Weak Behavioral Equivalences for Verifying Secure and Performance-Aware Component-Based Systems

Alessandro Aldini and Marco Bernardo

University of Urbino "Carlo Bo" – Italy
Information Science and Technology Institute

Abstract. Component-based systems are characterized by several orthogonal requirements, ranging from security to quality of service, which may demand for the use of opposite strategies and interfering mechanisms. To achieve a balanced tradeoff among these aspects, we have previously proposed the use of a predictive methodology, which encompasses classical tools such as the noninterference approach to security analysis and standard performance evaluation techniques. The former tool, which is based on equivalence checking, is used to reveal functional dependencies among component behaviors, while the latter tool, which relies on reward-based numerical analysis, is used to study the quantitative impact of these dependencies on the system performance. In order to strengthen the relation between these two different analysis techniques we advocate the use of performance-aware notions of behavioral equivalence as a formal means for detecting functional and performance dependencies and then pinpointing the metrics at the base of a balanced tradeoff.

1 Trading Security with Performance

One of the major issues in the design of modern computing systems is trading dependability aspects with the expected quality of service [16, 10, 15]. A balanced tradeoff is particularly hard to accomplish when the dependability aspect of interest is security and the system under analysis requires the interaction of several, possibly untrusted components performing their activities in wide-area, public networks. As an example, it is commonly recognized that lightweight securing infrastructures like those employed for access control in the setting of the IEEE 802.11 standard for wireless local area networks [26] are able to mitigate the impact of the securing mechanisms on quality of service parameters, such as system throughput and response time, still preserving to a specific extent the properties for which they are introduced.

Examples such as this emphasize the importance of integrating the different qualitative and quantitative views of a system in order to understand whether a reasonable balance can be achieved between the satisfaction of security requirements and the expected quality of service. However, foundational approaches to the analysis of secure and performance-aware systems have not successfully joined with the aim of assessing a balanced qualitative and quantitative profile of these systems. Different aspects of a system behavior are usually dealt with heterogeneous analysis techniques that are applied separately. These techniques consider different descriptions of the software architecture, without a clear comprehension of how to validate mutually such descriptions, how to combine the results obtained through the various analysis techniques and, most importantly, how to evaluate the correlation among such results. On the other hand, an integrated view of these aspects can be at the base of a predictive methodology combining functional verification and quantitative analysis, with the aim of guiding the system design towards the desired tradeoff among security and performance.

For component-based systems, in [4] we have introduced a predictive methodology that can be used in the early stages of the system design to estimate the impact of untrusted components on the system security and performance, thus providing the base for balancing functional and nonfunctional aspects of system behavior. From the modeling standpoint, the methodology relies on formal architectural description languages, which represent a useful aid for the design of effective and efficient software applications. In fact, they provide support for the rigorous specification of systems together with related automated analysis techniques of functional properties and performance measures. From the analysis standpoint, the methodology relies on the application of two phases based on formal tools for the verification of functional interferences among system components and the estimate of the metrics that quantitatively characterize these interferences, respectively. In the first phase, the functional verification is performed through the noninterference approach to information flow analysis [14], which is widely recognized as a valid support to the investigation of several different aspects of security [19]. In the second phase, the quantitative analysis is conducted through standard numerical techniques [23]. For example, this methodology can be used for studying the influence of faults/events triggered by nontrusted components upon the behavior of other components performing security-critical applications [24].

In this paper we extend the methodology of [4] in order to bridge the gap between its two phases, i.e. between the functional noninterference analysis and the nonfunctional performance-oriented analysis. This is accomplished by means of performance-aware notions of behavioral equivalence to be used during noninterference analysis, which make it possible to study both functional and nonfunctional undesired dependencies and, as a consequence, to pinpoint directly the metrics that guide the performance evaluation towards the desired tradeoff. Such an approach can be profitably employed also by those designers who are not familiar with the formal approaches underlying the methodology and are not interested in going into the technicalities of the related ingredients. Moreover, the employed analysis techniques are sufficiently general to represent a valid tool for the study of many dependability aspects – not only security, but also, e.g., safety and reliability – and, therefore, for the assessment of the performability profile of component-based software systems. The revised methodology is

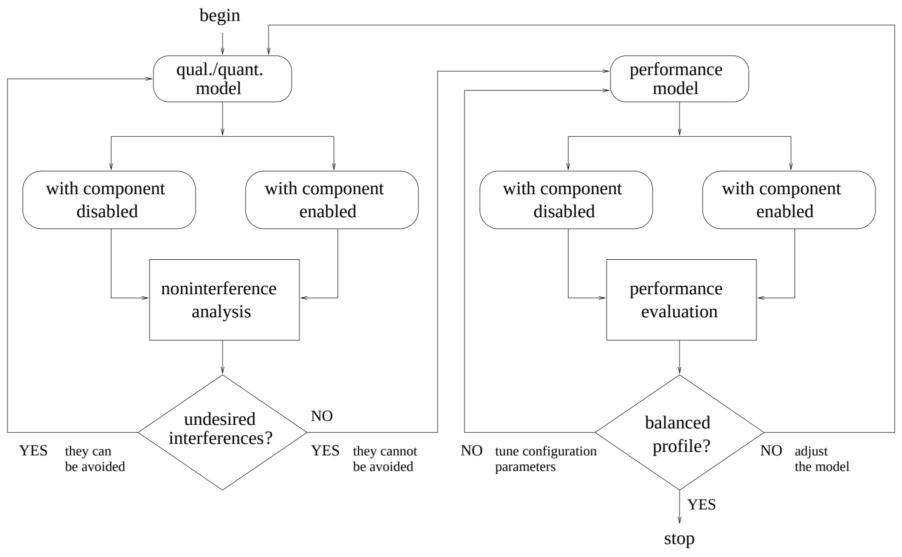

Fig. 1. Phases of the predictive methodology.

illustrated through its application to a running example based on a multilevel security routing system.

The paper, which is an extended version of [3], is organized as follows. In Sect. 2 we illustrate the revised predictive methodology by abstracting from the specific paradigms and companion analysis techniques that can be used to implement it. In Sect. 3 we briefly introduce a simple multilevel security routing system, which is used throughout the paper as a running example. In Sect. 4 we show that the basic ingredients needed by the methodology are supplied by the stochastic process-algebraic architectural description language ÆMILIA [8]. In Sect. 5 we describe the stochastic process algebra and the related behavioral equivalences underlying the application of the quantitative noninterference approach, whose properties are then illustrated in Sect. 6. In Sect. 7 we show how to apply the two phases of the revised predictive methodology to the running example. Finally, in Sect. 8 some conclusions are drawn.

2 Revising the Predictive Methodology

The predictive methodology of [4] aims at integrating in a transparent way orthogonal formal approaches for predicting the existence, estimating the impact, and mitigating the effect of interferences caused by some system components on the behavior of other system components. For this purpose, the methodology employs an integrated system view and combines different techniques for security analysis and performance evaluation. More specifically, the methodology, which is illustrated in Fig. 1, consists of the following two phases:

1. Noninterference analysis, which is carried out to predict the influence of specific components on security properties, so as to establish the absence of undesired, direct and indirect information flows through the system [14, 12]. Essentially, it reduces to verify whether system projections in which certain components are enabled or disabled are equivalent to each other by employing behavioral equivalences.
2. Performance evaluation, which is conducted to estimate the impact of the previously revealed interferences and the effect of the corresponding mitigating strategies on the quality of service. To this aim, standard performance techniques are employed, including, e.g., the numerical solution of Markov chain models [23].

The results returned by each phase should help the designer to pinpoint the causes of system crosscutting anomalies, change the system model, and configure system parameters, depending on the security and performance requirements that should be met. With respect to [4], in this paper we add quantitative modeling and analysis capabilities to the first phase. This extension allows for a more complete investigation of the dependencies among components and a stricter relation with the performance analysis.

In the following, we describe in detail the two phases of the revised methodology as illustrated in Fig. 1.

2.1 Noninterference Analysis

The objective of the first phase of the methodology is to reveal potential interferences among system components that may affect the satisfaction of security requirements. Information flow analysis is a basic approach to the verification of security properties. Among the several conditions that describe the characteristics of unauthorized information flows one of the most interesting, for its intuitive and wide-used idea in security analysis, is the noninterference requirement [14]. Very briefly, in a multilevel secure system a group of high security level users, who perform confidential operations only, does not interfere with a group of low security level users, who observe public operations only, if what the former group of users can do with the confidential operations has no effect on what the latter group of users can see. Noninterference analysis can reveal direct and indirect information flows that violate the security policies based on the access clearances assigned to different user groups.

In order to formalize what a user at a certain security level can see, the activities performed by the system are divided into two disjoint sets: *High*, representing system activities at high security level, and *Low*, representing system activities at low security level. Then, independently of the specific formalization, checking for noninterference actually consists of verifying the indistinguishability of the different low-level views of the system that are obtained by changing the high-level behavior.

Several notions of noninterference have been designed to analyze sequential programs and concurrent systems (see, e.g., [25, 12, 22]). For instance, one of these properties, called strong nondeterministic noninterference and formalized in the CCS process algebraic setting [12], establishes whether the view of the system behavior as observed by a low-level user when the system interacts with high-level users is the same – according to weak bisimulation equivalence $\approx_{\mathrm{B}}$ [20] – as that observed by the low-level user in the absence of high-level users.

Formally, a process term P representing the behavior of a system has no information leakage if the system view where the high-level activities are made unobservable – denoted by P / High – is indistinguishable from the system view where these activities are prevented from execution – denoted by $P \backslash \mathit{High}$:

$$P / \mathit{High} \;\approx_{\mathrm{B}}\; P \backslash \mathit{High}$$

A weak behavioral equivalence is needed because the noninterference comparison requires the ability of abstracting from the high-level activities that a low-level user cannot see directly. In particular, $\approx_{\mathrm{B}}$ is sufficiently expressive to be sensitive to high-level interferences causing, e.g., deadlock or violations of properties that depend on the branching structure of the models. If the two system views to compare do not behave the same, then a low-level user can detect indirectly the behavior of the high-level part of the system by observing what happens at the low level. In other words, an indirect information flow from high level to low level, called covert channel, is set up by exploiting the distinguishing power of the low-level user.

With respect to [4], the first phase of the revised methodology relies on a wide range of fine-grained notions of noninterference including deterministic ones, nondeterministic ones, probabilistic ones, timed ones, or a combination of these, whose choice is left to the designer and depends on how strict the security requirements are. For example, as already mentioned the nondeterministic noninterference check is based on weak bisimulation equivalence $\approx_{\mathrm{B}}$, while the timed noninterference check is defined in terms of weak Markovian bisimulation equivalence, which is illustrated in Sect. 5. Moving to a quantitative framework including fine-grained information augments the distinguishing power of the observer [18]. In general, the more information is added to a system model, the higher the number of vulnerabilities revealed through fine-grained notions of noninterference. In this case, some covert channels that are revealed cannot be completely eliminated without introducing complicated (and perhaps invasive) securing strategies.

In the case that unwanted information flows are captured, diagnostic information – in the form of a modal logic formula returned by the equivalence check – reveals the causes of the interference. If the information flow can be eliminated, then this diagnostic information can be employed by the designer to modify system components. Obviously, such modifications must be validated also from a performance standpoint, in the sense that they should not cause an intolerable degradation of the quality of service. This performance-based validation is mandatory even if the two system views to compare satisfy the strongest property based on the finest information details. For instance, the satisfaction of timed noninterference ensures that no timed covert channel occurs, but does not provide specific information about the delivered quality of service, which may be unsatisfactory because of the strategies adopted to avoid the information leakage.

By contrast, due to their intrinsic nature many covert channels are either unavoidable or tolerated, because they would require impractical revisions of the system. In this case, we have to estimate the impact of these interferences on the system performance.

2.2 Performance Analysis

The objective of the second phase of the methodology is to provide a performance profile of the system. On the one hand, all the unavoidable information flows that have been revealed in the first phase by the noninterference check must be quantitatively analyzed in order to estimate their negative impact on security. For this purpose, the bandwidth of the covert channels detected in the first phase is quantitatively assessed in terms of information leakage per unit of time. On the other hand, even in the case that every covert channel has been eliminated by means of adequate securing strategies, the application of these possibly invasive modifications could be made impractical by hard quality of service constraints.

Therefore, in this phase we trade performance aspects with covert channel bandwidth and with each possible solution proposed to mitigate the information leakage. This is done by observing the performance behavior of the system when disabling and enabling the interfering components.

In this paper we refer to the representation of time passing that uses nonnegative random variables, which is particularly appropriate when the time taken by an event fluctuates according to some probability distribution. Among the many distributions that can be used to model event timing, we concentrate on exponential distributions. The reason is that they yield a simpler mathematical treatment both on the semantic side and on the stochastic side, without sacrificing expressiveness. Whenever all the activity durations are expressed through exponentially distributed random variables, the derived performance model, which has already been used in the first phase to check for timed noninterference, turns out to yield a continuous-time Markov chain, which can be analyzed through standard numerical techniques [23].

With respect to [4], the choice of the performance metrics to analyze in the second phase is facilitated by the feedback provided by the timed noninterference check, which pinpoints the component activities interfering with the observable quantitative behavior of the system. Should the first phase reveal undesired information flows that are unavoidable or whose elimination is impractical, an estimate of the related information leakage is provided in the second phase by evaluating the performance metrics that are directly related to the bandwidth of each information flow. These metrics provide different results for the two system views corresponding to the presence and the absence of the interfering components, respectively, and the difference between such results represents the amount of information leakage. Similarly, quality of service metrics are assessed by analyzing the same system views in order to measure the impact of any residual covert channel on such metrics.

The output of this performance comparison is given by the value of some important efficiency measures of the system together with the bandwidth of its covert channels, expressed as the amount of information leaked per unit of time. Such performance figures can be used in the second phase as a feedback to tune system configuration parameters, in a way that lowers the covert channel bandwidth under a tolerable threshold without jeopardizing the quality of service delivered by the system. In the case that a reasonable tradeoff cannot be obtained, it is necessary to adjust the model and restart the analysis.

In any case, independently of the possibly strict/relaxed security needs and loose/tight quality of service constraints, the outcome resulting from the second phase reveals whether a balanced tradeoff between security – in terms of bandwidth of each covert channel – and performance – in terms of indices like system productivity and response time – is met or not.

3 Running Example: Multilevel Security Routing System

The two phases of the predictive methodology are illustrated through a simple multilevel security routing system. Multilevel security refers to the problem of sharing data with different access clearances in the same system or network. The goal is permitting information to flow freely among users having appropriate security clearances while preventing leaks to unauthorized users.

For the sake of simplicity, we consider only two access clearance levels, high and low, and users playing only two different roles, sender and receiver. The communication between these users is controlled by a router that regulates the exchange of messages among senders and receivers on the basis of their level. We also assume that there is only one high (resp. low) sender and only one high (resp. low) receiver.

4 Component-Oriented System Modeling and Verification

The application of the predictive methodology requires a sufficiently expressive specification language. In this paper, we use the stochastic process-algebraic architectural description language ÆMILIA [8].

As shown in Table 1, a textual architectural description in ÆMILIA starts with its name and formal parameters (initialized with default values), then comprises an architectural behavior section and an architectural topology section.

The first section defines the overall behavior of the system by means of types of software components and connectors, which are collectively called architectural element types. The behavior of an AET has to be provided in the form of a sequence of behavioral equations written in a verbose variant of process algebra allowing only for the inactive process (rendered as `stop`), the action prefix operator supporting possible boolean guards and value passing, the alternative composition operator (rendered as `choice`), and recursion.

Interactions are actions occurring in the process algebraic specification of the behavior of the AET that act as interfaces for the AET itself, while all the other

```
ARCHI_TYPE                     ◁name and initialized formal parameters▷

  ARCHI_BEHAVIOR
         ⋮                        ⋮
    ARCHI_ELEM_TYPE            ◁AET name and formal parameters▷
      BEHAVIOR                 ◁sequence of process algebraic equations built from
                                stop, action prefix, choice, and recursion▷
      INPUT_INTERACTIONS       ◁input synchronous/semi-synchronous/asynchronous
                                uni/and/or-interactions▷
      OUTPUT_INTERACTIONS      ◁output synchronous/semi-synchronous/asynchronous
                                uni/and/or-interactions▷
         ⋮                        ⋮

  ARCHI_TOPOLOGY
    ARCHI_ELEM_INSTANCES       ◁AEI names and actual parameters▷
    ARCHI_INTERACTIONS         ◁architecture-level AEI interactions▷
    ARCHI_ATTACHMENTS          ◁attachments between AEI local interactions▷

END
```

Table 1. Structure of an ÆMILIA textual description

actions are assumed to represent internal activities. Each interaction has to be equipped with three qualifiers, with the first qualifier establishing whether the interaction is an input or output interaction.

The second qualifier represents the synchronicity of the communications in which the interaction can be involved. We distinguish among synchronous interactions which are blocking, semi-synchronous interactions which cause no blocking as they raise an exception if prevented, and asynchronous interactions which are completely decoupled from the other parties involved in the communication.

The third qualifier describes the multiplicity of the communications in which the interaction can be involved. We distinguish among uni-interactions which are mainly involved in one-to-one communications, and-interactions guiding inclusive one-to-many communications, and or-interactions guiding selective one-to-many communications.

The second section of an ÆMILIA description defines the system topology. This is accomplished in three steps. Firstly, we have the declaration of the instances of the AETs – called AEIs – which represent the actual system components and connectors, together with their actual parameters. Secondly, we have the declaration of the architectural (as opposed to local) interactions, which are some of the interactions of the AEIs that act as interfaces for the whole systems. Thirdly, we have the declaration of the architectural attachments among the local interactions of the AEIs, which make the AEIs communicate with each other. An attachment is admissible only if it goes from an output interaction of an AEI to an input interaction of another AEI. Moreover, a uni-interaction can be attached only to one interaction, whereas an and/or-interaction can be attached only to uni-interactions.

Example 1. Let us model the system of Sect. 3 with ÆMILIA. Here is the architectural description header:

```
ARCHI_TYPE ML_Sec_Routing(const rate mlsr_sending_high := 4
                          const rate mlsr_sending_low  := 4
                          const rate mlsr_trans_high   := 5
                          const rate mlsr_trans_low    := 5)
```

The formal data parameters specify a set of rates expressed in sec^{-1} that are concerned with the duration of the system activities. These rates are passed as actual parameters to the instances in the architectural topology section. The average sending time for high and low senders is 250 msec, while the average transmission time from the routing system to each receiver is 200 msec. We use four different parameters because when conducting performance evaluation we will make them vary in different ranges. The system comprises four AETs: the sender, the buffer, the router, and the receiver.

The sender AET, which repeatedly sends messages, is defined as follows:

```
ARCHI_ELEM_TYPE Sender_Type(const rate sending_rate)

 BEHAVIOR
  Sender(void; void) =
   <send, exp(sending_rate)> . Sender()

 INPUT_INTERACTIONS  void
 OUTPUT_INTERACTIONS SYNC UNI send
```

Every action contains the specification of its duration. Exponentially timed actions are of the form `exp(.)`. The duration of each such action is exponentially distributed with parameter equal to the action rate (hence the average duration is the inverse of the rate).

The receiver AET, which is waiting for incoming messages, is defined as follows:

```
ARCHI_ELEM_TYPE Receiver_Type(void)

 BEHAVIOR
  Receiver(void; void) =
   <receive, _(0, 1)> . Receiver()

 INPUT_INTERACTIONS  SYNC UNI receive
 OUTPUT_INTERACTIONS void
```

Passive actions are of the form `_(.,.)`, where the two parameters are the priority constraint and the weight, respectively. Each passive action gets a duration only if it is attached to an exponentially timed or immediate action. Actions that are not passive cannot be attached to each other. A passive action and a non-passive action can be attached to each other if and only if their priority constraint and priority level, respectively, are equal.

The routing system is made of two one-position buffers – one for each level – and a shared router. The buffer AET is defined as follows:

```
ARCHI_ELEM_TYPE Buffer_Type(void)

 BEHAVIOR
  Buffer(void; void) =
   <deposit, _(0, 1)> . <withdraw, _(1, 1)> . Buffer()

 INPUT_INTERACTIONS  SYNC UNI deposit
 OUTPUT_INTERACTIONS SYNC UNI withdraw
```

The router accepts messages arriving from high and low senders and then transmits them to receivers of the corresponding level. The router AET is as follows:

```
ARCHI_ELEM_TYPE Router_Type(const rate trans_rate_high,
                            const rate trans_rate_low)

 BEHAVIOR
  Router(void; void) =
   choice
   {
    <get_high, inf(1, 1)> .
     <trans_high, exp(trans_rate_high)> . Router(),
    <get_low, inf(1, 1)> .
     <trans_low, exp(trans_rate_low)> . Router()
   }

 INPUT_INTERACTIONS  SYNC UNI get_high; get_low
 OUTPUT_INTERACTIONS SYNC UNI trans_high; trans_low
```

Immediate actions are of the form `inf(.,.)`, where the two parameters are the priority level and the weight, respectively. Each immediate action has duration zero and takes precedence over exponentially timed actions, which are assumed to have priority level 0.

Finally, the architectural topology section is as follows:

```
ARCHI_ELEM_INSTANCES
 S_High : Sender_Type(mlsr_sending_high);
 S_Low  : Sender_Type(mlsr_sending_low);
 B_High : Buffer_Type();
 B_Low  : Buffer_Type();
 U      : Router_Type(mlsr_trans_high,
                      mlsr_trans_low);
 R_High : Receiver_Type();
 R_Low  : Receiver_Type()
```

```
ARCHI_INTERACTIONS
 void

ARCHI_ATTACHMENTS
 FROM S_High.send     TO B_High.deposit;
 FROM S_Low.send      TO B_Low.deposit;
 FROM B_High.withdraw TO U.get_high;
 FROM B_Low.withdraw  TO U.get_low;
 FROM U.trans_high    TO R_High.receive;
 FROM U.trans_low     TO R_Low.receive
```

■

ÆMILIA is equipped with a translation semantics into stochastic process algebra as well as analysis techniques that, in the performance evaluation case, require the solution of the underlying stochastic process in the form of a continuous-time Markov chain (CTMC). In order to enable the specification of performance metrics in a component-oriented fashion, ÆMILIA is endowed with a companion notation called Measure Specification Language (MSL) [5]. This notation builds on a simple first-order logic by means of which reward structures [17] are associated with the CTMCs underlying component-oriented system models expressed in ÆMILIA. The notation itself is component oriented because it includes a mechanism for defining measures that are parameterized with respect to component activities and component behaviors. 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, thus facilitating the task for nonexperts.

For instance, the use of the measure expressing system throughput simply requires the designer to specify the component activities contributing to the throughput. In fact the measure is defined in MSL as follows:

`MEASURE throughput` $(C_1.a_1, \ldots, C_n.a_n)$
`IS` $\lhd$ *body* $\rhd$

where *body* is a first-order logic formula specifying how the component activities $C_1.a_1, \ldots, C_n.a_n$ contribute to the reward structure associated with the metric. In particular, the throughput formula establishes that each state transition labeled with an activity in $\{C_1.a_1, \ldots, C_n.a_n\}$ is given a unit reward, which specifies the instantaneous gain implied by the execution of the related transition.

MSL provides support for the incremental definition of performance measures. Basic measures like system throughput can be combined to define derived measures. The body of a derived measure definition is an expression involving identifiers of previously defined metrics each denoting the value of the corresponding measure, as well as arithmetical operators and mathematical functions.

Example 2. As an example, the low-level productivity of the system of Sect. 3 is obtained by evaluating the following MSL definition:

```
MEASURE low_prod(U.trans_low)
     IS throughput(U.trans_low)
```

while the overall system productivity can be specified in MSL as follows:

```
MEASURE total_prod(U.trans_low, U.trans_high)
     IS low_prod(U.trans_low) + high_prod(U.trans_high)
```

where the high-level productivity is defined similarly to the low-level one. 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 Stochastic Process Algebra Framework

In this section we describe the formal framework supporting ÆMILIA. We consider a Markovian process calculus that we call MPC, which includes durational actions and a multiway communication policy based on a mixture of the generative and reactive models of [13]. In the following, we illustrate the calculus together with its bisimulation semantics.

5.1 Markovian Process Calculus

The basic elements of MPC are the actions, which are durational, hence they are represented as pairs of the form $<a, \tilde{\lambda}>$, where a is the action name and $\tilde{\lambda}$ is the action rate. There are three kinds of actions: exponentially timed, immediate, and passive.

Exponentially timed actions are of the form $<a, \lambda>$ with $\lambda \in \mathbb{R}_{>0}$. The average duration of the action is equal to the reciprocal of its rate, i.e. $1/\lambda$. When several exponentially timed actions are enabled, the race policy is adopted: the action that is executed is the fastest one. The sojourn time associated with a process term P is thus the minimum of the random variables quantifying the durations of the exponentially timed actions enabled by P. Since the minimum of several exponentially distributed random variables is exponentially distributed and its rate is the sum of the rates of the original variables, the sojourn time associated with P is exponentially distributed with rate equal to the sum of the rates of the actions enabled by P. Therefore, the average sojourn time associated with P is the reciprocal of the sum of the rates of the actions it enables. The probability of executing one of those actions is given by the action rate divided by the sum of the rates of all the considered actions.

Immediate actions are of the form $<a, \infty_{l,w}>$, where $l \in \mathbb{N}_{>0}$ is the priority level and $w \in \mathbb{R}_{>0}$ is the weight. Each immediate action has duration zero and takes precedence over exponentially timed actions, which are assumed to have priority level 0. When several immediate actions are enabled, the generative preselection policy is adopted. This means that the lower priority immediate actions are discarded, whereas each of the highest priority immediate actions is given an execution probability equal to the action weight divided by the sum of the weights of all the highest priority immediate actions.

Passive actions are of the form $<a, *^{l'}_{w}>$, where $l' \in \mathbb{N}$ is the priority constraint and $w \in \mathbb{R}_{>0}$ is the weight. The duration of a passive action is undefined. When several passive actions are enabled, the reactive preselection policy is adopted. This means that, within every set of enabled passive actions having the same name, each such action is given an execution probability equal to the action weight divided by the sum of the weights of all the actions in the set. Instead, the choice among passive actions having different names is nondeterministic. Likewise, the choice between a passive action and a non-passive action is nondeterministic.

MPC relies on an asymmetric synchronization discipline, according to which a nonpassive action can synchronize only with a passive action having the same name. In other words, the synchronization between two nonpassive actions is forbidden. Following the terminology of [13], the adopted synchronization discipline mixes generative and reactive probabilistic aspects. Firstly, among all the enabled nonpassive actions, the proposal of an action name is generated through a selection based on the rates of those actions. Secondly, the enabled passive actions that have the same name as the proposed one react by means of a selection based on their weights. Thirdly, the nonpassive action winning the generative selection and the passive action winning the reactive selection synchronize with each other. The rate of the synchronization is given by the rate of the selected nonpassive action multiplied by the execution probability of the selected passive action. Multiway synchronizations are allowed provided that they involve at most one nonpassive action, with all the other actions being passive.

Definition 1. *Let $Act = Name \times Rate$ be a set of actions, with $Name$ being a set of action names containing a distinguished symbol τ for the invisible action and $Rate = \mathbb{R}_{>0} \cup \{\infty_{l,w} \mid l \in \mathbb{N}_{>0} \wedge w \in \mathbb{R}_{>0}\} \cup \{*^{l'}_{w} \mid l' \in \mathbb{N} \wedge w \in \mathbb{R}_{>0}\}$ being a set of action rates (ranged over by $\tilde{\lambda}$). The set $\mathcal{L}$ of process terms is generated by the following syntax:*

$$P ::= \underline{0} \mid <a, \tilde{\lambda}>.P \mid P + P \mid P/L \mid P \,\|_S\, P \mid A$$

where $L, S \subseteq Name - \{\tau\}$ and A is a process constant defined through the (possibly recursive) equation $A \overset{\Delta}{=} P$.

The semantics for the set $\mathcal{P}$ of closed and guarded process terms of $\mathcal{L}$ is defined in the usual operational style by taking into account that the alternative composition operator is not idempotent. For instance, process term $<a, \lambda>.\underline{0} + <a, \lambda>.\underline{0}$ is not the same as $<a, \lambda>.\underline{0}$, because the average sojourn time associated with the latter, i.e. $1/\lambda$, is twice the average sojourn time associated with the former, i.e. $1/(\lambda + \lambda)$. In order to assign distinct semantic models to process terms like the two considered above, it is sufficient to keep track of the multiplicity of each transition, intended as the number of different proofs for the transition derivation. The labeled multitransition system for a process term $P \in \mathcal{P}$ si denoted by $[\![P]\!]$.

The null term $\underline{0}$ cannot execute any action, hence the corresponding semantics is given by a state with no transitions. The action prefix term $<a, \tilde{\lambda}>.P$ can execute an action with name a and rate $\tilde{\lambda}$ and then behaves as P:

$$<a, \lambda>.P \xrightarrow{a,\lambda} P \qquad <a, \infty_{l,w}>.P \xrightarrow{a,\infty_{l,w}} P \qquad <a, *_w^{l'}>.P \xrightarrow{a,*_w^{l'}} P$$

The alternative composition $P_1 + P_2$ behaves as either P_1 or P_2 depending on whether P_1 or P_2 executes an action first:

$$\frac{P_1 \xrightarrow{a,\tilde{\lambda}} P'}{P_1 + P_2 \xrightarrow{a,\tilde{\lambda}} P'} \qquad \frac{P_2 \xrightarrow{a,\tilde{\lambda}} P'}{P_1 + P_2 \xrightarrow{a,\tilde{\lambda}} P'}$$

The hiding term P/L behaves as P with the difference that the name of every action executed by P that belongs to L is turned into τ:

$$\frac{P \xrightarrow{a,\tilde{\lambda}} P' \quad a \in L}{P/L \xrightarrow{\tau,\tilde{\lambda}} P'/L} \qquad \frac{P \xrightarrow{a,\tilde{\lambda}} P' \quad a \notin L}{P/L \xrightarrow{a,\tilde{\lambda}} P'/L}$$

The parallel composition $P_1 \parallel_S P_2$ behaves as P_1 in parallel with P_2 as long as actions are executed whose name does not belong to S:

$$\frac{P_1 \xrightarrow{a,\tilde{\lambda}} P_1' \quad a \notin S}{P_1 \parallel_S P_2 \xrightarrow{a,\tilde{\lambda}} P_1' \parallel_S P_2} \qquad \frac{P_2 \xrightarrow{a,\tilde{\lambda}} P_2' \quad a \notin S}{P_1 \parallel_S P_2 \xrightarrow{a,\tilde{\lambda}} P_1 \parallel_S P_2'}$$

Generative-reactive synchronizations are forced between any non-passive action executed by one term and any passive action executed by the other term that have the same name belonging to S and the same priority level/constraint:

$$\frac{P_1 \xrightarrow{a,\lambda} P_1' \quad P_2 \xrightarrow{a,*_w^0} P_2' \quad a \in S}{P_1 \parallel_S P_2 \xrightarrow{a,\lambda\cdot\frac{w}{weight(P_2,a,0)}} P_1' \parallel_S P_2'} \qquad \frac{P_1 \xrightarrow{a,*_w^0} P_1' \quad P_2 \xrightarrow{a,\lambda} P_2' \quad a \in S}{P_1 \parallel_S P_2 \xrightarrow{a,\lambda\cdot\frac{w}{weight(P_1,a,0)}} P_1' \parallel_S P_2'}$$

$$\frac{P_1 \xrightarrow{a,\infty_{l,v}} P_1' \quad P_2 \xrightarrow{a,*_w^l} P_2' \quad a \in S}{P_1 \parallel_S P_2 \xrightarrow{a,\infty_{l,v\cdot\frac{w}{weight(P_2,a,l)}}} P_1' \parallel_S P_2'} \qquad \frac{P_1 \xrightarrow{a,*_w^l} P_1' \quad P_2 \xrightarrow{a,\infty_{l,v}} P_2' \quad a \in S}{P_1 \parallel_S P_2 \xrightarrow{a,\infty_{l,v\cdot\frac{w}{weight(P_1,a,l)}}} P_1' \parallel_S P_2'}$$

where $weight(P, a, l) = \sum\{\!| w \mid \exists P' \in \mathcal{P}.\, P \xrightarrow{a,*_w^l} P' |\!\}$. Reactive-reactive synchronizations are forced between any two passive actions of the two terms that have the same name belonging to S and the same priority constraint:

$$\frac{P_1 \xrightarrow{a,*_{w_1}^l} P_1' \qquad P_2 \xrightarrow{a,*_{w_2}^l} P_2' \qquad a \in S}{P_1 \parallel_S P_2 \xrightarrow{a,*^l_{\frac{w_1}{weight(P_1,a,l)}\cdot\frac{w_2}{weight(P_2,a,l)}\cdot(weight(P_1,a,l)+weight(P_2,a,l))}} P_1' \parallel_S P_2'}$$

The process constant A behaves as the right-hand side process term in its defining equation:

$$\frac{P \xrightarrow{a,\tilde{\lambda}} P' \quad A \stackrel{\Delta}{=} P}{A \xrightarrow{a,\tilde{\lambda}} P'}$$

We use the abbreviation $P \backslash S$ to stand for $P \parallel_S \underline{0}$, which intuitively describes the behavior of a restriction operator. Moreover, if $P \xrightarrow{a_1,\tilde{\lambda}_1} \dots \xrightarrow{a_n,\tilde{\lambda}_n} P'$, with $n \in \mathbb{N}$, then we say that P' is a derivative of P and we denote with $Der(P)$ the set of derivatives of P. Finally, we denote with $\mathcal{P}_{\text{pc}}$ the set of performance closed process terms of $\mathcal{P}$, i.e. those terms with no passive transitions. The stochastic process underlying $P \in \mathcal{P}_{\text{pc}}$ is a CTMC possibly extended with immediate transitions. States having outgoing immediate transitions are called vanishing as the sojourn time in these states is zero. In order to retrieve a pure CTMC stochastically equivalent to an extended CTMC, all the vanishing states can be adequately eliminated.

5.2 Bisimulation Semantics

Markovian bisimulation equivalence relates two process terms whenever they are able to mimic each other's functional and performance behavior stepwise. This notion is based on the process term exit rate, which is the rate at which a process term can execute actions of a certain name that lead to a certain set of terms and is given by the sum of the rates of those actions due to the race policy.

We now recall from [7] an extension of Markovian bisimilarity, whose basic idea is to compare the exit rates of the process terms by taking into account the three kinds of actions. This is accomplished by parameterizing the notion of exit rate with respect to a number in $\mathbb{Z}$ representing the priority level of the action, which is 0 if the action is exponentially timed, l if the action rate is $\infty_{l,w}$, $-l' - 1$ if the action rate is $*_w^{l'}$.

Definition 2. *Let $P \in \mathcal{P}$, $a \in Name$, $l \in \mathbb{Z}$, and $C \subseteq \mathcal{P}$. The exit rate of P when executing actions with name a and priority level l that lead to C is defined through the following non-negative real function:*

$$rate(P, a, l, C) = \begin{cases} \sum\{\!| \lambda \mid \exists P' \in C.\, P \xrightarrow{a,\lambda} P' |\!\} & \textit{if } l = 0 \\ \sum\{\!| w \mid \exists P' \in C.\, P \xrightarrow{a,\infty_{l,w}} P' |\!\} & \textit{if } l > 0 \\ \sum\{\!| w \mid \exists P' \in C.\, P \xrightarrow{a,*_w^{-l-1}} P' |\!\} & \textit{if } l < 0 \end{cases}$$

where each sum is taken to be zero whenever its multiset is empty.

Extended Markovian bisimilarity compares the process term exit rates for all possible action names and priority levels, except for those actions that will always be pre-empted by higher priority actions of the form $<\tau, \infty_{l,w}>$. We denote by $pri_\infty^\tau(P)$ the priority level of the highest priority immediate τ-action enabled by P, and we set $pri_\infty^\tau(P) = 0$ if P does not enable any immediate τ-action. Moreover, given $l \in \mathbb{Z}$, we use $no\text{-}pre(l, P)$ to denote that no action of level l can be pre-empted in P. Formally, this is the case whenever $l \geq pri_\infty^\tau(P)$ or $-l - 1 \geq pri_\infty^\tau(P)$.

Definition 3. *An equivalence relation $\mathcal{B} \subseteq \mathcal{P} \times \mathcal{P}$ is an extended Markovian bisimulation iff, whenever $(P_1, P_2) \in \mathcal{B}$, then for all action names $a \in Name$, equivalence classes $C \in \mathcal{P}/\mathcal{B}$, and priority levels $l \in \mathbb{Z}$ such that $no\text{-}pre(l, P_1)$*

and no-pre(l, P_2)*:*

$$\mathit{rate}(P_1, a, l, C) = \mathit{rate}(P_2, a, l, C)$$

Extended Markovian bisimilarity, denoted by $\sim_{\mathrm{EMB}}$*, is the union of all the extended Markovian bisimulations.*

The notion of exit rate can be weakened by means of a suitable notion of reachability involving internal actions with zero duration $<\tau, \infty_{l,w}>$, which are unobservable. The idea is that, if a given class of process terms is not reached directly after executing an action of a certain name and level, then we have to explore the possibility of reaching that class by performing a finite-length sequence of immediate τ-actions starting from the term reached after executing the considered action. If this is possible, the probability of executing those action sequences has to be taken into account too.

Definition 4. *Let* $P \in \mathcal{P}$ *and* $l \in \mathbb{N}_{>0}$*. We say that* P *is* l*-unobservable iff* $\mathit{pri}^{\tau}_{\infty}(P) = l$ *and* P *does not enable any immediate non-*τ*-action with priority level* $l' \geq l$*, nor any passive action with priority constraint* $l' \geq l$*.*

Definition 5. *Let* $n \in \mathbb{N}_{>0}$ *and* $P_1, P_2, \ldots, P_{n+1} \in \mathcal{P}$*. A path* π *of length* n*:*

$$P_1 \xrightarrow{\tau, \infty_{l_1, w_1}} P_2 \xrightarrow{\tau, \infty_{l_2, w_2}} \ldots \xrightarrow{\tau, \infty_{l_n, w_n}} P_{n+1}$$

is unobservable iff for all $i = 1, \ldots, n$ *process term* P_i *is* l_i*-unobservable. In that case, the probability of executing path* π *is given by:*

$$\mathit{prob}(\pi) \;=\; \prod_{i=1}^{n} \frac{w_i}{\mathit{rate}(P_i, \tau, l_i, \mathcal{P})}$$

Definition 6. *Let* $P \in \mathcal{P}$*,* $a \in \mathit{Name}$*,* $l \in \mathbb{Z}$*, and* $C \subseteq \mathcal{P}$*. The weak exit rate at which* P *executes actions of name* a *and level* l *that lead to* C *is defined through the following non-negative real function:*

$$\mathit{rate}_{\mathrm{w}}(P, a, l, C) = \sum_{P' \in C_{\mathrm{w}}} \mathit{rate}(P, a, l, \{P'\}) \cdot \mathit{prob}_{\mathrm{w}}(P', C)$$

where C_{w} *is the weak backward closure of* C*:*

$$C_{\mathrm{w}} = C \cup \{Q \in \mathcal{P} - C \mid Q \text{ can reach } C \text{ via unobservable paths}\}$$

and $\mathit{prob}_{\mathrm{w}}$ *is a* $\mathbb{R}_{]0,1]}$*-valued function representing the sum of the probabilities of all the unobservable paths from a term in* C_{w} *to* C*:*

$$\mathit{prob}_{\mathrm{w}}(P', C) \;=\; \begin{cases} 1 & \text{if } P' \in C \\ \sum \{\!| \, \mathit{prob}(\pi) \mid \pi \text{ unobs. path from } P' \text{ to } C \, |\!\} & \text{if } P' \in C_{\mathrm{w}} - C \end{cases}$$

When comparing process term weak exit rates, besides taking pre-emption into account, we also have to skip the comparison for classes that contain certain unobservable terms. More precisely, we distinguish among observable, initially unobservable, and fully unobservable terms. An observable process term is a term that enables a visible action that cannot be preempted by any enabled immediate τ-action. An initially unobservable process term is a term in which all the enabled visible actions are preempted by some enabled immediate τ-action, but at least one of the computations starting at this term with one of the higher priority enabled immediate τ-actions reaches an observable process term. A fully unobservable process term is a term in which all the enabled visible actions are preempted by some enabled immediate τ-action, and all the computations starting at this term with one of the higher priority enabled immediate τ-actions are unobservable.

The weak exit rate comparison with respect to observable and fully unobservable classes must obviously be performed. In order to maximize the abstraction power in the presence of quantitative information attached to immediate τ-actions, the comparison should be conducted with respect to the whole set $\mathcal{P}_{\mathrm{fu}}$ of fully unobservable process terms of $\mathcal{P}$. By constrast, the comparison with respect to initially unobservable classes should be skipped, otherwise process terms like the following would not be weakly Markovian bisimilar to each other:

$$<a, \lambda>.<\tau, \infty_{l_1,w_1}>.<b, \mu>.\underline{0}$$
$$<a, \lambda>.<\tau, \infty_{l_2,w_2}>.<b, \mu>.\underline{0}$$
$$<a, \lambda>.<b, \mu>.\underline{0}$$

In fact, the initially unobservable process term $<\tau, \infty_{l_1,w_1}>.<b, \mu>.\underline{0}$ reached by the first one is not weakly Markovian bisimilar to the initially unobservable process term $<\tau, \infty_{l_2,w_2}>.<b, \mu>.\underline{0}$ reached by the second one if $l_1 \neq l_2$ or $w_1 \neq w_2$, with neither of those initially unobservable process terms being reached by the third one.

Definition 7. *An equivalence relation $\mathcal{B} \subseteq \mathcal{P} \times \mathcal{P}$ is a weak extended Markovian bisimulation iff, whenever $(P_1, P_2) \in \mathcal{B}$, then for all action names $a \in Name$ and levels $l \in \mathbb{Z}$ such that no-pre(l, P_1) and no-pre(l, P_2):*

$$rate_{\mathrm{w}}(P_1, a, l, C) = rate_{\mathrm{w}}(P_2, a, l, C) \quad \text{for all observable } C \in \mathcal{P}/\mathcal{B}$$
$$rate_{\mathrm{w}}(P_1, a, l, \mathcal{P}_{\mathrm{fu}}) = rate_{\mathrm{w}}(P_2, a, l, \mathcal{P}_{\mathrm{fu}})$$

Weak extended Markovian bisimilarity, denoted by $\approx_{\mathrm{EMB}}$, is the union of all the weak extended Markovian bisimulations.

As examples of weakly extended Markovian bisimilar process terms we mention:

$$P_1 \equiv <a, \lambda>.<\tau, \infty_{l,w}>.<b, \mu>.\underline{0}$$
$$P_2 \equiv <a, \lambda>.<b, \mu>.\underline{0}$$

and also:

$$P_3 \equiv <a, \lambda>.(<\tau, \infty_{l,w_1}>.<b, \mu>.\underline{0} + <\tau, \infty_{l,w_2}>.<c, \gamma>.\underline{0})$$
$$P_4 \equiv <a, \lambda \cdot \tfrac{w_1}{w_1+w_2}>.<b, \mu>.\underline{0} + <a, \lambda \cdot \tfrac{w_2}{w_1+w_2}>.<c, \gamma>.\underline{0}$$

which is related to vanishing state elimination. We also point out that $\approx_{\mathrm{EMB}}$ can abstract not only from intermediate immediate τ-actions, but also from intermediate unobservable self-loops, consistently with the fact that the probability to escape from them is 1. For instance, P_4 is also weakly extended Markovian bisimilar to:

$$P_3' \equiv <a, \lambda>.A$$
$$A \stackrel{\Delta}{=} <\tau, \infty_{l,w_1}>.<b, \mu>.\underline{0} + <\tau, \infty_{l,w_2}>.<c, \gamma>.\underline{0} + <\tau, \infty_{l,w_3}>.A$$

Moreover, $\approx_{\mathrm{EMB}}$ cannot abstract from initial immediate τ-actions, otherwise compositionality with respect to the alternative composition operator would be broken. Indeed, the following process terms:

$$P_5 \equiv <\tau, \infty_{l,w}>.<a, \lambda>.\underline{0}$$
$$P_6 \equiv <a, \lambda>.\underline{0}$$

are not related by $\approx_{\mathrm{EMB}}$.

The careful classification of states on the basis of their functional and performance observability is a key ingredient thanks to which congruence and axiomatization can be achieved for $\approx_{\mathrm{EMB}}$. In particular, compositionality with respect to parallel composition is preserved by restricting to a well-prioritized subset of the non-divergent process terms of $\mathcal{P}$ [7].

6 Noninterference Properties

In MPC, the classification of security levels surveyed in Sect. 2.1 is realized by assuming that the set $\mathit{Name} - \{\tau\}$ of visible action names includes a set Name_L of low-level names and a set Name_H of high-level names, such that $\mathit{Name}_L \cap \mathit{Name}_H = \emptyset$. All the remaining, unclassified action names are disregarded by hiding them, as they do not represent behaviors of high/low security level users. In the following, we show two noninterference properties relying on this classification.

6.1 Bisimulation-Based Strong Noninterference

The requirement at the base of the lack of any interference from high level to low level can be easily expressed by the strong nondeterministic noninterference property, which informally says that a system is secure if its observable low-level behavior is the same in the presence and in the absence of high-level interactions. The stochastic version of this property is called Bisimulation-based Strong Stochastic Noninterference (*BSSNI*) and is defined as follows.

Definition 8. ***(BSSNI)*** *$P \in \mathcal{P}_{\mathrm{pc}}$ is secure iff $P / \mathit{Name}_H \approx_{\mathrm{EMB}} P \backslash \mathit{Name}_H$.*

The nondeterministic version of this property, termed *BSNNI* [12], is easily obtained by replacing $\approx_{\mathrm{EMB}}$ with $\approx_{\mathrm{B}}$. Similarly, the probabilistic version of *BSSNI*, termed *BSPNI* [6], is defined by replacing $\approx_{\mathrm{EMB}}$ with the weak probabilistic bisimilarity of [11], denoted by $\approx_{\mathrm{PB}}$. Observed that $\approx_{\mathrm{EMB}}$ implies $\approx_{\mathrm{PB}}$, which in turn implies $\approx_{\mathrm{B}}$, from [11] it is immediate to derive the following theorem showing the inclusion relations among the three different fine-grained notion of bisimulation-based strong noninterference.

Theorem 1. ***(Conservative Extension)*** *BSSNI $\subset$ BSPNI $\subset$ BSNNI.*

6.2 Bisimulation-Based Strong Local Noninterference

The strongest property of [12] is the Strong Bisimulation Nondeducibility on Compositions, which corresponds to the Strong Local Noninterference property that was independently defined in [21]. In our framework, we consider such a property under the name Bisimulation-based Strong Stochastic Local Noninterference. The underlying intuition states that the absence of any interference is ensured when the low-level user cannot distinguish which, if any, high-level event has occurred at some point in the past.

Definition 9. ***(BSSLNI)*** *$P \in \mathcal{P}_{\rm pc}$ is secure iff $\forall P' \in Der(P)$ and $\forall P''$ such that $P' \xrightarrow{a,\tilde{\lambda}} P''$, with $a \in Name_H$, $P' \backslash Name_H \approx_{\rm EMB} P'' \backslash Name_H$.*

In practice, this property states that the low-level view of the system is not affected by the execution of high-level actions, because it is always the same, from the viewpoint of the low-level user, just before and immediately after any high-level event. The nondeterministic version, *BSNLNI*, and the probabilistic version, *BSPLNI*, can be derived in the obvious way.

While the conservative extension theorem holds also for the notion of bisimulation-based strong local noninterference, in the following it is more interesting to investigate the nature of the relation between the strong local noninterference notion and the strong noninterference notion. Both in the nondeterministic setting and in the probabilistic setting, the former is stronger than the latter. As an example, consider the following process term:

$$P_7 \equiv <h, \infty_{1,w}>.<h, \infty_{1,w}>.<l, \mu>.\underline{0} + <\tau, \infty_{1,w}>.<l, \mu>.\underline{0}$$

where $h \in Name_H$ and $l \in Name_L$. Then, P_7 satisfies the three bisimulation-based strong noninterference properties, while it does not satisfy the three bisimulation-based strong local noninterference properties. In particular, the interference captured by the latter group of properties occurs whenever the high-level user enables the first high-level action h and then disables the second one.

In general, *BSNLNI* $\subset$ *BSNNI* [12] and *BSPLNI* $\subset$ *BSPNI* [6]. However, this inclusion relation does not hold anymore in the stochastic framework. The reason stems from the fine-grained information associated with the high-level actions. The intuition is that the strong local noninterference notion analyzes the low-level view of the system before and after the execution of a high-level action by taking into account neither the time spent by its execution (if it is exponentially timed) nor its priority (if it is immediate).

For instance, consider the process term:

$$P_8 \equiv <h, \lambda>.<l, \mu>.\underline{0} + <l, \mu>.\underline{0}$$

which satisfies *BSSLNI*, because its low-level view is $<l, \mu>.\underline{0}$ before and after the execution of the high-level action h. However, P_8 is not *BSSNI* secure, because $<l, \mu>.\underline{0}$ and $<\tau, \lambda>.<l, \mu>.\underline{0} + <l, \mu>.\underline{0}$, which represent the two low-level views to compare, are not weakly extended Markovian bisimilar. The motivation is given by the race policy between the two exponentially timed actions. Similarly, consider the process terms:

$$P_9 \equiv <h, \infty_{2,w}>.P_{10} + P_{10}$$
$$P_{10} \equiv <l, \infty_{1,w}>.\underline{0} + <l', \infty_{2,w}>.\underline{0}$$

Then P_9 is clearly *BSSLNI* secure, but not *BSSNI* secure. In particular, while in $P_9 \backslash Name_H$ no pre-emption occurs, in $P_9 / Name_H$ the low-level action l is initially pre-empted by the higher priority τ-action, thus altering the probability of executing the two low-level actions l and l'.

In practice, in the stochastic setting the notions of strong noninterference and strong local noninterference cannot be compared.

Theorem 2. *BSSLNI $\not\subset$ BSSNI and BSSNI $\not\subset$ BSSLNI.*

7 Experimental Study

In this section we apply the predictive methodology to the ÆMILIA description of the multilevel security routing system of Sect. 3. The analysis is conducted by means of the ÆMILIA-based software tool TwoTowers [9].

7.1 Noninterference Analysis of the Running Example

The main security requirement of interest for the considered system is given by the absence of any kind of interference from high users to low users. Hence, according to the two roles played by users at the two different security clearances, we assume that the following classification of local interactions accompanies the ÆMILIA description:

```
HIGH S_High.send; R_High.receive
LOW  S_Low.send; R_Low.receive
```

while the remaining local interactions, which represent internal communications within the routing system, are made unobservable as they do not represent activities under the direct control of high/low users.

Firstly, let us examine functional covert channels and consider the property *BSNNI*. By omitting the rates from the labeled multitransition system underlying the ÆMILIA description `ML_Sec_Routing` we obtain the nondeterministic semantic model depicted in the upper part of Fig. 2. As far as transition labels are concerned, we assume what follows: a_{H} (resp. a_{L}) is the action name resulting from the attachment of `S_High.send` (resp. `S_Low.send`) with `B_High.deposit` (resp. `B_Low.deposit`); the routing activities, modeled by the attachments from `B_High.withdraw` to `U.get_high` and from `B_Low.withdraw` to `U.get_low`, are turned into τ-actions; the action name resulting from the attachment of `U.trans_high` (resp. `U.trans_low`) with `R_High.receive` (resp. `R_Low.receive`) is expressed by the label b_{H} (resp. b_{L}). Fig. 2 also shows in its lower part the two system views to compare according to *BSNNI*. The weak bisimulation relating these two system views is illustrated by graphically representing in the same way the states that belong to the same class. Hence, it is easy to verify that the noninterference check is satisfied and the functional behavior of the system is secure. Intuitively, the availability to serve low messages is never compromised independently of the high behavior.

Secondly, consider a richer noninterference analysis based on probabilistic information and *BSPNI*. The related noninterference check is satisfied as well so that the system turns out to be secure. Intuitively, the unique probabilistic choice observable at the low level, which is between the transmission of a low message from the router to the corresponding receiver and the sending of a new message from the low sender to the corresponding buffer, is not altered by any high activity.

Thirdly, suppose that fine-grained information based on time is important for security requirements. For instance, a motivation for this stronger verification

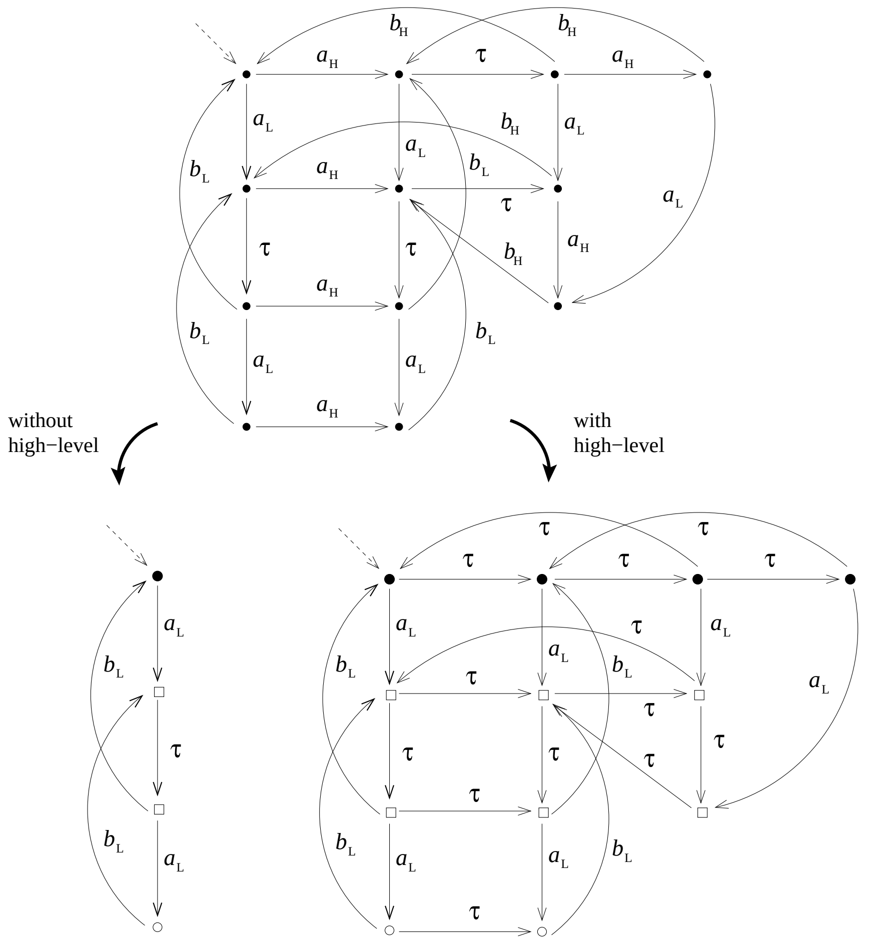

Fig. 2. Labeled multitransition system of the running example and its low-level views to compare according to *BSNNI*.

is to verify whether at the low level we can capture the behavior of the high sender by observing the time needed to receive a low message. The introduction of temporal information causes an information flow, which is revealed by the violation of *BSSNI*. The diagnostic information returned by this check intuitively reveals two interferences.

On the one hand, the presence of `S_High` is detected at the low level by observing the time passage. Indeed, the version of this component with hiding describes a working process that, according to the race policy, competes with the other durational processes, while the version of the same component with restriction does not. On the other hand, from the viewpoint of the low receiver, the time spent by the router to transmit high messages describes an observable busy-waiting phase.

As a commonly used approach in security modeling, the removal of these information flows requires the application of strict control mechanisms that, as expected, degrade the performance of the system in order to make the behavior of the high sender transparent to the low observer.

The first interference that has been captured shows that `S_High` reveals its behavior when executing high durational activities. To avoid this covert channel, it is necessary to confine the behavior of the component in order to hide its impact on the timing of low activities. This can be done by defining a sort of black box that limits and controls the activities performed by the high sender. Formally, `S_High` becomes an instance of the new AET `High_Sender_Type`, which is defined as follows:

```
ARCHI_ELEM_TYPE High_Sender_Type(const rate sending_rate,
                                 const prio h,
                                 const prio k)

 BEHAVIOR
  High_Sender(void; void) =
   <tau, inf(2,1)> .
    choice
    {
     <high_interaction, inf(h, 1)> .
      <send, exp(sending_rate)> . High_Sender(),
     <no_high_interaction, inf(k, 1)> .
      <tau, exp(sending_rate)> . High_Sender()
    }

 INPUT_INTERACTIONS  void
 OUTPUT_INTERACTIONS SYNC UNI send; high_interaction
```

where we assume that $\texttt{h} > \texttt{k} > 2$. The initial τ-action denotes the activation of the black box and is technically needed because it allows $\approx_{\rm EMB}$ to abstract from the subsequent immediate τ-actions. Action `high_interaction` $\in$ *High* denotes the intention by the high sender of sending a message, while action `no_high_interaction` represents the absence of any activity by the high sender. Because of the chosen priorities, the branch guarded by `no_high_interaction`, which is internal and, therefore, unobservable when applying the noninterference check, is enabled iff the high sender is prevented from any interaction with the routing system. The role of this branch is to simulate, from a temporal standpoint, the presence of the high sender in a way that makes its absence invisible to the low observer.

This is not enough to hide completely the interference. Whenever the high sender is blocked because the high buffer is full and hence not willing to accept further messages, then the black box does not compete for the resource time. Indeed, in this case the high sender declares its intention of sending a message and then waits for the transmission of the message. This observable behavior would reveal to the low observer that the high buffer is full. This covert channel can be avoided by introducing the high buffer AET, of which `B_High` becomes an instance:

```
ARCHI_ELEM_TYPE High_Buffer_Type(const rate waiting_rate)

 BEHAVIOR
  High_Buffer(void; void) =
   <deposit, _(0, 1)> .
    choice
    {
     <withdraw, _(1, 1)> . High_Buffer(),
     <tau, exp(waiting_rate)> . High_Buffer()
    }

 INPUT_INTERACTIONS  SYNC UNI deposit
 OUTPUT_INTERACTIONS SYNC UNI withdraw
```

where we assume that the actual rate passed to `B_High` is the same as that passed to `S_High`, because its role is to simulate the durational activities of the high sender whenever it is blocked because of buffer saturation.

The second interference that has been captured shows that the AEI `U` forces a busy-waiting phase for the low receiver whenever transmitting high messages. The router can be made transparent to the low receiver by following an approach borrowed from round-robin scheduling strategies. The intuition is similar to that underlying the black box. The routing activities are divided into temporal slots, each one dedicated to a class of senders in a round-robin fashion. Independently of the presence of a pending message from a sender of the current class, the temporal slot is spent. In this way, a low receiver cannot deduce whether the high slot has been actively exploited. Formally, we replace the AET `Router_Type` with the following round-robin router type, of which `U` becomes an instance:

```
ARCHI_ELEM_TYPE RR_Router_Type(const rate trans_rate_high,
                               const rate trans_rate_low)

 BEHAVIOR
  Low_Round(void; void) =
   choice
   {
    <get_low, inf(1, 1)> .
     <trans_low, exp(trans_rate_low)> . High_Round(),
    <tau, exp(trans_rate_low)> . High_Round()
   };
  High_Round(void; void) =
   choice
   {
    <get_high, inf(1, 1)> .
     <trans_high, exp(trans_rate_high)> . Low_Round(),
    <tau, exp(trans_rate_high)> . Low_Round()
   }
```

```
INPUT_INTERACTIONS  SYNC UNI get_high; get_low
OUTPUT_INTERACTIONS SYNC UNI trans_high; trans_low
```

With these modifications, the system `ML_Sec_Routing` passes the stochastic noninterference check based on *BSSNI*, i.e. the two views of the system that are obtained by enabling and disabling, respectively, the high activities are indistinguishable from the viewpoint of the low observer.

The introduction of information about time makes the satisfaction of noninterference properties a very hard task, which can be accomplished through invasive strategies aiming at controlling the temporal behavior of the system. This claim is strengthened by the fact that we employed one of the least restrictive noninterference properties in the literature. As an example, we ignored the interference caused by a high receiver that blocks the transmission of the high message, because this problem can be easily avoided by making the interaction `trans_high` asynchronous [2].

This and more subtle problems can be revealed by strong notions of security like strong local noninterference. In particular, the revised version of `ML_Sec_Routing` satisfies both *BSNLNI* and *BSPLNI*, provided that the transmission of the high message to the corresponding receiver is made asynchronous. On the other hand, such a relaxation is not enough to make the system *BSSLNI* secure. For this purpose, all the exponentially timed actions executed by the black box must be made invisible in order to keep them out of the control of the high sender. Then, the black box should be further complicated in such a way that the (priority, probabilistic, and temporal) low view of the system must be the same before and after the execution of action `high_interaction`.

7.2 Performance Evaluation of the Running Example

Now we are ready to apply the second phase of the predictive methodology to the ÆMILIA description of the multilevel security routing system. Moving on to the second phase of the methodology is needed to estimate the impact of covert channels and of the related securing countermeasures on the quality of service delivered by the system.

The first analysis we conduct aims at measuring the amount of information leakage for the original version of the running example. The *BSSNI*-based noninterference analysis has shown that some covert channels reveal to the low receiver the high behavior. In particular, the observations of the low receiver are expressed in terms of low productivity of the system. Hence, the related performance metric, described by the MSL measure `low_prod`, offers different results depending on the presence/absence of high interferences. This different quantitative behavior that is exhibited by the two system views to compare must be estimated from a performance standpoint.

Formally, the metric `low_prod` is estimated in the presence and in the absence of high interactions. The results are depicted in Fig. 3(a), where we also report, for the sake of comparison, the number of messages transmitted to the high

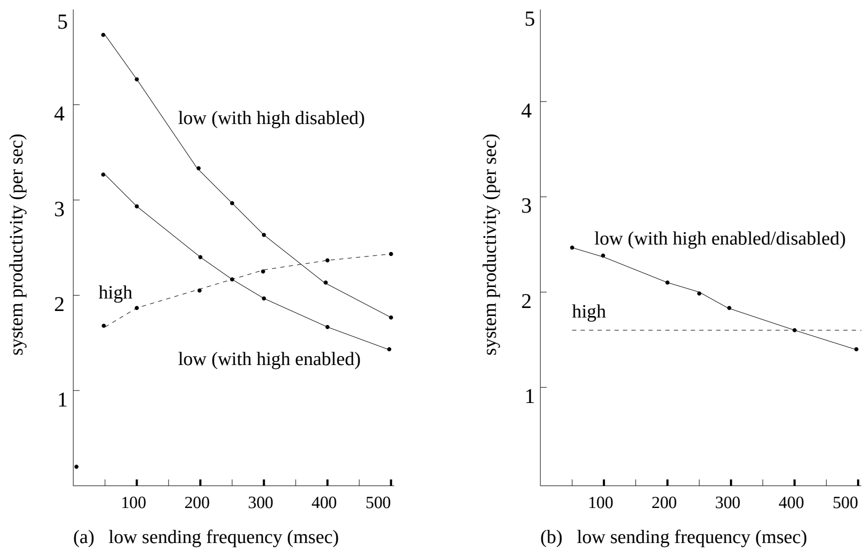

Fig. 3. Performance evaluation of different versions of the multilevel security routing system

receiver whenever the high activities are enabled, which is represented by the metric `high_prod`.

The curves refer to the scenario in which the average sending time for the low sender varies in the range $[50, 500]$ msec. The influence of the undesired information flow is easily estimated by comparing the two thicked curves that are related to the low system productivity in the presence and in the absence of high interferences.

The removal of each covert channel requires strict control mechanisms that degrade the performance. In Fig. 3(b) we estimate the system productivity when activating all the securing strategies described in the previous section. Thanks to these strategies, the two thicked curves of Fig. 3(a) collapse into the same curve, i.e. the low system productivity is independent of the high sender behavior, while the high system productivity becomes constant. This is an expected result as the securing countermeasures make the two system views to compare indistinguishable from the viewpoint of the low receiver. However, it is easy to observe the cost that is paid in terms of decrease of the low system productivity with respect to the scenario of Fig. 3(a). In this respect, it is interesting to compare the low system productivity with that of Fig. 3(a) in the presence of high interferences. The performance degradation experienced by the low receiver when activating the securing mechanisms is remarkable if the low sending frequency is high (about 23% for `mlsr_sending_low` equal to 20, i.e. one request every 50 msec). The degradation is reduced when the low sending frequency decreases and is not perceived anymore for `mlsr_sending_low` equal to 2, i.e. one request every 500 msec.

Hence, depending on the scenario we consider, the securing mechanisms may or may not have a sustainable impact from a performance perspective. Obviously, any intermediate tradeoff can be analyzed by removing some of the securing mechanisms that are needed to make the system completely secure.

8 Conclusions

Strong notions of noninterference are hard to satisfy when modeling real-world systems that are much more complex than the running example surveyed in this paper. Adding fine-grained information such as time opens new scenarios where covert channels cannot be completely eliminated without severely limiting the system behaviors and functionalities. For this reason it becomes important to apply a methodology aiming at trading the minimization of the covert channel bandwidth with the reduction of the quality of service. More concrete examples of the relevance of this methodology as a valid approach to architecting secure and, more in general, dependable systems have been provided in [4, 1].

In general, an approach towards the mitigation of strict constraints imposed by noninterference properties should be based on tolerance thresholds, which are expressed in terms of negligible difference with respect to a family of performance metrics of interest. By following this approach, the predictive methodology can be used to estimate whether unwanted interferences are negligible (e.g. because they are revealed whenever the system execution is observed for a long time) and, therefore, cause a tolerable amount of information leakage in terms of sensitive data that are revealed on the long run.

Another way to improve the flexibility of the methodology consists of employing approximate notions of behavioral equivalences, through which it is possible to estimate the difference between the system views to compare according to the specific noninterference property. The tolerance introduced by the approximation would allow us to relate systems that are similar but do not behave exactly the same.

Acknowledgement
The authors thank the anonymous referees for their valuable comments. This work has been funded by MIUR-PRIN project *PaCo – Performability-Aware Computing: Logics, Models, and Languages*.

References

1. Acquaviva, A., Aldini, A., Bernardo, M., Bogliolo, A., Bontà, E., Lattanzi, E.: A Methodology Based on Formal Methods for Predicting the Impact of Dynamic Power Management. In Formal Methods for Mobile Computing, Springer, LNCS 3465, pp. 155–189 (2005)
2. Aldini, A.: Classification of security properties in a Linda-like process algebra. Journal of Science of Computer Programming 63, pp. 16–38 (2006)

3. Aldini, A., Bernardo, M.: A General Framework for Nondeterministic, Probabilistic, and Stochastic Noninterference. In Joint Workshop on Automated Reasoning for Security Protocol Analysis and Issues in the Theory of Security (ARSPA-WITS 2009), Springer, LNCS 5511, pp. 18–33 (2009)
4. Aldini, A., Bernardo, M.: A Formal Approach to the Integrated Analysis of Security and QoS. Journal of Reliability Engineering & System Safety 92, pp. 1503–1520 (2007)
5. Aldini, A., Bernardo, M.: Mixing Logics and Rewards for the Component-oriented Specification of Performance Measures. Journal of Theoretical Computer Science 382, pp. 3–23 (2007)
6. Aldini, A., Bravetti, M., Gorrieri, R.: A Process-Algebraic Approach for the Analysis of Probabilistic Noninterference. Journal of Computer Security 12, pp. 191–245 (2004)
7. Bernardo, M., Aldini, A.: Weak Markovian Bisimilarity: Abstracting from Prioritized/Weighted Internal Immediate Actions. In 10th Italian Conference on Theoretical Computer Science (ICTCS 2007), World Scientific, pp. 39–56 (2007)
8. Balsamo, S., Bernardo, M., Simeoni, M.: Performance Evaluation at the Software Architecture Level. In Formal Methods for Software Architectures, Springer, LNCS 2804, pp. 207–258 (2003)
9. Bernardo, M.: TwoTowers 5.1 User Manual (2006) `http://www.sti.uniurb.it/bernardo/twotowers/`.
10. Bondavalli, A., Majzik, I., Pataricza, A.: Stochastic Dependability Analysis of System Architecture based on UML Designs. In Architecting Dependable Systems, Springer, LNCS 2677, pp. 219–244 (2003)
11. Bravetti, M., Aldini, A.: Discrete Time Generative-Reactive Probabilistic Processes with Different Advancing Speeds. Theoretical Computer Science 290, pp. 355–406 (2003)
12. Focardi, R., Gorrieri, R.: A Classification of Security Properties. Journal of Computer Security 3, pp. 5–33 (1995)
13. van Glabbeek, R.J., Smolka, S.A., Steffen, B.: Reactive, Generative and Stratified Models of Probabilistic Processes. Information and Computation 121, pp. 59–80 (1995)
14. Goguen, J.A., Meseguer, J.: Security Policy and Security Models. In Symp. on Research in Security and Privacy (SSP 1982), IEEE CS Press, pp. 11–20 (1982)
15. Gupta, V., Lam, V., Ramasamy, H.V., Sanders W.H., Singh S.: Dependability and Performance Evaluation of Intrusion-Tolerant Server Architectures. In Dependable Computing: First LatinAmerican Symposium, Springer, LNCS 2847, pp. 81–101 (2003)
16. Hein, A., Dal Cin, M.: Performance and Dependability Evaluation of Scalable Massively Parallel Computer Systems with Conjoint Simulation. ACM Transactions on Modeling and Computer Simulation 8, pp. 333–373 (1998)
17. R.A. Howard, *"Dynamic Probabilistic Systems"*, John Wiley & Sons (1971)
18. McLean, J.: Security Models and Information Flow. In Symp. on Research in Security and Privacy (SSP 1990), IEEE CS Press, pp. 180–187 (1990)
19. Meadows, C.: What Makes a Cryptographic Protocol Secure? The Evolution of Requirements Specification in Formal Cryptographic Protocol Analysis. In 12th European Symp. on Programming Languages and Systems (ESOP 2003), Springer, LNCS 2618, pp. 10–21 (2003)
20. Milner, R.: Communication and Concurrency. Prentice Hall (1989)
21. Roscoe, A.W., Reed, G.M., Forster, R.: The Successes and Failures of Behavioural Models. In Millenial Perspectives in Computer Science (2000)

22. Ryan, P.Y.A., Schneider, S.A.: Process Algebra and Non-interference. Journal of Computer Security 9, pp. 75–103 (2001)
23. Stewart, W.J.: Introduction to the Numerical Solution of Markov Chains. Princeton University Press (1994)
24. Stravridou, V., Dutertre, B.: From Security to Safety and Back. In Workshop on Computer Security, Dependability, and Assurance: From Needs to Solutions (CSDA 1998), IEEE CS Press, pp. 182-195 (1998)
25. Wittbold, J.T., Johnson, D.M.: Information Flow in Nondeterministic Systems. In Symp. on Research in Security and Privacy (SSP 1990), IEEE CS Press, pp. 144–161 (1990)
26. Wu, F., Johnson, H., Nilsson, A.: SOLA: Lightweight Security for Access Control in IEEE 802.11. IT Professional 6, IEEE CS Press, pp. 10–16 (2004)