined by applying a choice function cf to the indirect state rewards $act\_rew(a, \lambda)$ associated with the transitions labeled with an activity $a \in A$ that depart from the current state. The third existential formula schema specifies 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 transition rewards $act\_rew(a, \lambda)$

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 queuing system example, the system utilization can be specified through a formula of type (iv) where:

$$\begin{aligned} Z &= \{\texttt{S1.Server\_Busy}, \texttt{S2.Server\_Busy}\} \\ \mathit{lstate\_rew}(\texttt{S1.Server\_Busy}) &= \mathit{lstate\_rew}(\texttt{S2.Server\_Busy}) = 1 \\ \mathit{cf} &= \min \end{aligned}$$

or, equivalently, through a formula of type (v) such that:

$$\begin{aligned} A &= \{\texttt{S1.serve}, \texttt{S2.serve}\} \\ \mathit{act\_rew}(\texttt{S1.serve}, \_) &= \mathit{act\_rew}(\texttt{S2.serve}, \_) = 1 \\ \mathit{cf} &= \min \end{aligned}$$

Similarly, the utilization of S1 alone can be specified through a formula of type (iv) where:

$$\begin{aligned} A &= \{\texttt{S1.Server\_Busy}\} \\ \mathit{lstate\_rew}(\texttt{S1.Server\_Busy}) &= 1 \\ \mathit{cf} &= \min \end{aligned}$$

or through a formula of type (v) such that:

$$\begin{aligned} A &= \{\texttt{S1.serve}\} \\ \mathit{act\_rew}(\texttt{S1.serve}, \_) &= 1 \\ \mathit{cf} &= \min \end{aligned}$$

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

$$\begin{aligned} A &= \{\texttt{arrive}\} \\ \mathit{act\_rew}(\texttt{arrive}, \_) &= 1 \\ \mathit{cf} &= \min \end{aligned}$$

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 Definition 3.1. As a shorthand, we use the notation $z \in s$ to express that $s = [z_1, z_2, \ldots, z_n]$ with $z = z_i$ for some $1 \leq i \leq n$. We also denote by CF the set of the choice functions that can occur in the existential formula schemas:

$$CF = \{f : 2^{\mathbb{R}} \to \mathbb{R} \mid f(\emptyset) = 0 \wedge \forall n \in \mathbb{N}_{>0}.\ f(\{x_1, \ldots, x_n\}) \in \{x_1, \ldots, x_n\}\}$$

- $\mathit{is\_local} : Z \times S \to \{\text{true}, \text{false}\}$ such that:

$$\mathit{is\_local}(z, s) = \begin{cases} \text{true} & \text{if } z \in s \\ \text{false} & \text{otherwise} \end{cases}$$

- $\mathit{is\_trans} : S \times \mathit{Act} \times \mathbb{R}_{>0} \times S \to \{\text{true}, \text{false}\}$ such that:

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

- $\mathit{eq} : \mathbb{R} \times \mathbb{R} \to \{\text{true}, \text{false}\}$ such that:

$$\mathit{eq}(x, y) = \begin{cases} \text{true} & \text{if } x = y \\ \text{false} & \text{otherwise} \end{cases}$$

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

- *lstate_contrib* : $Z \times S \to \mathbb{R}$ such that $lstate\_contrib(z, s)$ is the partial contribution given by local state z of s to the rate at which the state reward is gained at s.
- *act_contrib* : $\longrightarrow_{\mathcal{M}} \to \mathbb{R}$ such that $act\_contrib(s, a, \lambda, s')$ is the partial contribution given by transition (s, a, λ, s') to the rate at which the state reward is gained at s.
- *sum_lstate_contrib* : $S \times 2^{S_{\text{local}}} \to \mathbb{R}$ such that:

$$sum\_lstate\_contrib(s, Z) = \sum_{z \in Z \wedge is\_local(z,s)} lstate\_contrib(z, s)$$

  where the sum is zero whenever no local state $z \in Z$ is part of s.
- *sum_act_contrib* : $S \times 2^{Act} \to \mathbb{R}$ such that:

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

  where the sum is zero whenever no transition labeled with $a \in A$ can be executed by s.
- *choose_lstate_rew* : $S \times 2^{S_{\text{local}}} \times CF \to \mathbb{R}$ such that:

$$choose\_lstate\_rew(s, Z, cf) = cf(\{\!|\, lstate\_rew(z) \mid z \in Z \wedge is\_local(z, s) \,|\!\})$$

- *choose_act_rew* : $S \times 2^{Act} \times CF \to \mathbb{R}$ such that:

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

- *choose_trans* : $S \times 2^{Act} \times CF \multimap \longrightarrow_{\mathcal{M}}$ such that:

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

  iff there are transitions labeled with an action in A executable by s and:

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

- *choose_trans_rew* : $S \times 2^{Act} \times CF \multimap \mathbb{R}$ such that:

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

  iff for some $s' \in S$:

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

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

$$\begin{aligned} &\sum_{s \in S} (UR_{\text{r}}^{\text{ls}}(s, Z) + UR_{\text{r}}^{\text{a}}(s, A)) \cdot \pi(s) + \sum_{a \in A} \sum_{(s,a,\lambda,s') \in \longrightarrow_{\mathcal{M}}} UR_{\text{i}}(s, a, \lambda, s') \cdot \phi(s, a, \lambda, s') \\ &\quad + \sum_{s \in S} (ER_{\text{r}}^{\text{ls}}(s, Z, cf) + ER_{\text{r}}^{\text{a}}(s, A, cf)) \cdot \pi(s) + \sum_{s \in S} ER_{\text{i}}(s, A, cf) \cdot \phi(s, A, cf) \end{aligned} \tag{2}$$

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

(i) $UR_{\text{r}}^{\text{ls}}(s, Z)$ is the universal state reward with respect to Z that is accumulated while staying in s, which is given by $sum\_lstate\_contrib(s, Z)$.

(ii) $UR_{\text{r}}^{\text{a}}(s, A)$ is the universal state reward with respect to A that is accumulated while staying in s, which is given by $sum\_act\_contrib(s, A)$.

(iii) $UR_{\text{i}}(s, a, \lambda, s')$ 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 $trans\_rew(s, a, \lambda, s')$.

(iv) $ER_{\text{r}}^{\text{ls}}(s, Z, cf)$ is the existential state reward with respect to Z and cf that is accumulated while staying in s, which is given by $choose\_lstate\_rew(s, Z, cf)$.

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

(vi) $ER_{\text{i}}(s, A, cf)$ 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 mentioned in the introduction. 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 behaviors and component activities. Third, given 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 ⊲ *name*⊳ (⊲*parameters*⊳) IS ⊲ *body*⊳

In practice, we can envision how 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 behaviors (together with possibly associated real numbers), as well as component activities. The component behaviors result in the local state sets occurring in the quantifications of the MSL formula schemas (i) and (iv), with the possibly associated real numbers expressing the reward contributions of the local states within the MSL formula schemas (i.e. they are used in the definition of function *lstate_rew*). The component activities, instead, result in the activity sets occurring in the quantifications of the MSL formula schemas (ii), (iii), (v), and (vi).

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 queuing 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 such activity. Using the dot notation for expressing the component activities in the form $C.a$, we have the following definition for the throughput:

MEASURE *throughput_iii*$(C_1.a_1, \ldots, C_n.a_n)$ 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:

MEASURE *throughput_ii*$(C_1.a_1, \ldots, C_n.a_n)$ IS
$\forall a \in \{C_1.a_1, \ldots, C_n.a_n\}$
$(is\_trans(s, a, \lambda, s') \Rightarrow eq(act\_contrib(s, a, \lambda, s'), act\_rew(a, \lambda))) \Rightarrow$
$eq(state\_rew(s), sum\_act\_contrib(s, \{C_1.a_1, \ldots, C_n.a_n\}))$

where $act\_rew(a, \lambda) = \lambda$ whenever $a = C_i.a_i$ for some $1 \leq i \leq n$.

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:

MEASURE *utilization*$(C.a_1, \ldots, C.a_n)$ IS
$\exists a \in \{C.a_1, \ldots, C.a_n\}(is\_trans(s, a, \lambda, s')) \Rightarrow$
$eq(state\_rew(s), choose\_act\_rew(s, \{C.a_1, \ldots, C.a_n\}, \min))$

where $act\_rew(a, \_) = 1$ whenever $a = C.a_i$ for some $1 \leq i \leq n$. 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.

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 queuing. Using the dot notation for expressing the component behavior parts in the form $C.B$, we have the following definition:

$$\begin{array}{l}\texttt{MEASURE}\ mean\_queue\_length(C.B_1(k_1), \ldots, C.B_n(k_n))\ \texttt{IS}\\ \quad \exists z \in \{C.B_1, \ldots, C.B_n\}(is\_local(z, s)) \Rightarrow\\ \quad\quad eq(state\_rew(s), choose\_lstate\_rew(s, \{C.B_1, \ldots, C.B_n\}, \min))\end{array}$$

where $lstate\_rew(z) = k_i$ whenever $z = C.B_i$ for some $1 \leq i \leq n$. According to the ACTMC interpretation of the MSL core logic, the definition above means that each state comprising one of the considered behavior parts must be given as the 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 $1 \leq i \leq n$.

Another useful class of performance measures is the one concerning the probability of being in a specific behavior of an individual component of the system. In this case, it should be enough for the designer to specify the behavior of interest:

$$\begin{array}{l}\texttt{MEASURE}\ behavior\_prob(C.B)\ \texttt{IS}\\ \quad \exists z \in \{C.B\}(is\_local(z, s)) \Rightarrow\\ \quad\quad eq(state\_rew(s), choose\_lstate\_rew(s, \{C.B\}, \min))\end{array}$$

where $lstate\_rew(C.B) = 1$.

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 queuing components $C_1, C_2, \ldots, C_m$, which is defined as follows:

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

As can be noted, the body of this derived measure definition is an arithmetic expression whose atomic constituents are identifiers of basic measure definitions with actual component-oriented parameters.

5. SPA interpretation of MSL

In this section, we provide an interpretation based on SPA of the core logic of MSL. The purpose is to develop a framework in which system models can be compositionally manipulated without altering the value of instant-of-time performance measures specified with MSL.

Whenever a formal description technique like SPA is used to model a system and to represent its performance aspects, rewards are not directly specified at the level of the underlying stochastic process like, e.g., an ACTMC. Instead, they are defined at the level of the process algebraic description, and then automatically inherited by the underlying stochastic process. Therefore, in order to extend an action-based modeling notation like SPA with universal and existential rewards, it is necessary to decide how to represent state and transition rewards at the process algebraic level.

As illustrated in Section 3, on the one hand indirect state rewards and transition rewards are associated with the system activities. Thus, in SPA it will be natural to attach such rewards to the process algebra actions representing these

activities. On the other hand, the direct state rewards are associated with the local states, but these are not explicitly described in an action-based formalism like SPA. Our proposal is to attach state rewards to behavioral equations, as the operational semantic rules make them correspond to the local states.

In this section, 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. Syntax of SPA with universal and existential rewards

Here we adopt a variant of $\text{EMPA}_{\text{gr}_1}$ [11], which we extend with universal and existential rewards. In this calculus, every action α is either exponentially timed or passive:

$$\alpha ::= <a, \lambda, (uy, ub, ey, eb)> \mid <a, *_w, (*, *, *, *)>$$

where:

- $a \in Act$ is the action name (τ if invisible).
- $\lambda \in \mathbb{R}_{>0}$ expresses the rate of an exponentially timed action.
- $*_w$ denotes a passive action (whose duration is unspecified) with reactive weight $w \in \mathbb{R}_{>0}$.
- (uy, ub, ey, eb) is a reward 4-tuple for an exponentially timed action, where every reward belongs to $\mathbb{R}$.
- $(*, *, *, *)$ is a reward 4-tuple for a passive action, where $*$ denotes an unspecified reward.

In the case of an exponentially timed action, the attached rewards uy, ub, ey, eb express the contribution $act\_rew(a, \lambda)$ occurring in four of the six MSL formula schemas of Definition 3.1. More precisely, the universal yield reward uy is related to (ii), the universal bonus reward ub is related to (iii), the existential yield reward ey is related to (v), and the existential bonus reward eb is related to (vi). Hence, a performance measure defined through an MSL formula schema quantified with respect to an activity set A is rendered by inserting the rewards $act\_rew(a, \lambda)$ occurring in the MSL formula schema into the appropriate position of the reward 4-tuple of the exponentially timed actions whose name is in A.

The set $\mathcal{G}$ of process terms is generated by the following syntax:

$$E ::= \sum_{i \in I} \alpha_i.E_i \mid E \parallel_A E \mid B(us, es)$$

where:

- The guarded alternative composition operator expresses a choice among $|I|$ actions, where I is a finite set of indices. The sum expresses the null term $\underline{0}$ if $I = \emptyset$. In general, whenever α_i is the selected action for some $i \in I$, the system performs α_i and then behaves as E_i. The selection of the action is performed according to the following rules. The choice among exponentially timed actions is solved according to the race policy, i.e. the action sampling the least duration wins. The choice among passive actions with the same name is probabilistic. More precisely, each passive action with the same name is given an execution probability proportional to its reactive weight. Finally, the choice among passive actions with different names or among passive actions and exponentially timed actions is nondeterministic.
- The parallel composition operator $\_ \parallel_A \_$, with $A \subseteq Act$, expresses the concurrent execution of two terms. $E_1 \parallel_A E_2$ asynchronously executes the actions of E_1 and E_2 that do not belong to A, and synchronously executes actions of E_1 and E_2 with the same name in A, which becomes the name of the resulting action. A synchronization is possible only between a passive action and an exponentially timed action, with the latter determining the rate of the resulting action, or between two passive actions, which results in a passive action.
- For each process constant B, there exists a behavioral equation of the following form:

$$B(x, y) \triangleq \sum_{i \in I} \alpha_i.E_i$$

where (x, y) expresses a pair of rewards belonging to $\mathbb{R}$.

In the case of a constant invocation like $B(us, es)$, the rewards us, es represent the contribution $lstate\_rew(z)$ occurring in two of the six MSL formula schemas of Definition 3.1. More precisely, the universal local state reward us is related to (i), while the existential local state reward es is related to (iv). Hence, a performance measure defined through an MSL formula schema quantified with respect to a local state set Z is rendered by inserting the rewards $lstate\_rew(z)$ occurring in the MSL formula schema into the appropriate position of the reward pair of the constant invocations expressing the local states in Z.

The reason for restricting the syntax to the guarded alternative composition operator and the guarded definition of constants derives from the role played by the local state rewards. In particular, it is worth noting that a sequential process term describes a local state to which a pair of local state rewards is attached whenever the term is invoked. On the one hand, we want to avoid the definition of a local state described by an ambiguous process term like $B(us, es) + B'(us', es')$, for which the value of the local state rewards would depend on the result of the choice. On the other hand, a process term like $B(x, y) \triangleq \alpha.E \parallel_A B'(us', es')$ would be ambiguous as well, because a constant invocation like $B(us, es)$ is intended to describe a local state rather than the parallel composition of local states. Instead, the process term $B(x, y) \triangleq \alpha.(E \parallel_A E')$ is acceptable, because it models a local state (with its pair of local state rewards) that evolves into the parallel composition of several local states after the execution of α.

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 uy are governed by the activity 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 name is in A that are enabled at that state. By contrast, the existential yield rewards ey 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 name is in A that are enabled at that state. Similarly, in the case of the universal and existential bonus rewards ub and eb, we can argue in accordance with the ACTMC interpretation of the core logic of MSL.

As far as the local state rewards are concerned, in accordance with the ACTMC interpretation of the core logic of MSL, the universal local state rewards us are governed by the local state 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 local state rewards associated with the local states in Z of that state. By contrast, the existential local state rewards es are not summed up. Instead, they are subject to the application of a choice function cf that picks up the reward associated with one of the local states in Z which are part of the state under consideration.

5.2. Semantics for SPA with universal and existential rewards

The semantics for our calculus is given by a labeled multi-transition system whose states are described by triples of the form $\langle z, us, es\rangle$ or by the parallel composition of several such triples. Each triple denotes a local state z described by a sequential process term, a universal local state reward us, and an existential local state reward es. In order to correctly construct such triples starting from the process algebraic specification of a system, we employ a function $init$ that appropriately associates universal and existential local state rewards with process terms in $\mathcal{G}$:

$$\begin{aligned} init\left(\sum_{i\in I}\alpha_i.E_i\right) &= \left\langle \sum_{i\in I}\alpha_i.E_i, 0, 0\right\rangle \\ init(E_1 \parallel_A E_2) &= init(E_1) \parallel_A init(E_2) \\ init(B(us, es)) &= \left\langle \sum_{i\in I}\alpha_i.E_i, us, es\right\rangle \quad \text{if } B(x, y) \triangleq \sum_{i\in I}\alpha_i.E_i \end{aligned}$$

The set $\mathcal{T}$ of process states is then defined as follows:

$$T ::= \left\langle \sum_{i\in I}\alpha_i.E_i, us, es\right\rangle \mid T \parallel_A T$$

Formally, the operational semantics of a process term $E \in \mathcal{G}$ is given by a labeled multi-transition system whose transition relation is the least multiset satisfying the operational rules reported in Table 1, and whose initial state is

Table 1
Operational semantics

$$\langle \sum_{i\in I} \alpha_i.E_i, us, es\rangle \xrightarrow{\alpha_i} \langle E_i, us, es\rangle \qquad \text{if } E_i = \sum_{j\in J} \alpha'_j.E'_j$$

$$\langle \sum_{i\in I} \alpha_i.E_i, us, es\rangle \xrightarrow{\alpha_i} init(E_i) \qquad \text{if } E_i = B(us', es') \vee E_i = E_1 \parallel_A E_2$$

$$\frac{T_1 \xrightarrow{\alpha} T'_1}{T_1 \parallel_A T_2 \xrightarrow{\alpha} T'_1 \parallel_A T_2} \text{ if } name(\alpha) \notin A$$

$$\frac{T_1 \xrightarrow{a,\lambda,(uy,ub,ey,eb)} T'_1 \qquad T_2 \xrightarrow{a,*_w,(*,*,*,*)} T'_2}{T_1 \parallel_A T_2 \xrightarrow{a,\lambda\cdot\frac{w}{W_a(T_2)},(uy\cdot\frac{w}{W_a(T_2)},ub,ey,eb)} T'_1 \parallel_A T'_2} \text{ if } a \in A$$

$$\frac{T_1 \xrightarrow{a,*_{w_1},(*,*,*,*)} T'_1 \qquad T_2 \xrightarrow{a,*_{w_2},(*,*,*,*)} T'_2}{T_1 \parallel_A T_2 \xrightarrow{a,*\frac{w_1}{W_a(T_1)}\cdot\frac{w_2}{W_a(T_2)}\cdot(W_a(T_1)+W_a(T_2)),(*,*,*,*)} T'_1 \parallel_A T'_2} \text{ if } a \in A$$

where:

$$name(<a, \_, (\_, \_, \_, \_)>) = a$$

$$W_a(T) = \sum\!\!\!\!\!\!\{\!\!\{ w \mid \exists T' \in \mathcal{T}.\, T \xrightarrow{a,*_w,(*,*,*,*)} T' \}\!\!\}$$

$init(E) \in \mathcal{T}$. As far as the first two rules for parallel composition are concerned, in addition to them, we also consider the symmetric ones that are obtained by exchanging the roles of T_1 and T_2 in the premises.

We say that E is performance closed if and only if the semantics of E does not contain transitions labeled with passive actions. In this case, the semantics of E gives a well-defined ACTMC on which it is possible to conduct the reward-based performance analysis, as seen in Section 2.2. In the following, we denote by $\mathcal{E}$ the set of the performance closed process terms of $\mathcal{G}$.

5.3. Congruence result

We now show that it is possible to define a performance-measure-sensitive congruence for an 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 [10]. In essence, this equivalence aggregates the transitions labeled with the same name 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 and the universal local state rewards of the departing state are summed up, while the universal bonus rewards are multiplied by the probability of executing the corresponding transitions before being summed up. The existential local state rewards, the existential yield rewards, and the existential bonus rewards are subject to the application of a choice function instead of the addition. By doing so, we are consistent with the ACTMC interpretation summarized through Eq. (2).

Definition 5.1. Let $cf \in CF$. We define the partial function aggregated rate-reward with respect to cf:

$$RR_{cf} : \mathcal{T} \times Act \times \{\exp, *\} \times 2^{\mathcal{T}} \nrightarrow \mathbb{R}_{>0} \times \mathbb{R} \times (\mathbb{R}\cup\{*\})^2 \times \mathbb{R} \times (\mathbb{R}\cup\{*\})^2$$

by letting:

$$\begin{aligned} RR_{cf}(T, a, l, C) \;=\; (&Rate(T, a, l, C),\\ &US(T), UY(T, a, l, C), UB(T, a, l, C),\\ &ES_{cf}(T), EY_{cf}(T, a, l, C), EB_{cf}(T, a, l, C)) \end{aligned}$$

where:

$$Rate(T, a, \exp, C) = \sum\{\!\{ \lambda \mid \exists uy, ub, ey, eb.\, \exists T' \in C.\, T \xrightarrow{a,\lambda,(uy,ub,ey,eb)} T' \}\!\}$$
$$Rate(T, a, *, C) = \sum\{\!\{ w \mid \exists T' \in C.\, T \xrightarrow{a,*_w,(*,*,*,*)} T' \}\!\}$$
$$US(T) = \sum\{\!\{ us \mid us \in u\_lstate\_rew\_set(T) \}\!\}$$
$$UY(T, a, \exp, C) = \sum\{\!\{ uy \mid \exists \lambda, ub, ey, eb.\, \exists T' \in C.\, T \xrightarrow{a,\lambda,(uy,ub,ey,eb)} T' \}\!\}$$
$$UB(T, a, \exp, C) = \sum\{\!\{ \frac{\lambda}{Rate(T, a, \exp, C)} \cdot ub \mid \exists uy, ey, eb.\, \exists T' \in C.\, T \xrightarrow{a,\lambda,(uy,ub,ey,eb)} T' \}\!\}$$
$$ES_{cf}(T) = cf\{\!\{ es \mid es \in e\_lstate\_rew\_set(T) \}\!\}$$
$$EY_{cf}(T, a, \exp, C) = cf\{\!\{ ey \mid \exists \lambda, uy, ub, eb.\, \exists T' \in C.\, T \xrightarrow{a,\lambda,(uy,ub,ey,eb)} T' \}\!\}$$
$$EB_{cf}(T, a, \exp, C) = cf\{\!\{ \tfrac{\lambda}{Rate(T,a,\exp,C)} \cdot eb \mid \exists uy, ub, ey.\, \exists T' \in C.\, T \xrightarrow{a,\lambda,(uy,ub,ey,eb)} T' \}\!\}$$
$$UY(T, a, *, C) = UB(T, a, *, C) = EY_{cf}(T, a, *, C) = EB_{cf}(T, a, *, C) = *$$

with $RR_{cf}(T, a, l, C) = \bot$ whenever the multisets above are empty, and:

$$u\_lstate\_rew\_set(T) = \begin{cases} \{\!\{ us \}\!\} & \text{if } T = \langle \_, us, es \rangle \\ \{\!\{ us \}\!\} \cup u\_lstate\_rew\_set(T') & \text{if } T = \langle \_, us, es \rangle \parallel_A T' \end{cases}$$

$$e\_lstate\_rew\_set(T) = \begin{cases} \{\!\{ es \}\!\} & \text{if } T = \langle \_, us, es \rangle \\ \{\!\{ es \}\!\} \cup u\_lstate\_rew\_set(T') & \text{if } T = \langle \_, us, es \rangle \parallel_A T'. \end{cases}$$ ■

Definition 5.2. Let $cf \in CF$. An equivalence relation $\mathcal{B} \subseteq \mathcal{T} \times \mathcal{T}$ is a reward-based Markovian bisimulation with respect to cf iff, whenever $(T_1, T_2) \in \mathcal{B}$, then for all action names $a \in Act$, levels $l \in \{\exp, *\}$, and equivalence classes $C \in \mathcal{T}/\mathcal{B}$:

$$RR_{cf}(T_1, a, l, C) = RR_{cf}(T_2, a, l, C). \quad ■$$

It is easy to see that the union of all the reward-based Markovian bisimulations with respect to cf is the largest reward-based Markovian bisimulation with respect to cf. Such a union, denoted $\sim^{cf}_{\mathrm{RMB}}$, is called a reward-based Markovian bisimilarity with respect to cf.

It is possible to lift $\sim^{cf}_{\mathrm{RMB}}$ in order to equate process terms rather than process states. In essence, we assume that $\sim^{cf}_{\mathrm{RMB}}$ equates two process terms E and F if and only if the tuples constructed from E and F through the function *init* are reward-based Markovian bisimilar with respect to cf.

Definition 5.3. Let $E_1, E_2 \in \mathcal{G}$ and $cf \in CF$. Then:

$$E_1 \sim^{cf}_{\mathrm{RMB}} E_2 \Leftrightarrow init(E_1) \sim^{cf}_{\mathrm{RMB}} init(E_2). \quad ■$$

Theorem 5.4. *Let $cf \in CF$ be commutative, associative, and distributive with respect to multiplication by non-negative numbers, and let I be a finite set of indices. Then:*

- $\forall i \in I.\, E_i \sim^{cf}_{\mathrm{RMB}} F_i \Rightarrow \forall \{\alpha_1, \ldots, \alpha_{|I|}\}.\, \sum_{i \in I} \alpha_i.E_i \sim^{cf}_{\mathrm{RMB}} \sum_{i \in I} \alpha_i.F_i$.
- $E \sim^{cf}_{\mathrm{RMB}} E' \Rightarrow \forall F \in \mathcal{G}.\, \forall A \subseteq Act.\, E \parallel_A F \sim^{cf}_{\mathrm{RMB}} E' \parallel_A F \wedge F \parallel_A E \sim^{cf}_{\mathrm{RMB}} F \parallel_A E'$.

Proof. First, we observe that the proof for the congruence with respect to the guarded alternative composition operator is a straightforward generalization of that concerning the prefix and the binary alternative composition operator of the corresponding theorem of [10,11]. As far as the rates and the universal yield/bonus rewards are concerned, the proof is the same as that of the corresponding theorem of [10,11]. In the case of the existential yield/bonus 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 yield/bonus rewards when working with addition. Finally, in the case of the universal/existential local state rewards, it is sufficient to observe that such rewards do not affect each other inside the vector of local states. ■

Table 2
Axiomatization of $\sim^{cf}_{\mathrm{RMB}}$

$(\mathcal{A}_1)^{cf}_{\mathrm{RMB}}$	$\sum_{i\in I}\alpha_i.E_i = \sum_{i\in I}\alpha_{\sigma(i)}.E_{\sigma(i)}$ where σ is a permutation of I
$(\mathcal{A}_2)^{cf}_{\mathrm{RMB}}$	$<a,\lambda_1,(uy_1,ub_1,ey_1,eb_1)>.E + <a,\lambda_2,(uy_2,ub_2,ey_2,eb_2)>.E + \sum_{i\in I}\alpha_i.E_i =$ $<a,\lambda_1+\lambda_2,(uy_1+uy_2,\frac{\lambda_1}{\lambda_1+\lambda_2}\cdot ub_1+\frac{\lambda_2}{\lambda_1+\lambda_2}\cdot ub_2,$ $cf(ey_1,ey_2),cf(\frac{\lambda_1}{\lambda_1+\lambda_2}\cdot eb_1,\frac{\lambda_2}{\lambda_1+\lambda_2}\cdot eb_2))>.E+\sum_{i\in I}\alpha_i.E_i$
$(\mathcal{A}_3)^{cf}_{\mathrm{RMB}}$	$<a,*_{w_1},(*,*,*,*)>.E + <a,*_{w_2},(*,*,*,*)>.E + \sum_{i\in I}\alpha_i.E_i =$ $<a,*_{w_1+w_2},(*,*,*,*)>.E+\sum_{i\in I}\alpha_i.E_i$
$(\mathcal{A}_4)^{cf}_{\mathrm{RMB}}$	$B_0(us_0,es_0)\parallel_A B_1(us_1,es_1) = B(us_0+us_1,cf(es_0,es_1))$

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

Theorem 5.5. *Let $P_1, P_2 \in \mathcal{E}$ and $cf \in CF$. If $P_1 \sim^{cf}_{\mathrm{RMB}} P_2$, then the value of the reward-based performance measure defined with MSL is the same for P_1 and P_2.*

Proof. We can argue similarly as done in the proof of Theorem 5.4. ■

5.4. Axiomatization

We now provide a sound and complete axiomatization of $\sim^{cf}_{\mathrm{RMB}}$. This is illustrated by the set $\mathcal{A}^{cf}_{\mathrm{RMB}}$ of axioms in Table 2, the first of which simply subsumes the commutativity and the associativity axioms of the deduction system of [10].

Axioms $(\mathcal{A}_2)^{cf}_{\mathrm{RMB}}$ and $(\mathcal{A}_3)^{cf}_{\mathrm{RMB}}$ express the aggregation of rates and of universal and existential transition rewards according to the definition of the aggregated rate-reward function RR_{cf} (see Definition 5.1). Unlike the deduction system of [10], here a subterm of the form $\sum_{i\in I}\alpha_i.E_i$ occurs on both sides of the two axioms, as we are no longer dealing with a binary alternative composition operator but with a multi-operand guarded alternative composition operator.

The details of axiom $(\mathcal{A}_4)^{cf}_{\mathrm{RMB}}$, which is a reworking of the expansion law, can be found in Table 3. In axiom $(\mathcal{A}_4)^{cf}_{\mathrm{RMB}}$, we use the following notation: $\tilde{\lambda} \in \mathbb{R}_{>0}$ and $\widetilde{uy}, \widetilde{ub}, \widetilde{ey}, \widetilde{eb} \in \mathbb{R}$ for an exponentially timed action, while $\tilde{\lambda}$ is $*_w$ and $\widetilde{uy}, \widetilde{ub}, \widetilde{ey}, \widetilde{eb}$ are $*$ for a passive action. $(\mathcal{A}_4)^{cf}_{\mathrm{RMB}}$ is similar to the corresponding axiom of [10] with some differences concerning the manipulation of the universal and existential rewards. In particular, note that $(\mathcal{A}_4)^{cf}_{\mathrm{RMB}}$ is the unique axiom that equates a vector of local states to a single local state. For this reason, the universal and existential local state rewards are calculated by means of adequate invocations of process constants obeying the function RR_{cf} (see Definition 5.1) and the semantics of the guarded alternative composition operator.

Theorem 5.6. *Let $cf \in CF$ satisfy the same constraints as Theorem 5.4. Then the deduction system $Ded(\mathcal{A}^{cf}_{\mathrm{RMB}})$ is sound and complete for $\sim^{cf}_{\mathrm{RMB}}$ over the set of the non-recursive terms of $\mathcal{G}$.*

Proof. We can argue similarly as done in the proof of Theorem 5.4. As far as the universal/existential local state rewards are concerned, it is worth noting what follows. In the case of axioms $(\mathcal{A}_1)^{cf}_{\mathrm{RMB}}$ to $(\mathcal{A}_3)^{cf}_{\mathrm{RMB}}$, the local state rewards do not depend on the form of the process term, which describes a single local state. In the case of axiom $(\mathcal{A}_4)^{cf}_{\mathrm{RMB}}$, it is sufficient to observe that the constant invocation $B(us_0+us_1, cf(es_0,es_1))$ fulfils the definition of the aggregated rate-reward function RR_{cf}, and that the use of constant invocations in the definition of $B(x,y)$ ensures a correct calculation of the local state rewards in accordance with the semantic rules of the guarded alternative composition operator. ■

Table 3
Details of axiom $(\mathcal{A}_4)^{cf}_{\mathrm{RMB}}$

$$B_0(x,y) \triangleq \sum_{i\in I_0} <a_i, \tilde{\lambda}_i, (\widetilde{uy}_i, \widetilde{ub}_i, \widetilde{ey}_i, \widetilde{eb}_i)>.E_i$$

$$B_1(x,y) \triangleq \sum_{i\in I_1} <a_i, \tilde{\lambda}_i, (\widetilde{uy}_i, \widetilde{ub}_i, \widetilde{ey}_i, \widetilde{eb}_i)>.E_i$$

$$B(x,y) \triangleq \sum_{j\in I_0, a_j\notin A} <a_j, \tilde{\lambda}_j, (\widetilde{uy}_j, \widetilde{ub}_j, \widetilde{ey}_j, \widetilde{eb}_j)>.$$
$$(f(E_j, us_0, es_0) \parallel_A B_1(us_1, es_1)) +$$
$$\sum_{j\in I_1, a_j\notin A} <a_j, \tilde{\lambda}_j, (\widetilde{uy}_j, \widetilde{ub}_j, \widetilde{ey}_j, \widetilde{eb}_j)>.$$
$$(B_0(us_0, es_0) \parallel_A f(E_j, us_1, es_1)) +$$
$$\sum_{k\in K_0} \sum_{h\in P_{1,a_k}} <a_k, \tilde{\lambda}_k \cdot \frac{w_h}{W_{1,a_k}}, (\widetilde{uy}_k \cdot \frac{w_h}{W_{1,a_k}}, \widetilde{ub}_k, \widetilde{ey}_k, \widetilde{eb}_k)>.$$
$$(f(E_k, us_0, es_0) \parallel_A f(E_h, us_1, es_1)) +$$
$$\sum_{k\in K_1} \sum_{h\in P_{0,a_k}} <a_k, \tilde{\lambda}_k \cdot \frac{w_h}{W_{0,a_k}}, (\widetilde{uy}_k \cdot \frac{w_h}{W_{0,a_k}}, \widetilde{ub}_k, \widetilde{ey}_k, \widetilde{eb}_k)>.$$
$$(f(E_h, us_0, es_0) \parallel_A f(E_k, us_1, es_1)) +$$
$$\sum_{k\in P_0} \sum_{h\in P_{1,a_k}} <a_k, *_{\frac{w_k}{W_{0,a_k}} \cdot \frac{w_h}{W_{1,a_k}} \cdot (W_{0,a_k}+W_{1,a_k})}, (*,*,*,*)>.$$
$$(f(E_k, us_0, es_0) \parallel_A f(E_h, us_1, es_1))$$

such that $I_0 \cap I_1 = \emptyset$ and for $j \in \{0,1\}$:

$$\begin{aligned} P_{j,a} &= \{k \in I_j \mid a_k = a \wedge \tilde{\lambda}_k = *_{w_k}\} \\ K_j &= \{k \in I_j \mid a_k \in A \wedge \tilde{\lambda}_k \in \mathbb{R}_{>0} \wedge P_{1-j,a_k} \neq \emptyset\} \\ P_0 &= \{k \in I_0 \mid \exists a \in A.\, k \in P_{0,a} \wedge P_{1,a} \neq \emptyset\} \\ W_{j,a} &= \sum\!\!\!\!\!\!\{\!\{ w_k \mid k \in P_{j,a} \wedge \tilde{\lambda}_k = *_{w_k} \}\!\!\!\!\!\!\} \end{aligned}$$

and for $i \in I_0 \cup I_1$:

$$f(E_i, us, es) = \begin{cases} B_i(us, es) \text{ with } B_i(x,y) \triangleq E_i & \text{if } E_i = \sum_{j\in J} \alpha_j.E'_j \\ B_i(us', es') & \text{if } E_i = B_i(us', es') \\ f(E_1, 0, 0) \parallel_{A'} f(E_2, 0, 0) & \text{if } E_i = E_1 \parallel_{A'} E_2 \end{cases}$$

In order to augment the aggregation power of $\sim^{cf}_{\mathrm{RMB}}$ without losing the congruence property, as shown in [10] 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:

$$<a, \lambda_1, (uy_1, ub_1, 0, 0)>.E + <a, \lambda_2, (uy_2, ub_2, 0, 0)>.E + \sum_{i\in I} \alpha_i.E_i =$$
$$<a, \lambda_1 + \lambda_2, (uy_1 + uy_2 + \lambda_1 \cdot ub_1 + \lambda_2 \cdot ub_2, 0, 0, 0)>.E + \sum_{i\in I} \alpha_i.E_i$$

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 where the choice function is max.

Example 5.7. Consider the constant invocation $B(us, es)$ of the following process term:

$$B(x,y) \triangleq <a, \lambda_1, (0, 0, ey_1, eb_1)>.B(x,y) + <a, \lambda_2, (0, 0, ey_2, eb_2)>.B(x,y)$$

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 that is existentially quantified with respect to $\{a\}$. The instant-of-time value of such a performance measure is given by $ER^{\mathrm{a}}_{\mathrm{r}}(B(us, es), \{a\}, \max) + ER_{\mathrm{i}}(B(us, es), \{a\}, \max) = \max(ey_1, ey_2) + \max(\lambda_1 \cdot eb_1, \lambda_2 \cdot eb_2)$. 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 $\max(1 + 2, 2 + 1) = 3$. ■

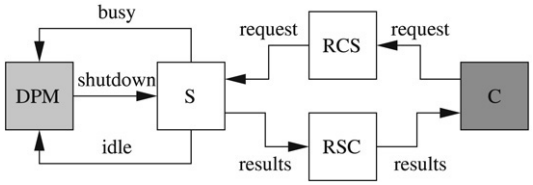

Fig. 2. Power-manageable system.

5.5. Expressiveness

We conclude by observing that the introduction of existential rewards and local state rewards enhances the expressiveness with respect to [10,11].

For instance, let us consider the ACTMC underlying the queuing system example of Section 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 Section 2.1 by inserting rewards into the actions or local states occurring in the specification. We soon realize that the only ways to carry out this task correctly are to associate either a unitary existential local state reward with the behaviors `S1.Server_Busy` and `S2.Server_Busy`, which corresponds to using an MSL formula of type (iv), or a unitary existential yield reward with any `serve` action, which corresponds to using an MSL formula of type (v), as was done in Section 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.

As another example, consider the basic measure *behavior_prob* of Section 4, which determines the probability of being in a certain behavior of an individual component. This measure is rendered in SPA by inserting a universal or existential local state reward equal to 1 into the appropriate position of the constant invocation expressing the behavior of interest. Instead, it cannot be specified by attaching yield or bonus rewards to the actions of that behavior. This is because every state including that behavior must gain a reward equal to 1, even if the state does not enable any action of such a behavior.

6. Case study: Dynamic power management

In this section, we reconsider a case study [1] conducted by some PhD students at the University of Urbino, which emphasizes the fundamental role of an easy-to-use notation to specify performance measures.

These students dealt with the problem of evaluating the introduction of a dynamic power manager within a battery-powered device. The use of a policy that aims at modifying the power consumption of the device on the basis of certain run-time conditions may not be transparent, as it may alter the overall system behavior and efficiency. Thus, the main objective of the case study was to evaluate the impact of the dynamic power management on the system functionalities and performance. The students used the Æmilia specification language and the companion tool TwoTowers [9].

The considered system is depicted in Fig. 2. The client (C) synchronously interacts with the battery-powered server (S) through a full-duplex radio channel implemented by two half-duplex radio channels: RCS, from C to S, and RSC, from S to C. RCS is used by the client to send requests to the server, while RSC is used by the server to send the results back to the client. The server also interacts with the dynamic power manager (DPM), which periodically issues shutdown commands in order to put the server in a low-power inactive state whenever appropriate. Two more signals, idle and busy, are used by the server to notify the DPM about every change of its service state, in order for the DPM to shut down the server only when the latter is idle.

Since the impact of the DPM on the system's efficiency can be measured through the energy consumed by the server, we concentrate on the server component type. Its behavior is characterized through four states: idle (the server is waiting for a request or a shutdown to arrive), busy (the server is processing a request), sleeping (the server has been shut down by the DPM), waking (the server has been woken up by the arrival of a request).

The Æmilia description developed by the students is the following:

```
ARCHI_ELEM_TYPE Server_Type(rate server_proc_rate,
                            rate server_awaking_rate,
                            rate server_notify_rate,
                            rate server_response_rate)
```

```
BEHAVIOR

  Idle_Server(void) =
    choice {
      <receive_request_packet, _> .
        <notify_busy, exp(server_notify_rate)> . Busy_Server(),
      <receive_shutdown, _> . Sleeping_Server()
    };

  Busy_Server(void) =
    <prepare_result_packet, exp(server_proc_rate)> .
      Responding_Server();

  Responding_Server(void) =
    <send_result_packet, exp(server_response_rate)> .
      <notify_idle, exp(server_notify_rate)> . Idle_Server();

  Sleeping_Server(void) =
    <receive_request_packet, _> . Awaking_Server();

  Awaking_Server(void) =
   <awake, exp(server_awaking_rate)> . Busy_Server();

INPUT_INTERACTIONS
  UNI receive_request_packet; receive_shutdown

OUTPUT_INTERACTIONS
  UNI send_result_packet; notify_busy; notify_idle
```

The first equation is associated with the idle state, while the second and the third equations represent the busy state. Two equations are necessary for this state because two activities are carried out — processing the request and sending the results back to the client. The fourth and the fifth equations are concerned with the sleeping and the waking states, respectively. While the processing of a request and the waking represent internal activities, the reception of a request or of a shutdown command are input interactions and the sending of the results is an output interaction. Two more output interactions (whose names start with `notify_`) are used to keep the DPM aware of the state of the server.

As far as the specification of the performance behavior is concerned, the description of `Server_Type` is parameterized with respect to the rate of its durational activities. In fact, every exponentially timed action contains the specification of its duration through `exp(_)`. All the other actions are passive and get a duration only if they communicate with an exponentially timed action.

In order to assess the impact of the DPM from the performance viewpoint, the students evaluated – besides typical metrics like e.g. server utilization – the energy that is consumed by the server for different values of the shutdown period of the DPM. The objective was to get an insight into the trend of the energy consumption. At steady state, the energy consumption is the sum of the probabilities of being in the various server states, each multiplied by a factor that describes the rate at which the server consumes energy in that state.

The students initially followed a classical approach based on reward structures in a way inspired by [11]. Since in this approach the measures are specified by associating yield and bonus rewards with actions, it was not possible to single out the states of interest through the actions occurring in the behavior of `Server_Type`. In order to overcome this drawback, the students realized that it was necessary to modify the Æmilia specification by augmenting each defining equation of `Server_Type` with a self-looping, exponentially timed action exploited to measure the percentage of time that is spent by the server in each of its states. For instance, equation `Idle_Server` became:

```
Idle_Server(void) =
  choice {
    <receive_request_packet, _> .
      <notify_busy, exp(server_notify_rate)> . Busy_Server(),
    <receive_shutdown, _> . Sleeping_Server(),
    <monitor_idle_server> . exp(1)> . Idle_Server()
  };
```

Then, to measure the energy consumption, the students gave a suitable yield reward to every action whose name started with `monitor_`. The value of such a reward was chosen depending on the local state of the server. In particular,

they assumed that the energy consumed in the busy state is 50% more than the energy consumed in the idle and waking states, while of course no energy is consumed in the sleeping state.

Subsequently, the students were exposed to the novel approach based on MSL, which is currently being implemented in TwoTowers. This was quite beneficial for the students, because it turned out that it was no longer necessary to modify the Æmilia specification with monitoring actions in order to define the energy consumption metric. Instead, the students employed the following approach.

Given that the energy consumption depends on the probabilities of being in the various server states, first of all the students defined the basic measure *state_energy_consumption* as follows:

$$\begin{array}{l}\texttt{MEASURE}\ \mathit{state\_energy\_consumption}(C.B(l))\ \texttt{IS}\\ \quad \exists z \in \{C.B\}(\mathit{is\_local}(z, s)) \Rightarrow\\ \qquad \mathit{eq}(\mathit{state\_rew}(s), \mathit{choose\_lstate\_rew}(s, \{C.B\}, \min))\end{array}$$

where $\mathit{lstate\_rew}(C.B) = l$. Note that this measure definition is a generalization of the basic metric *behavior_prob* of Section 4 where the parameter of the metric identifier is equipped with a real number denoting the value of the reward associated with the local state. Then, the students employed the basic metric *state_energy_consumption* to define the following derived metric expressing the overall energy consumption of the server:

$$\begin{array}{l}\texttt{MEASURE}\ \mathit{energy\_consumption}(C.\mathit{Idle}(l_i), C.\mathit{Busy}(l_b), C.\mathit{Responding}(l_r),\\ \qquad\qquad C.\mathit{Sleeping}(l_s), C.\mathit{Awaking}(l_a))\ \texttt{IS}\\ \quad \mathit{state\_energy\_consumption}(C.\mathit{Idle}(l_i)) +\\ \quad \mathit{state\_energy\_consumption}(C.\mathit{Busy}(l_b)) +\\ \quad \mathit{state\_energy\_consumption}(C.\mathit{Responding}(l_r)) +\\ \quad \mathit{state\_energy\_consumption}(C.\mathit{Sleeping}(l_s)) +\\ \quad \mathit{state\_energy\_consumption}(C.\mathit{Awaking}(l_a))\end{array}$$

Finally, based on the value of the rewards chosen by the students, the energy consumption was easily evaluated through the following measure invocation:

$$\begin{array}{l}\mathit{energy\_consumption}(S.\mathit{Idle\_Server}(2), S.\mathit{Busy\_Server}(3), S.\mathit{Responding\_Server}(3),\\ \qquad\qquad S.\mathit{Sleeping\_Server}(0), S.\mathit{Awaking\_Server}(2))\end{array}$$

As the students pointed out, it was quite easy to describe the performance measure of interest in this bottom-up way. In fact, it adheres to the intuition behind the performance measure, and hence results in an arithmetical expression in which the energy consumptions in the individual server states are summed up. This incremental approach is even more beneficial whenever basic metrics like *state_energy_consumption* become part of a library of measure definitions that can be exploited to easily define derived metrics like *energy_consumption*.

7. 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 that affects many system modeling formalisms, we have proposed MSL, a precise and expressive notation for specifying performance measures. MSL is equipped with a measure definition mechanism, through which it is possible to associate mnemonic names with performance metrics derived from reward structures specified through sets of MSL core logic formulas, as well as to parameterize them with respect to component activities and component 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. In this paper, MSL has been exemplified on a number of typical performance measures and its enhanced expressiveness and usability have been illustrated through a realistic case study.

MSL mixes traditional reward structures with a simple first-order logic, which we have shown to support performance-sensitive compositional reasoning in the context of SPA. We believe that MSL fits well with the recent trend of extending model checking tools and performability tools to combine into a single unifying framework logical verification and performance analysis. Among the various proposals, we mention SMART [13], PRISM [24,23], MRMC [22], and Möbius [17].

The SMART tool is based on the Petri nets formalism and offers stochastic timing analysis for both DTMCs and CTMCs. For logical analysis, SMART implements the branching-time logic CTL. For stochastic timing analysis, both numerical solutions and simulation are available. However, SMART does not implement reward-based analysis techniques.

The PRISM tool integrates model checking and performance analysis by extending more expressive logics such as PCTL and CSL with costs/rewards and an expectation operator. The input language is a probabilistic extension of the reactive modules [5], whose underlying stochastic models can be DTMCs, CTMCs, or Markov Decision Processes.

Along the same line, MRMC is a model checker for discrete-time and continuous-time Markov reward models. It supports reward extensions of PCTL and CSL, and allows for the automated verification of properties concerning long-run and instantaneous rewards as well as cumulative rewards. For instance, MRMC allows the modeler to specify non-trivial properties such as the probability to reach one of the goal states within a given number of steps (amount of time) while having earned a certain accumulated reward. MRMC expects as inputs several data structures describing e.g. the probability/rate matrix and the reward structure.

While both PRISM and MRMC privilege the expressiveness of their logic-based measure specification languages, they are neither intended to favor the description of the reward structure for performance non-experts, nor to allow for a component-oriented specification of performance measures.

The Möbius modeling environment offers several modeling formalisms (from process algebra to Petri nets and stochastic automata networks), and a reward model for measure specification, called performance variable, that is based on both rate rewards and instantaneous (called impulse) rewards. The options for evaluating a performance variable include solving for the mean, variance, or distribution of the measure, or for the probability that the measure will fall within a specified range. In order to estimate a reward-based measure, Möbius supports discrete event simulation and state-based numerical techniques. From the usability viewpoint, Möbius uses an explicit component-based system model. A hierarchical approach to modeling is adopted that permits the construction of composed models from previously defined ones. Similarly, reward models build upon atomic and composed models, by equipping them with the specification of a performance measure, which is given as a piece of C++ code. However, Möbius does not include expressive mechanisms such as universal/existential quantification or temporal logic reward formulas.

We conclude by observing that, 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 end, we plan to investigate a way to integrate MSL and CSL in order to further enhance expressiveness while retaining a satisfactory degree of usability.

References

[1] A. Acquaviva, A. Aldini, M. Bernardo, A. Bogliolo, E. Bontà, E. Lattanzi, A methodology based on formal methods for predicting the impact of dynamic power management, in: Formal Methods for Mobile Computing, in: LNCS, vol. 3465, 2005, pp. 155–189.

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

[3] A. Aldini, M. Bernardo, On the usability of process algebra: An architectural view, Theoretical Computer Science 335 (2005) 281–329.

[4] A. Aldini, M. Bernardo, Component-oriented specification of performance measures, in: Proc. of the 4th Int. Workshop on Quantitative Aspects of Programming Languages, QAPL 2006, in: ENTCS, vol. 164, 2006, pp. 27–43.

[5] R. Alur, T.A. Henzinger, Reactive modules, in: Proc. of the 11th Symp. on Logic in Computer Science, LICS 1996, IEEE-CS Press, 1996, pp. 207–218.

[6] C. Baier, J.-P. Katoen, H. Hermanns, Approximate symbolic model checking of continuous time Markov chains, in: Proc. of the 10th Int. Conf. on Concurrency Theory, CONCUR 1999, in: LNCS, vol. 1664, 1999, pp. 146–162.

[7] C. Baier, B. Haverkort, H. Hermanns, J.-P. Katoen, On the logical characterisation of performability properties, in: Proc. of the 27th Int. Coll. on Automata, Languages and Programming, ICALP 2000, in: LNCS, vol. 1853, 2000, pp. 780–792.

[8] S. Balsamo, M. Bernardo, M. Simeoni, Performance evaluation at the software architecture level, in: Formal Methods for Software Architectures, in: LNCS, vol. 2804, 2003, pp. 207–258.

[9] M. Bernardo, TwoTowers 5.1 User Manual, http://www.sti.uniurb.it/bernardo/twotowers/, 2006.

[10] M. Bernardo, 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, in: LNCS, vol. 2165, 2001, pp. 136–151.

[11] M. Bernardo, M. Bravetti, Performance measure sensitive congruences for Markovian process algebras, Theoretical Computer Science 290 (2003) 117–160.

[12] M. Bernardo, L. Donatiello, P. Ciancarini, Stochastic process algebra: From an algebraic formalism to an architectural description language, in: Performance Evaluation of Complex Systems: Techniques and Tools, in: LNCS, vol. 2459, 2002, pp. 236–260.

[13] G. Ciardo, R.L. Jones, A.S. Miner, R. Siminiceanu, SMART: Stochastic model analyzer for reliability and timing, in: Tools of Int. Multiconference on Measurement, Modelling and Evaluation of Computer Communication Systems, 2001, pp. 29–34.
[14] G. Ciardo, J. Muppala, K.S. Trivedi, On the solution of GSPN reward models, Performance Evaluation 12 (1991) 237–253.
[15] G. Clark, Formalising the specification of rewards with PEPA, in: Proc. of the 4th Workshop on Process Algebras and Performance Modelling, PAPM 1996, CLUT, 1996, pp. 139–160.
[16] G. Clark, S. Gilmore, 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, in: LNCS, vol. 1601, 1999, pp. 211–227.
[17] T. Courtney, D. Daly, S. Derisavi, S. Gaonkar, M. Griffith, V. Lam, W.H. Sanders, The Möbius modeling environment: Recent developments, in: Proc. of the 1st Int. Conference on Quantitative Evaluation of Systems, QEST 2004, IEEE-CS Press, 2004, pp. 328–329.
[18] B.R. Haverkort, K.S. Trivedi, Specification techniques for Markov reward models, Discrete Event Dynamic Systems: Theory and Applications 3 (1993) 219–247.
[19] H. Hermanns, Interactive Markov Chains, in: LNCS, vol. 2428, 2002.
[20] J. Hillston, A Compositional Approach to Performance Modelling, Cambridge University Press, 1996.
[21] R.A. Howard, Dynamic Probabilistic Systems, John Wiley & Sons, 1971.
[22] J.-P. Katoen, M. Khattri, I.S. Zapreev, A Markov reward model checker, in: Proc. of the 2nd Int. Conference on Quantitative Evaluation of Systems, QEST 2005, IEEE-CS Press, 2005, pp. 243–244.
[23] M. Kwiatkowska, G. Norman, A. Pacheco, Model checking expected time and expected reward formulae with random time bounds, Computers & Mathematics with Applications 51 (2006) 305–316.
[24] M. Kwiatkowska, G. Norman, D. Parker, PRISM: Probabilistic symbolic model checker, in: Proc. of the 12th Int. Conference on Modelling Techniques and Tools for Computer Performance Evaluation, TOOLS 2002, in: LNCS, vol. 2324, 2002, pp. 200–204.
[25] L. Kleinrock, Queueing Systems, John Wiley & Sons, 1975.
[26] M.A. Qureshi, W.H. Sanders, Reward model solution methods with impulse and rate rewards: An algorithm and numerical results, Performance Evaluation 20 (1994) 413–436.
[27] W.H. Sanders, J.F. Meyer, A unified approach for specifying measures of performance, dependability, and performability, Dependable Computing and Fault Tolerant Systems 4 (1991) 215–237.
[28] W.J. Stewart, Introduction to the Numerical Solution of Markov Chains, Princeton University Press, 1994.
[29] 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, 2000, pp. 511–522.