The ORIS Tool: Quantitative Evaluation of Non-Markovian Systems

Marco Paolieri, Marco Biagi, Laura Carnevali, and Enrico Vicario (IEEE Member)

Abstract—We present the next generation of ORIS, a toolbox for quantitative evaluation of concurrent models with non-Markovian timers. The tool shifts its focus from timed models to stochastic ones, it includes a new graphical user interface, new analysis methods and a Java Application Programming Interface (API). Models can be specified as Stochastic Time Petri Nets (STPNs) through the graphical editor, validated using an interactive token game, and analyzed through several techniques to compute instantaneous or cumulative rewards. STPNs can also be exported as Java code to conduct extensive parametric studies through the Java library, now distributed as open-source. A well-engineered software architecture allows the user to implement new features for STPNs, new modeling formalisms, and new analysis methods. The most distinctive features of ORIS include transient and steady-state analysis of STPNs modeling Markov Regenerative Processes (MRPs), and transient analysis of STPNs modeling generalized semi-Markov processes. ORIS also supports state-space analysis of Time Petri Nets (TPNs), simulation of STPNs, and standard analysis techniques for continuous-time Markov chains or MRPs with at most one non-exponential timer in each state. We illustrate the general workflow for the application of ORIS to the modeling and evaluation of non-functional requirements of software-intensive systems.

Index Terms—Quantitative Evaluation, Formal Methods, Stochastic Models, Concurrency, Stochastic Petri Nets, Non-Markovian Processes, Markov Regenerative Processes, Performance, Reliability, Software Tools and Libraries.

1 Introduction

In the engineering of non-functional requirements of a large class of software or cyber-physical systems, quantitative evaluation of stochastic models enables early assessment of design choices and provides model-driven guidance for implementation and integration stages. In the specific context of software, notable examples include testing of real-time software components [24], [71], evaluation of quality of service in distributed systems [48], and analysis of restart mechanisms or rejuvenation policies [67], [58], [37].

Most relevant results in the literature on quantitative evaluation have addressed models with exponentially distributed (EXP) durations, which always satisfy the Markov property (i.e., future evolution depends on the current logical location but not on past history). In this case, the underlying stochastic process of the model is a Continuous-Time Markov Chain (CTMC), and evaluation can resort to consolidated numerical approaches and tools such as PRISM [51], MRMC [47], SMART [29], GreatSPN [3], Storm [31], or specialized functions within MATLAB and Mathematica.

However, in several application contexts, the system under analysis is strongly characterized by deterministic (DET) timers (e.g., inter-release times of periodic jobs, timeouts or watchdogs) or non-exponential generally distributed (GEN) durations (e.g. execution times, jittering latencies, asynchronous arrivals, aging). In this case, the underlying stochastic process of the model falls in more complex classes of so-called non-Markovian processes [28], but numerical solution may still be viable if the model satisfies the Markov property at specific times, termed *regeneration points*. In particular, if a new regeneration is always reached with probability 1 (w.p.1), then the model subtends a Markov Regenerative Process (MRP) [50], which can be solved numerically once it has been described in terms of a *local kernel*, characterizing the behavior until the first regeneration, and a *global kernel*, characterizing sequencing and timing of visits to subsequent regenerations. Solutions for evaluation of kernels have been consolidated only for models satisfying the *enabling restriction*, requiring that at most one timer with general distribution is enabled in each state [40], [63], [27], [39], [5]. Software tools implementing these techniques include SHARPE [65], TimeNET [72], and GreatSPN [3].

In this paper, we present the next generation of ORIS [54], which supports construction of Petri net models with concurrent non-Markovian durations and quantitative evaluation of their underlying stochastic processes, through a new Graphical User Interface (GUI) and an Application Programming Interface (API) based on the included Java library SIRIO [61].[1] Functional capabilities of the ORIS tool are summarized in the use case diagram of Fig. 1. On the one hand, the GUI supports the specification of a model using the formalism of Stochastic Time Petri Nets (STPNs), its validation through interactive simulation (token game), and its quantitative evaluation, targeted to transient or steady state rewards, and performed through different solution engines applicable

- *M. Paolieri is with the University of Southern California, Department of Computer Science, 941 Bloom Walk, Los Angeles, CA 90089, USA. E-mail:* paolieri@usc.edu
- *M. Biagi, L. Carnevali and E. Vicario are with the University of Florence, Department of Information Engineering, Via di Santa Marta 3, 50139 Firenze, Italy. E-mail:* {marco.biagi, laura.carnevali, enrico.vicario}@unifi.it.

The authors are grateful to Dr. Jacopo Torrini, who developed the graphical Petri net editor and application framework of ORIS.

1. A tutorial on ORIS 2.0 was presented at VALUETOOLS'17 [12]. A very preliminary version of the API was presented at SAFECOMP'11 [22] and QEST'11 [23]. Release 1.0 of ORIS focused only on nondeterministic analysis of timed models and is now outdated [16].

under different assumptions on the underlying stochastic process. On the other hand, the API provides access to all the functions exposed by the GUI with various additional capabilities for the generalization of modeling features, analysis algorithms, simulation, symbolic manipulation of multivariate probability distributions.

The distinguishing feature of ORIS is the quantitative analysis of models where multiple timers with general distribution can be concurrently enabled in each state, which goes beyond the enabling restriction and significantly extends the class of models amenable to numerical solution. In turn, this feature relies on an implementation of the *method of stochastic state classes* [69], [44], which is put to work in various types of quantitative continuous-time analysis: exact transient and steady-state analysis of STPNs where a regeneration is always reached within a bounded number of state transitions (a subclass of MRPs); exact transient analysis of STPNs where the number of state transitions within the time limit is bounded (without restrictions on the occurrence of regenerations); approximate transient analysis of STPNs with no restriction on the presence or the sequencing of regenerations. In addition, ORIS provides a basic implementation of methods for transient and steady-state analysis of models with underlying CTMC [62] and for transient analysis of models with underlying MRP under the enabling restriction [39], which are the focus of other tools such as PRISM, SHARPE, TimeNET, and GreatSPN. ORIS also supports simulation of STPNs to compute transient and steady-state rewards, and nondeterministic analysis of Time Petri Nets (TPNs) [68] based on the so-called *state class graph* abstraction, also implemented in tools for qualitative verification, notably Tina [9], Romeo [36], and Uppaal [6].

ORIS has been developed for several years and put to the test of practice in various application domains, addressing different modeling and evaluation challenges [55], [24], [14], [25], [13], [20], [11]. Overall, ORIS is now a well-engineered software tool, fully implemented in Java, designed so as to accommodate agile implementation of new features for Petri net models and new solution techniques for their underlying stochastic processes. SIRIO, the Java library included in ORIS, can be used to integrate the analysis functionalities available in the GUI into custom software tools and toolchains, or to perform extensive *parametric studies*, where a system property (such as reliability or availability) is evaluated for many combinations of parameters and model variants defined using the Java API. To simplify this integration, STPN models can be exported from the ORIS GUI as Java code.

In this paper, we illustrate modeling and analysis features of ORIS, and we report on examples of its application. Specifically, in Sections 2 and 3, we illustrate how to construct STPN models and perform quantitative analysis of their underlying stochastic processes, respectively, using a running example from the literature on software rejuvenation to describe the typical modeling and evaluation workflow in ORIS (including derivation of distributions of GEN durations), the class of properties that can be analyzed, and the limits of the analysis methods. In Section 4, we review selected case studies where ORIS has been applied to the quantitative evaluation of models of real-time software [24], [55], railway signalling systems [13], repair procedures for gas and water distribution networks [14], [25], and activity recognition [20], [11]. Finally, conclusions are drawn in Section 5.

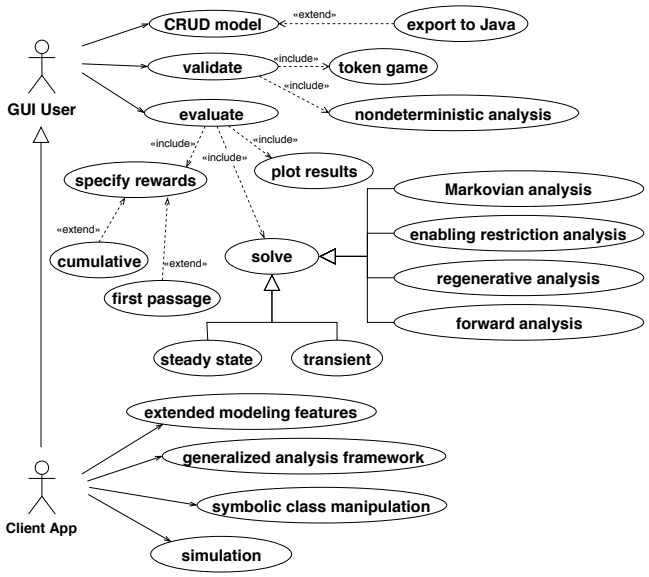

Figure 1: Use-case diagram of the functionalities provided by the ORIS GUI and API.

2 The ORIS Workflow: Model Specification

In ORIS, concurrent systems with stochastic temporal parameters are specified using the formalism of STPNs [69]. In the following, we define syntax and semantics of STPNs (Section 2.1), and we illustrate the typical workflow in the construction of a model: the set of feasible behaviors is initially identified by capturing the structure of concurrency and qualitative timing constraints (Section 2.2); and it is then associated with a quantitative measure of probability by setting stochastic parameters (Section 2.3); the model is finally validated through interactive simulation featured as a token game (Section 2.4).

2.1 Stochastic Time Petri Nets (STPNs)

STPNs are a formal model of concurrent timed systems where *transitions* (depicted as vertical bars, see for instance Fig. 2) represent activities, *places* (depicted as circles) represent discrete components of the logical state with values encoded by a number of *tokens* (depicted as dots), and *directed arcs* from *input* places to transitions and from transitions to *output* places represent token moves occurring at the execution of activities: a transition is enabled when all its input places contain at least one token; and its firing will remove a token from each input place and add one to each output place. The time elapsing from the enabling to the firing of a transition is a random variable (possibly imposing minimum and maximum duration). Besides, the choice between transitions with equal time to fire is solved by a random switch determined by probabilistic weights.

Similarly to stochastic reward nets [64] and stochastic activity networks [59], STPNs also permit the enabling of a transition to be restricted by general constraints on token counts, called *enabling functions*. Moreover, additional updates of token counts after the firing of a transition can

be specified through *update functions*, the restart of selected transitions can be forced using *reset sets*, and *priorities* can be imposed among immediate or deterministic transitions.

Definition 1 (Syntax)**.** An STPN is a tuple $\langle P, T, A^-, A^+, B, U, R, EFT, LFT, F, W, Z\rangle$ where: P and T are disjoint sets of places and transitions, respectively; $A^- \subseteq P \times T$ and $A^+ \subseteq T \times P$ are precondition and post-condition relations, respectively; B, U, and R associate each transition $t \in T$ with an enabling function $B(t)\colon \mathcal{M} \to \{\text{TRUE}, \text{FALSE}\}$, an update function $U(t)\colon \mathcal{M} \to \mathcal{M}$, and a reset set $R(t) \subseteq T$, respectively, where $\mathcal{M}$ is the set of reachable markings $m\colon P \to \mathbb{N}$; EFT and LFT associate each transition $t \in T$ with an earliest firing time $EFT(t) \in \mathbb{Q}_{\geqslant 0}$ and a latest firing time $LFT(t) \in \mathbb{Q}_{\geqslant 0} \cup \{\infty\}$ such that $EFT(t) \leq LFT(t)$; F, W, and Z associate each transition $t \in T$ with a Cumulative Distribution Function (CDF) F_t for its duration $\tau(t) \in [EFT(t), LFT(t)]$ (i.e., $F_t(x) = P\{\tau(t) \leq x\}$, with $F_t(x) = 0$ for $x < EFT(t)$, $F_t(x) = 1$ for $x > LFT(t)$), a weight $W(t) \in \mathbb{R}_{>0}$, and a priority $Z(t) \in \mathbb{N}$, respectively.

A place p is said to be an *input* or *output* place for a transition t if $(p, t) \in A^-$ or $(t, p) \in A^+$, respectively. Following the usual terminology of stochastic Petri nets, a transition t is termed *immediate* (IMM) if $EFT(t) = LFT(t) = 0$ and *timed* otherwise; a timed transition is termed *exponential* (EXP) if $F_t(x) = 1 - \exp(-\lambda x)$ for some rate $\lambda \in \mathbb{R}_{>0}$, or *general* (GEN) if its time to fire has a non-exponential distribution; as a special case, a GEN transition t is *deterministic* (DET) if $EFT(t) = LFT(t) > 0$. For each transition t with $EFT(t) < LFT(t)$, we assume that F_t can be expressed as the integral function of a probability density function (PDF) f_t, i.e., $F_t(x) = \int_0^x f_t(y)\, dy$. The same notation is also adopted for an IMM or DET transition $t \in T$, which is associated with a Dirac impulse function $f_t(y) = \delta(y - \overline{y})$ with $\overline{y} = EFT(t) = LFT(t)$.

A marking $m \in \mathcal{M}$ assigns a natural number of tokens to each place of an STPN. A transition t is *enabled* by m if m assigns at least one token to each of its input places and the enabling function $B(t)(m)$ evaluates to TRUE. The set of transitions enabled by m is denoted as $E(m)$.

Definition 2 (State)**.** The state of an STPN is a pair $\langle m, \vec{\tau}\rangle$ where $m \in \mathcal{M}$ is a marking and vector $\vec{\tau} \in \mathbb{R}_{\geqslant 0}^{|E(m)|}$ assigns a *time to fire* $\vec{\tau}(t) \in \mathbb{R}_{\geqslant 0}$ to each enabled transition $t \in E(m)$.

Definition 3 (Semantics)**.** Given an initial marking m_0, an *execution* of the STPN is a (finite or infinite) path $\omega = s_0 \xrightarrow{\gamma_1} s_1 \xrightarrow{\gamma_2} s_2 \xrightarrow{\gamma_3} \cdots$ such that: $s_0 = \langle m_0, \vec{\tau}_0\rangle$ is the initial state, where the time to fire $\vec{\tau}_0(t)$ of each enabled transition $t \in E(m_0)$ is sampled according to the distribution F_t; $\gamma_i \in T$ is the ith fired transition; $s_i = \langle m_i, \vec{\tau}_i\rangle$ is the state reached after the firing of γ_i. In each state s_i:

- The next transition γ_{i+1} is selected from the set of enabled transitions with minimum time to fire and maximum priority according to a discrete distribution given by weights: if $E_{\text{min}} = \arg\min_{t \in E(m_i)} \vec{\tau}_i(t)$ and $E_{\text{prio}} = \arg\max_{t \in E_{\text{min}}} Z(t)$, then $t \in E_{\text{prio}}$ is selected with probability $p_t = W(t)/(\sum_{u \in E_{\text{prio}}} W(u))$.
- After the firing of γ_{i+1}, the new marking m_{i+1} is derived by (1) removing a token from each input place of γ_{i+1}, (2) adding a token to each output place of γ_{i+1}, and (3) applying the update function $U(\gamma_{i+1})$ to the resulting marking. A transition t enabled by m_{i+1} is termed *persistent* if it is distinct from γ_{i+1}, it is not contained in $R(\gamma_{i+1})$, and it is enabled also by m_i and by the intermediate markings after steps (1) and (2); otherwise, t is termed *newly enabled* (thus, transitions in the reset set of γ_{i+1} are newly enabled if enabled after the firing).
- For each newly enabled transition t, the time to fire $\vec{\tau}_{i+1}(t)$ is sampled according to the distribution F_t; for each persistent transition t, the time to fire in s_{i+1} is reduced by the sojourn time in the previous marking, i.e., $\vec{\tau}_{i+1}(t) = \vec{\tau}_i(t) - \vec{\tau}_i(\gamma_{i+1})$.

When features are omitted for a transition $t \in T$, default values are assumed as suggested by intuition: an always-true enabling function $B(t)(m) = \text{TRUE}$ and an identity update function $U(t)(m) = m$ for all $m \in \mathcal{M}$; an empty reset set $R(t) = \varnothing$; a weight $W(t) = 1$; and, a priority $Z(t) = 0$.

Arc cardinalities greater than 1 could be easily introduced in STPN syntax and semantics, letting the firing of a transition remove an arbitrary number of tokens from each input place or add an arbitrary number of tokens to each output place. Though explicitly supported by the SIRIO library, arc cardinalities were not introduced in the ORIS GUI to reduce model clutter, in contrast with explicit features provided by other tools [72]; instead, arc cardinalities can be modeled in ORIS through enabling and update functions.

2.2 Defining the structure of concurrency of the model

The process for the design of an STPN is conveniently decomposed in activities focusing on two different aspects of the model: the set of feasible timed behaviors is first identified by designing the structure of concurrency, possibly involving qualitative min-max timing constraints; a (quantitative) measure of probability is then introduced by setting stochastic parameters.

We illustrate the concept with reference to an example related to the practice of *software rejuvenation* where a software system subject to *aging* is restarted to prevent failures and avoid extended unavailability intervals [67], [58], [37], [46]. Inspired by [38], we consider a system that accumulates internal errors over time, which are not observed and do not impair correct operation but may eventually lead to a disruptive failure. To prevent failures, the system is periodically rejuvenated by switching it off and restoring it to the initial safe state. If a failure occurs, upon detection, an unplanned repair is performed and the next rejuvenation time is rescheduled.

The basic structure of concurrency is captured by the underlying Petri Net (PN) of the model (i.e., by places, transitions, enabling functions, update functions, and priorities), using places and transitions to represent logical states and state transitions, respectively. In the case of software rejuvenation, the model (shown in Fig. 2) captures the concurrency between the aging process of the software system and the rejuvenation mechanism. In the top part, places `Up`, `Down`, and `Detected` represent the aging of the software through the states of correct functionality, undetected down state, and detected failure state, respectively, while transitions `fail`, `detect`, and `repair` account for the time

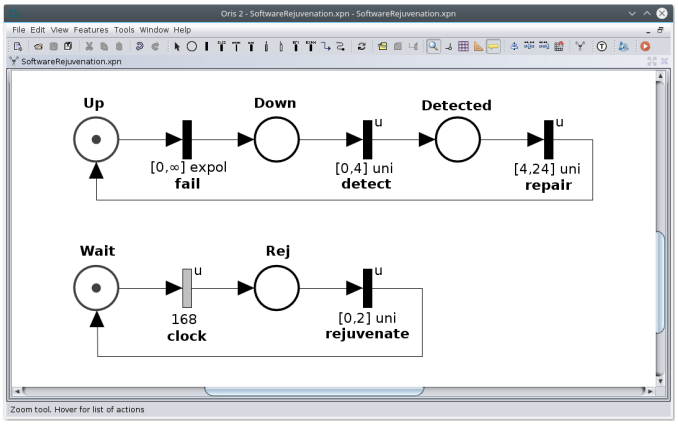

Figure 2: GUI editor of ORIS showing an STPN model of software rejuvenation. As usual in stochastic Petri nets, the color and thickness of a transition represent the PDF type of its stochastic duration: thick bars filled in with white, gray, and black represent EXP, DET, and GEN transitions, respectively; thin black bars represent IMM transitions. Transitions associated with a non-default value of enabling function, update function, or reset set are marked with label `e`, `u`, or `r`, respectively. The update functions are: `Wait` $= 1$ for `repair`; `Wait` $= 0$ for `detect`; `Up` $= 1$ for `rejuvenate`; `Up` $= 0$; `Down` $= 0$; `Detected` $= 0$ for `clock`. Time is in h.

required by software aging, failure detection, and unplanned repair, respectively. In the bottom part, places `Wait` and `Rej` model a rejuvenation scheduled and in progress, respectively, while transitions `clock` and `rejuvenate` represent the time between two consecutive rejuvenations and the time needed to perform rejuvenation, respectively. The interactions between the software system and the rejuvenation mechanism are modeled through update functions: transition `clock` is associated with an update function that flushes places `Up`, `Down`, and `Detected` to represent system switch-off during rejuvenation; transition `rejuvenate` is associated with an update function that assigns a token to place `Up` to model system restart after rejuvenation; transition `detect` is associated with an update function that flushes place `Wait` to account for disabling of rejuvenation during unplanned repair; and, transition `repair` is associated with an update function that adds a token to place `Wait` to represent reschedule of rejuvenation. The caption of Fig. 2 provides the exact definition of the update functions: specifically, ORIS accepts a semicolon-separated list of update functions with syntax "`p` $= e$" where `p` is a place name and e is an expression made of place names, constants, and operators (the exact syntax of expression e is described in Section 3.1).

Firm timing constraints and reset sets associated with transitions contribute to the definition of the set of feasible behaviors by extending the basic PN model into a Time Petri Net (TPN) [8]. Note that this determines the set Ω of timed firing sequences (i.e., transition firing sequences, each associated with the time elapsed between each pair of consecutive firings). In the example of software rejuvenation, we assume that the system specification requires the time to detect a failure to be lower than $4\,\mathrm{h}$, the repair time to be between 4 and $24\,\mathrm{h}$, the rejuvenation time to be lower than $2\,\mathrm{h}$, and the rejuvenation period to be equal to $168\,\mathrm{h}$ (7 days). According to this, in Fig. 2, transitions `detect`, `repair` and `rejuvenate` have support $[0,4]$ h, $[4,24]$ h, and $[0,2]$ h, respectively, while transition clock has a deterministic value of $168\,\mathrm{h}$. Conversely, we assume that the failure time is unbounded, thus transition `fail` has support $[0,\infty)$ h. Note how bounded firing intervals restrict the set of possible behaviors: for instance, if the system fails $160\,\mathrm{h}$ hours after it has been started, the failure will be detected with certainty before rejuvenation; in contrast, if inter-rejuvenation time had support $[160,168]$ h, rejuvenation could be triggered before failure detection. Finally, the initial marking is `Up Wait` so that the software system is initially up and the timer to the next rejuvenation has just started.

2.3 Deriving the stochastic parameters of the model

Stochastic duration of system activities and random choices are captured by probability distributions and weights associated with transitions, respectively. The characterization of these stochastic parameters identifies an STPN model that casts the set Ω of timed firing sequences determined by the underlying TPN into a probability space [55].

ORIS supports *expolynomial* distributions (also known in the literature as *exponomials* [65]) obtained as products of exponentials and polynomials, on bounded or unbounded supports, with an analytical representation over the entire domain or piecewise-defined over multiple sub-domains. Expolynomials are general expressions with EBNF syntax

$$\begin{aligned}\text{EXPR} &:= \text{PROD}\ \{\ +\ \text{PROD}\ \}\\ \text{PROD} &:= \text{FLOAT}\ \{\ \star\ \text{TERM}\ \}\\ \text{TERM} &:= \texttt{x}\ \mid\ \texttt{x\^{}}\ \text{INT}\ \mid\ \texttt{Exp[}\ \text{FLOAT}\ \ \texttt{x]}\end{aligned}$$

where FLOAT and INT are floating-point and integer constants, respectively, e.g., `4.0 * Exp[-2.0 x] * x^2`.

Expolynomials permit the representation of common distributions (e.g., Erlang, uniform) and enable a variety of approaches to fit data (e.g., moments [70], shape [45], [57]) obtained in different manners (e.g., estimated from real measurements, synthetically generated, derived from system specification). For instance, in the example of software rejuvenation, we assume that the failure time has been repeatedly observed, obtaining measurements lower than $x_1 = 72\,\mathrm{h}$ (3 days), $x_2 = 144\,\mathrm{h}$ (6 days), and $x_3 = 216\,\mathrm{h}$ (9 days) with frequency $p_1 = 0.001$, $p_2 = 0.006$, and $p_3 = 0.016$, respectively; and, with frequency $p_4 = 0.984$, measurements larger than x_3, for which the mean value was $672\,\mathrm{h}$ (28 days). These measurements can be modeled by the PDF of transition `fail`, with piecewise representation over 4 intervals: *i*) 3 uniform PDFs with value 0.0000139, 0.0000694, and 0.000139 over $[0,72)$ h, $[72,144)$ h, and $[144,216)$ h, respectively, fitting the PDF of time to failure within each interval $[x_i, x_{i+1})$ as $(p_{i+1} - p_i)/(x_{i+1} - x_i)$ $\forall i \in \{0,1,2\}$, with $x_0 = 0$ and $p_0 = 0$; and, *ii*) a shifted exponential PDF $f(x) = p_4 \alpha \exp(-\lambda x)$ for $x \in [x_3, \infty)$ with rate $\lambda = 0.002193$, shift $x_3 = 216\,\mathrm{h}$, and $\alpha = 0.003522$ (which has mean value of $672\,\mathrm{h}$ when $x > x_3$). Other, more complex, expolynomial PDFs could also be used. Conversely, we assume no data measurements for the time spent in failure detection, repair, and rejuvenation; thus transitions `detect`, `repair` and `rejuvenate` can be associated with uniform distributions over $[0,4]$ h, $[4,24]$ h, and $[0,2]$ h, respectively.

ORIS also supports the definition of transition weights as expressions e built from token counts in the current marking (e.g., `2.0*p1`, the exact syntax is defined in Section 3.1); in particular, weights, used to select one among the enabled IMM transitions (or DET transitions with the same value), allow modeling of discrete probabilistic choices that depend on the logical state of the system (e.g., probability of message losses depending on the conditions of the radio channel). Rates of EXP transitions can also be expressed as functions of the current marking, representing activity durations that depend on the logical state of the system (e.g., service rates depending on the number of available servers).

2.4 Validating the model

As usual in the lifecycle of a model, a validation step is needed to achieve confidence about its correspondence with the intended (often implicit) modeling aim. This can be effectively supported by interactive simulation, which in ORIS is implemented in the form of a token game.

Specifically, the state space of the STPN model can be explored either manually, by selecting the next transition to fire from the set of firable transitions, or automatically, by specifying a number of transition firings to be executed or a stop condition (i.e., a function of the current marking, see Section 3.1). To support inspection of the model and gain insight into its behavior, the firing probability and the range of accepted firing times of each firable transition are evaluated using the method of stochastic state classes [44].

3 The ORIS Workflow: Model Evaluation

Reward properties and *stop conditions* allow the user to evaluate the probability of a subset of execution paths satisfying specific criteria, and to calculate the expected value of rewards accrued in each state of such paths. In this section, we illustrate how these mechanisms can be used in ORIS for the quantitative evaluation of non-functional requirements from an STPN model of the system (Section 3.1). Then, we compare the different analysis engines available in ORIS for the evaluation of reward properties (Section 3.2), and discuss their limitations and complexity factors, also with respect to alternative methods and tools (Section 3.3). Finally, we recall the method of *stochastic state classes*, which forms the basis for the regenerative engine, the most distinctive feature of ORIS (Section 3.4).

3.1 Reward Properties and Stop Conditions

Rewards. For the quantitative evaluation of an STPN model, ORIS supports formulation and evaluation of rewards defined from the *marking process* $\mathbb{M} = \{M(t),\, t \in \mathbb{R}_{\geq 0}\}$ where $M(t)$ is the marking at time $t \geq 0$. A reward r is an expression that combines constants and token counts so as to define a real-valued function over the set of markings. Formally, ORIS accepts any expression e with the following syntax:

$c :=$ *place id* | *constant*

$e :=$ `c` | `(e)` | `e+e` | `e-e` | `e/e` | `e*e` | `e^e` | `e==e` | `e!=e` | `e>e` | `e>=e` | `e<e` | `e<=e` | `e&&e` | `e||e` | `!e` | `If(e,e,e)` | `min(e,e)` | `max(e,e)`

where place identifiers evaluate to the number of tokens assigned by the marking $M(t)$, operators have the same precedence as in Java, comparison operators return `0` or `1`, and `If(`e_1`,`e_2`,`e_3`)` is a ternary conditional if operator as in the Java language (i.e., it evaluates to the value of e_2 or the value of e_3 depending on whether e_1 evaluates to TRUE or FALSE, respectively). In particular, the latter construct is often used in the form `If(`φ`,1,0)` which returns the probability of φ, i.e., the measure of probability of the set of behaviors where the marking satisfies the (state) property φ. For example, if $\mathtt{p}_1$ and $\mathtt{p}_2$ are names of places in the model, the reward expression `If(p1+p2>0,1,0)` evaluates the probability of states where at least one token is contained in $\mathtt{p}_1$ or $\mathtt{p}_2$. Since e_1 and e_2 are themselves expressions, the ternary operator also permits more complex forms implementing an if-then-else logic, with possible nesting, as enabled by the ternary operator in the Java or C languages. Conversely, a reward can also be defined as a simple expression e, which is a shorthand of the form `If(true,`e`,0)` and evaluates the expected value of expression e, e.g., the reward expression `p1+p2` evaluates the expected value of the sum of tokens in places $\mathtt{p}_1$ and $\mathtt{p}_2$.

ORIS allows the evaluation of the expected value of reward expressions during the transient evolution of the stochastic process and at steady-state. In particular, let $p_{ij}(t) = P\{M(t) = j \,|\, X_0 = i\}$ and $\overline{p}_{ij} = \lim_{t\to\infty} p_{ij}(t)$ represent, respectively, the transient and steady-state probabilities for each initial regeneration $i \in \mathcal{R}$ (an initial marking where times to fire are sampled independently) and each marking $j \in \mathcal{M}$; then, a reward r is evaluated for each marking $j \in \mathcal{M}$ to compute the instantaneous expected reward $I_r^i(t) = \sum_{j\in\mathcal{M}} r(j)\, p_{ij}(t)$ at each time t, its steady-state expected value $\overline{I}_r^i = \lim_{t\to\infty} I_r^i(t) = \sum_{j\in\mathcal{M}} r(j)\, \overline{p}_{ij}$, or its cumulative expected value $C_r^i(t) = \int_0^t I_r(u)\, du$ up to t.

Rewards can be used for the evaluation of a variety of non-functional requirements. For example, for the model of software rejuvenation in Fig. 2, a measure of *system availability* can be expressed as $r(m) = \mathtt{Up}$, which will return the probability to be in a correctly functioning state.

Stop Conditions. While rewards define quantitative properties over system states, *stop conditions* can turn states that satisfy a Boolean predicate into absorbing states where all transitions are disabled and the system sojourns indefinitely. This mechanism allows the evaluation of quantitative measures where only a subset of execution paths is of interest. For example, *system reliability* is defined as the "probability that the system will continuously perform its intended function during a specified period of time $[0, T]$": the system must be continuously available during the time interval $[0, T]$. We can evaluate this property in ORIS using a stop condition $s(m)$ that evaluates to TRUE when the system is down, i.e., `Up==0`. Then, the instantaneous expected value of the reward $r(m) = \mathtt{Up}$ at time T excludes execution paths where the system is up at time T but has visited a down state before: the reward then gives the probability that the system is up at time T and has never reached a down state in $[0, T]$.

In general, stop conditions are Boolean functions $s\colon \mathcal{M} \to \{\text{TRUE}, \text{FALSE}\}$ over the set of markings of an STPN. They are specified using the same syntax as rewards in ORIS, where any nonzero value is considered TRUE.

Notably, stop conditions allow the evaluation of reach-avoid objectives equivalent to a *bounded until operator* $\varphi_1 \mathcal{U}^{[0,t]} \varphi_2$ of probabilistic model checking [55]. This operator specifies the set of model behaviors where a safety condition φ_1 is always satisfied until a goal condition φ_2 is reached within the time bound t; to evaluate the probability measure of these behaviors, the user can run transient analysis using the stop condition $!\varphi_1 \,||\, \varphi_2$ (i.e., stop on illegal or goal states) and evaluate the reward φ_2 for each time instant t (i.e., compute the probability that a goal state is reached by time t traversing only safe states). Probabilistic until and *probabilistic existence* $\mathrm{TRUE}\ \mathcal{U}^{[0,t]} \varphi_2$ (a special case) were found to be the two most frequent specification patterns for quality and dependability requirements of software-intensive systems in application domains such as medical applications, automotive systems, air traffic control, and railway signalling [42].

Example. The model of software rejuvenation shown in Fig. 2 can be analyzed in ORIS using regenerative transient analysis, which is performed with time limit $t_{max} = 1344\,\mathrm{h}$ (corresponding to 8 weeks of wall-clock time) and with step size $k = 0.005\,\mathrm{h}$, so that transient probabilities are evaluated for all $t = 0, k, 2k, \ldots, \lfloor \frac{t_{max}}{k} \rfloor k$.

To evaluate system availability, we compute the instantaneous reward `Down>0||Detected>0||Rej>0`, which gives the *transient unavailability*, i.e., the probability that the system is not working at time t, with `Up Wait` being the initial marking. We also evaluate the *cumulative unavailability*, i.e., the expected outage time within the interval $[0, t]$. Plots in the top and middle of Fig. 3 depict transient and cumulative unavailability of the system, respectively; unavailability is very high (nearly 0.993) at time $168\,\mathrm{h}$, since this is the time of the first scheduled rejuvenation and the probability that the system fails before $168\,\mathrm{h}$ is low (nearly 0.009). As time progresses, failures and repairs before rejuvenation become more frequent, reducing unavailability at peaks produced by the initial schedule (rejuvenation is rescheduled after repair), and slightly increasing unavailability between peaks.

To evaluate system reliability, we compute the instantaneous reward `Down` with stop condition `Down>0`, yielding the *transient unreliability*, i.e., the transient probability that the system has failed at least once by time t (again, with `Up Wait` being the initial marking). As shown by the plot at the bottom of Fig. 3, system reliability is quite low: it could be improved by increasing the frequency of rejuvenations, at the cost of reducing system availability.

The results from Fig. 3 could be used to determine whether the software system meets given unavailability and unreliability requirements, and to predict the time intervals during which it is subject to maintenance. Moreover, evaluation of the considered rewards could also be repeated for different values of the model parameters (notably, for different values of the rejuvenation period), so as to derive the time to the next rejuvenation that achieves the best trade-off between reliability and availability of the software system.

3.2 Analysis Engines and Complexity Factors

As illustrated in Fig. 1, ORIS provides a suite of analysis engines that implement different solution methods, but support the evaluation of STPN properties specified using the same input format for rewards and stop conditions. Each analysis engine imposes different limitations on the class of the underlying stochastic process of the STPN, which result in more efficient solution methods, but can require additional care (or approximations) during the modeling phase. We provide here an overview of all analysis engines.

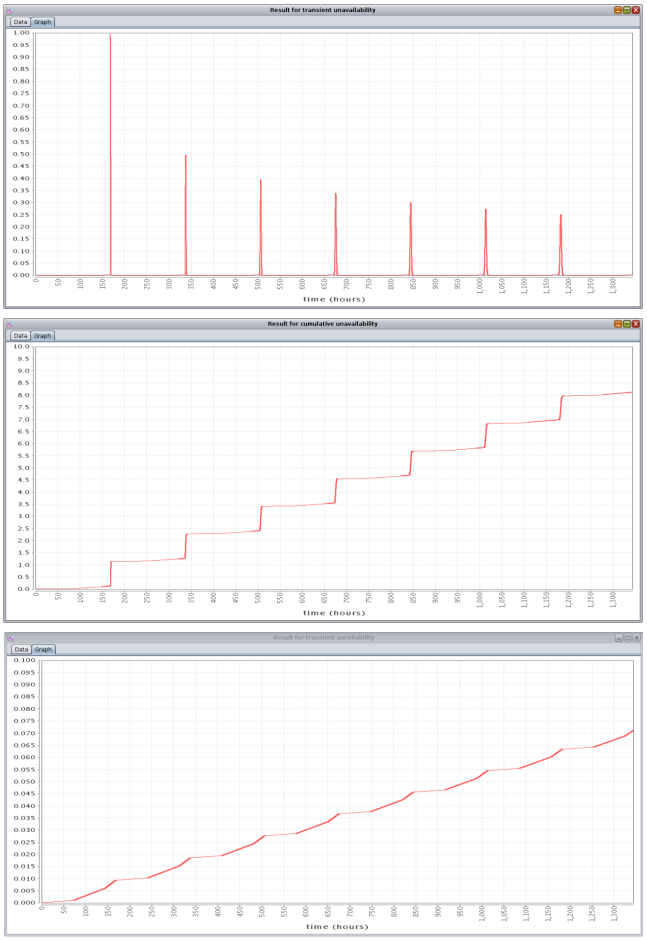

Figure 3: GUI of ORIS: for the model of software rejuvenation of Fig. 2, plots show transient rewards providing the transient unavailability (top), cumulative unavailability (center), and transient unreliability (bottom). Transient probabilities and rewards (instantaneous or cumulative up to time t) can be visualized also in table form (steady-state probabilities and rewards can be visualized in table form only).

Markovian Engine. This engine implements standard methods for transient and steady-state analysis of CTMCs [62]. It requires STPNs with only EXP and IMM transitions (also known in the literature as GSPNs [2]); in practice, to overcome this severe limitation, each GEN transition can be replaced (prior to the analysis) with the sequence of EXP transitions produced by a phase-type approximation (obtained, for example, using PhFit [45]).

The implementation uses dense transition matrices of size $|\mathcal{M}| \times |\mathcal{M}|$ to solve linear systems (for steady-state analysis in each bottom component of the process) and for matrix-vector multiplications in the method of *uniformization*, adopting the Fox-Glynn algorithm [34] to select the number of iterations that guarantee an error bound ϵ at each time point.

Enabling Restriction Engine. This engine implements transient analysis for MRPs under enabling restriction [41]. It requires STPNs with at most one GEN transition enabled in each state (this requirement is checked by ORIS at the beginning of the analysis). This method partitions the state space in subgraphs where only a specific GEN transition is enabled: the transient solution of each subgraph (computed with uniformization) is used to evaluate the global and local kernels of MRP, which are then used to solve a system of Volterra integral equations [50].

Similarly to the Markovian engine, the implementation uses dense transition matrices of size $|\mathcal{M}| \times |\mathcal{M}|$ for uniformization; the solution of Volterra integral equations requires $O(T^2)$ multiplications of $|\mathcal{R}| \times |\mathcal{M}|$ matrices, where $|\mathcal{R}|$ is the number of MRP subgraphs, $|\mathcal{M}|$ is the number of markings, and T is the number of time ticks (in the time interval $[0, T\Delta t]$ under analysis, for a given time step Δt).

Regenerative Engine. This engine implements transient and steady-state analysis of MRPs with multiple GEN transitions enabled in each state [44]. Steady-state analysis requires a bounded number of transition firings until a *regeneration*, i.e., a state where all transitions are newly-enabled; this condition can be checked for a given STPN by the (terminating) algorithm for nondeterministic analysis of the underlying TPN. In contrast, transient analysis can lift this restriction by allowing an error in the enumeration of MRP subgraphs [44].

The implementation is based on the enumeration of stochastic state classes, which encode (symbolically) the joint PDFs of transition timers after each firing. The analysis complexity depends on the number of classes and on the analytical form of the PDFs [60]. Once kernels are evaluated using stochastic state classes, steady-state analysis requires the solution of an $|\mathcal{R}| \times |\mathcal{R}|$ linear system, while transient analysis requires the solution of a system of Volterra integral equations, through $O(T^2)$ multiplications of $|\mathcal{R}| \times |\mathcal{M}|$ matrices, where $|\mathcal{R}|$ is the number of MRP subgraphs, $|\mathcal{M}|$ is the number of markings, and T is the number of time ticks.

Forward Engine. This engine implements transient analysis of STPNs without restrictions on the presence of regenerations [44]. The implementation enumerates a single graph of stochastic state classes from the initial state, until the earliest firing times $EFT(\gamma)$ of transitions along each sequence surpass the target time bound of the analysis. The computational cost depends on the number and length of firing sequences before the time bound, and on the complexity of PDF functions (number of terms in their analytical form). To reduce the number of enumerated classes, the user can specify a truncation error $\epsilon > 0$: in this case, the analysis guarantees the error bound $0 \leq p_{ij}(t) - \tilde{p}_{ij}(t) \leq \epsilon$ between approximate transient probabilities $\tilde{p}_{ij}(t)$ and exact ones $p_{ij}(t)$. A truncation error ϵ is required when the STPN allows cycles of transitions firing in zero time; effects of the truncation error on the analysis are evaluated in [44].

Nondeterministic Engine. In addition, ORIS supports nondeterministic analysis of the state space of STPNs [68]: the dense set of timed states reached by an STPN is encoded as a directed graph (*state class graph*) where edges are transition firings and nodes are *state classes* comprising a marking and a set of timer values. This analysis supports verification of qualitative properties of the model: for example verifying whether a marking can be reached, or whether the state class graph contains a regeneration on each cycle (allowing exact regenerative analysis) or cycles of IMM transitions (requiring $\epsilon > 0$ in the forward engine).

3.3 Advantages over alternative tools

Several alternative tools have been developed over the years for the evaluation of quantitative properties from models of stochastic systems. ORIS provides a contribution that is twofold, bringing improvements both to the applicability of analysis methods and to the user workflow.

On the one hand, ORIS extends the evaluation of quantitative properties to a larger class of stochastic processes, beyond Markov chains (analyzed in PRISM, MRMC, SMART), and beyond models with at most one GEN timer in each state (analyzed in TimeNET) or with at most one DET timer in each state (analyzed in MC4CSL^{TA} [4]). The method of stochastic state classes, leveraged by the regenerative and forward engines of ORIS, allows the analysis of models including, in each state, multiple GEN timers: deterministic, uniform, or timers with expolynomial PDFs, possibly with *firm lower and upper bounds*. This class of models is of particular importance for the analysis of systems where concurrency is limited through firm timing constraints (e.g., firm lower and upper bounds of activity duration in real-time systems). It is also important when GEN distributions are required not because of firm timing constraints between activities, but for an accurate and compact fitting of experimental data. We stress the fact that, in this latter case, phase-type distributions [45] can be used to fit input data with EXP timers [56], at the expense of a state-space explosion (producing models that can be analyzed with tools such as PRISM). Conversely, analysis methods applicable to more general classes of stochastic processes incur higher computational costs; and it is recommended to select the most specific analysis method.

On the other hand, in the user workflow, ORIS allows seamless integration of GUI modeling and validation with a programmatic approach based on the SIRIO API. Using the graphical interface, the user can quickly develop a model and validate its operation using the interactive simulator or through the evaluation of quantitative requirements; once validated, the model can be exported from ORIS as Java code using the API of the SIRIO library. This integration allows the user to introduce quantitative evaluation of parametric non-Markovian models into larger software projects, such as tools for computer-aided design of critical infrastructures [21], [25], autonomic systems using *models at runtime* [15], ambient assisted living applications [20]. In contrast, tools such as TimeNET or GreatSPN allow graphical editing of a model, but not its integration in other software projects; and tools such as PRISM or Storm require input models specified with text-based formalisms.

3.4 The method of stochastic state classes

The method of stochastic state classes integrates correctness verification of feasible behaviors (identified by the underlying TPN) with quantitative evaluation of their probability (induced by the stochastic parameters of the STPN). Specifically, in the execution of a TPN, a sequence of transitions can be fired with different timings, each reaching the same

marking but with possibly different values of the age and the vector of remaining times of enabled transitions. The set of all the reachable values of this vector takes the shape of a particular convex linear polyhedron, usually termed *zone*, which can be efficiently encoded and manipulated as a Difference Bounds Matrix (DBM) [32]. The composition of a DBM zone and a marking (or, more generally, a logical location) is usually called *state class*, and comprises the basic abstraction for finite symbolic representation of the state space of any model where all active timers advance with the same speed. This has been widely exploited in a number of software tools for qualitative verification of concurrent models with non-deterministic timers based on TPNs [9], [36], [68] or timed automata [7], [30].

The basic idea of *stochastic state classes* is to extend state classes with a multivariate distribution over the allowed values of the age and vector of remaining times of transitions (the *support* of their probability density). The analytical form of this distribution turns out to be derivable through efficient symbolic manipulation, provided that all the durations in the model are either deterministic or expolynomial. Many of the distinctive capabilities of the ORIS tool develop on an efficient Java implementation of this derivation and on its combination with Markov-regenerative analysis.

More specifically, a *stochastic state class* [69] encodes: *i*) a marking; *ii*) a joint support for the remaining times to fire of the enabled transitions and for the absolute elapsed time (termed *age*); and, *iii*) a joint PDF for such values.

Definition 4 (Stochastic state class)**.** A *stochastic state class* is a tuple $\Sigma = \langle m, D_{\langle \tau_{age}, \vec{\tau} \rangle}, f_{\langle \tau_{age}, \vec{\tau} \rangle} \rangle$ where: $m \in \mathcal{M}$ is a marking; $f_{\langle \tau_{age}, \vec{\tau} \rangle}$ is the PDF (immediately after the previous firing) of the random vector $\langle \tau_{age}, \vec{\tau} \rangle$ including the age timer τ_{age} and the times to fire $\vec{\tau}$ of transitions enabled by m; and, $D_{\langle \tau_{age}, \vec{\tau} \rangle} \subseteq \mathbb{R}^{n+1}$ is the support of $f_{\langle \tau_{age}, \vec{\tau} \rangle}$.

Given a stochastic state class Σ, a *succession relation* provides the joint support and the joint PDF of the random vector $\langle \tau_{age}, \vec{\tau} \rangle$ conditioned on the firing of a transition γ.

Definition 5 (Succession relation)**.** $\Sigma' = \langle m', D', f'_{\langle \tau_{age}, \vec{\tau} \rangle} \rangle$ is the *successor* of $\Sigma = \langle m, D, f_{\langle \tau_{age}, \vec{\tau} \rangle} \rangle$ through transition γ with probability μ, and we write $\Sigma \overset{\gamma,\mu}{\Longrightarrow} \Sigma'$, if, given that the marking of the STPN is m and $\langle \tau_{age}, \vec{\tau} \rangle$ is distributed over D according to $f_{\langle \tau_{age}, \vec{\tau} \rangle}$, then: γ has nonzero probability μ of firing in Σ; if γ fires in Σ, its firing yields the marking m' and, conditioned on this event, the new vector of times to fire $\langle \tau'_{age}, \vec{\tau}' \rangle$ is distributed over D' according to $f'_{\langle \tau_{age}, \vec{\tau} \rangle}$.

The relation $\overset{\gamma,\mu}{\Longrightarrow}$ can be enumerated by computing the probability of transition firings and the support and PDF of the random vector $\langle \tau_{age}, \vec{\tau} \rangle$ in successor classes [69]. In particular, the support is encoded as a DBM and, for models with expolynomial GEN distributions, the PDF takes a piecewise representation over a partition of the support in DBM sub-zones [19]. Starting from an initial stochastic state class in which the times to fire of the enabled transitions are independently distributed, *transient trees* can be computed where nodes are stochastic state classes and edges are labeled with transitions and their firing probabilities, supporting the derivation of quantitative measures for the transient and steady-state regime of STPNs.

For STPNs with underlying MRP, *regenerative transient analysis* [44] enumerates transient trees of stochastic state classes reached within a given time limit between each pair of *regeneration points* [26], [50], i.e., selected transition firings after which future evolution is independent of past history and univocally determined by the specific *regeneration* value. Transient measures between regeneration points are encoded in a local kernel and a global kernel, and then combined through a system of Markov renewal equations to compute transient probabilities of reachable markings.

A powerful class of "extended regenerations" can be detected in STPNs with DET transitions by verifying whether all GEN timers are reset or have been enabled for a deterministic time: in this case, given the current marking and the enabling times of GEN timers, the probability distribution of future states is conditionally independent of the previous history. This idea, introduced in [55], generalizes the usual concept of regeneration from the literature on stochastic Petri nets [27], which requires all enabled GEN transitions to be newly enabled (i.e., with enabling time equal to 0), and extends the applicability of regenerative analysis to a larger class of models with DET and GEN transitions.

4 Case Studies

We illustrate how ORIS can be exploited in quantitative evaluation of non-Markovian models by reviewing some selected cases of application [55], [24], [14], [25], [13], [20], [11], in various domains and with different usage patterns.

4.1 Software Engineering

4.1.1 Model checking of real-time systems

In real-time software models, firm bounds on task durations are often crucial in guaranteeing the system correctness. A representative example is Fischer's protocol [52], which guarantees mutual exclusion of a set of processes that compete for access to a critical section by imposing firm bounds on the duration of write operations and test delays.

Quantitative properties of this protocol were analyzed in [35] for read and write operations modeled with exponential or Erlang distributions, which do not guarantee mutual exclusion. In [53] and [55], using the regenerative engine of ORIS, it was possible to analyze steady-state and transient properties, respectively, of a correct model of the protocol, where read and write operations were modeled by GEN transitions with firm upper and lower bounds.

Fig. 4 illustrates such STPN model for three processes P_1, P_2, P_3: each horizontal section represents a process, while place `id` models a shared variable that can take the values $\{0, 1, 2, 3\}$. When `id` is empty, a process P_i for $i = 1, 2, 3$ can access the critical section by (1) setting the number of tokens of `id` to its identifier i (through the update function `id=`i of transition $\texttt{write}_i$), (2) waiting for a time greater than the maximum write time of any other process ($\texttt{wait}_i$ fires after 1.1 time units, since transitions $\texttt{writing}_i$ are uniformly distributed over $[0, 1]$), and then (3) ensuring that the number of tokens in `id` has not changed (i.e., no other process has started an access to the critical section). If some other process has changed the token count of `id`, process P_i aborts and waits for `id` to become empty before starting

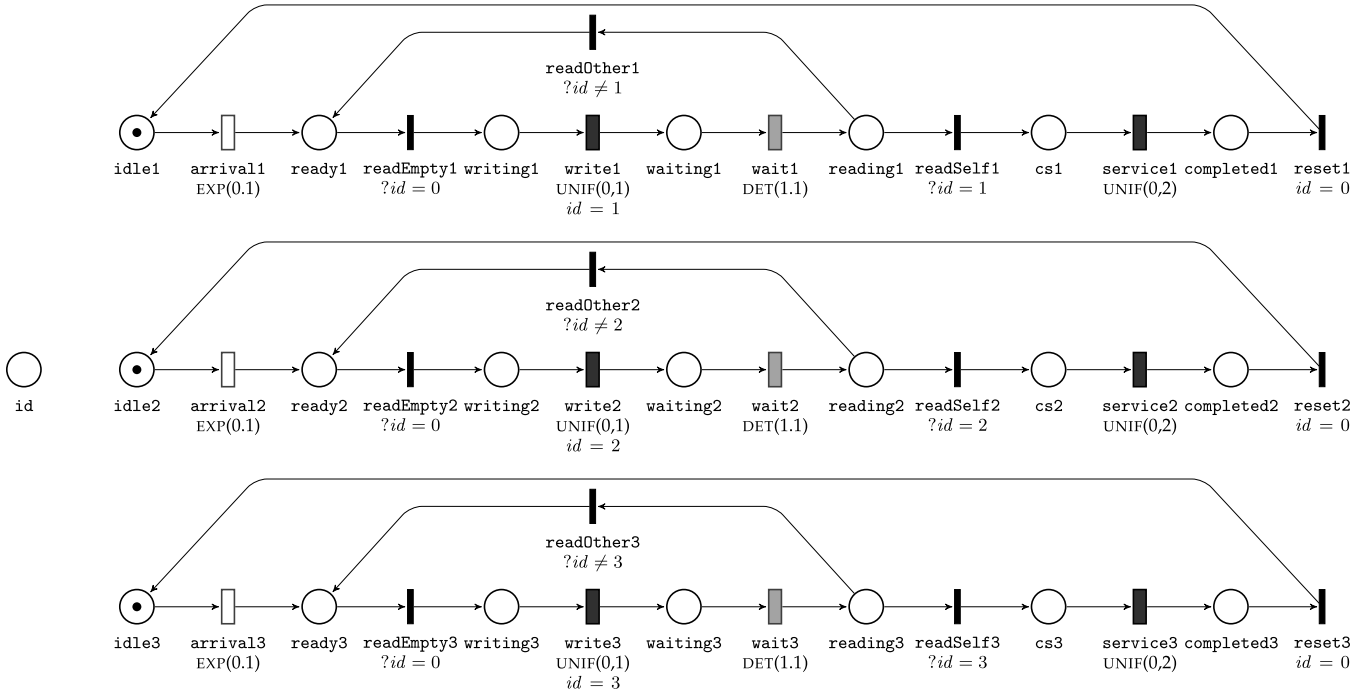

Figure 4: STPN model of Fischer's protocol, a software engineering case study analyzed in [53], [55].

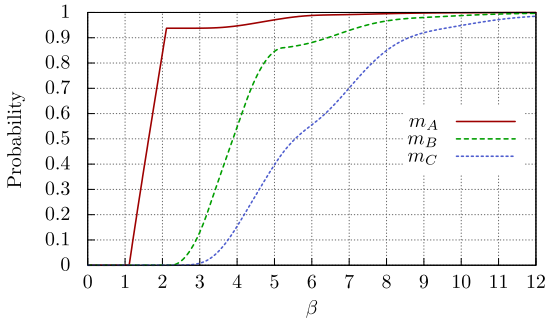

Figure 5: Probability that process P_1 enters the critical section for the first time within time β (transient reward `cs1` with stop condition `cs1==1`) from different initial states.

a new attempt. These conditions on `id` are checked using enabling functions `id==`i and `id!=`i for IMM transitions `readSelf`$_i$ and `readOther`$_i$, respectively. The time spent in the critical section is modeled as uniformly distributed over $[0, 2]$ for all processes (transitions `service`$_i$), while idle time is exponentially distributed with rate 0.1 (transitions `arrival`$_i$). Mutual exclusion is enforced by firm time bounds: after the waiting phase, a process can detect concurrent accesses and let other processes enter the critical section [52].

In Fig. 5, we illustrate the probability that process P_1 enters the critical section within time β, for $\beta \in [0, 12]$ and different initial states: P_1 is ready while P_2 and P_3 are idle (m_A = `ready1 idle2 idle3`), P_1 is ready but P_3 has already started to access the critical section (m_B = `3 id ready1 idle2 waiting3`), P_1 is ready but P_2 is trying to access the critical section after P_3 (m_C = `3 id ready1 writing2 waiting3`). This property can be expressed as the instantaneous reward `cs1` with stop condition `cs1==1`, comprising a *probabilistic existence* pattern [42].

4.1.2 Testing of real-time systems

In the testing process of concurrent timed systems including both *controllable* (*nondeterministic*) temporal parameters and *noncontrollable* (*stochastic*) temporal parameters, input generation is a complex task aimed at identifying the values of controllable parameters that maximize the probability that the system executes selected test cases [43], [49], [71]. This context is another testbed for modeling and analysis of systems with firmly bounded stochastic durations.

In [24], input generation is addressed with reference to a general system model, where controllable parameters take values in a min-max (possibly unbounded) interval and noncontrollable parameters have a non-Markovian distribution (possibly with bounded support). To this end, the SIRIO API is used to derive the dense set of timings that let the system run along a selected test case and to associate it with the measure of probability induced by the distributions of noncontrollable parameters. Then, the probability of conclusive test execution is derived as a function of values of controllable parameters, either in symbolic or in numerical form. On the one hand, the symbolic solution repeatedly integrates multi-variate PDFs with DBM support to eliminate non-controllable variables, with domain partitioning and polynomial degree growing with the length of the trace (i.e., number of events of the test case) and with the concurrency degree of the model. On the other hand, the numerical solution avoids the curse of dimensionality of symbolic integration while suffering the number of samples to be generated and the number of expolynomial terms in the considered multi-variate PDFs.

The approach is applied to the testing of real-time software, considering a set of concurrent tasks with controllable release times and non-controllable execution times. In particular, the latter are assumed to have an Erlang distribution truncated over the execution time range, so as

to preserve a nonzero Best Case Execution Time (BCET) and a firm Worst Case Execution Time (WCET). Experimental results prove that the approach can be effectively used for input generation, significantly reducing the number of test repetitions with respect to random testing.

4.2 Systems Engineering

4.2.1 Performability evaluation of railway signalling systems

In Level 3 of the European Rail Traffic Management System/European Train Control System (ERTMS/ETCS-L3) [33], the maximum distance that a train is allowed to travel is computed based on the minimum safe rear-end of the foregoing train (*moving-block signalling*), exploiting the Global System for Mobile Communications-Railway (GSM-R) [1] for a continuous bidirectional communication between trains and ground controllers. Hence, headways between trains can be considerably reduced (in principle to the braking distance) provided that the GSM-R is highly available [73]. Evaluation of emergency stops caused by consecutive message losses provides a challenging testbed for ORIS, given that the periodic exchange of messages between ground and on-board controllers may be impaired by synchronous and asynchronous phenomena with different time scales.

In [13], an ERTMS/ETCS-L3 scenario is considered where a foregoing train sends Position Reports (PRs) to a ground controller which, in turn, sends Movement Authorities (MAs) to the chasing train. Transient failures of the GSM-R may cause the loss of end-to-end messages (impairing transmission of PRs or MAs), due to *burst noise*, *connection losses* (requiring repetition of connection trials after a timeout), or *handovers* between neighboring radio stations (occurring periodically given the regular distance between radio stations). Evaluation of a flat model of this scenario is not feasible due to the complexity of behaviors resulting from multiple concurrent GEN timers with overlapping activity intervals. Moreover, approximation of GEN transitions having firmly bounded support would reduce the accuracy of results, while stochastic simulation would suffer the presence of rare events and the different order of magnitude of durations.

The approach of [13] resorts to the STPN model of Fig. 6, which consists of: *i*) a submodel representing communication failures due to handovers as a *periodic* process with GEN jitter and initial DET offset; *ii*) a submodel with M places (watch variables) keeping memory of which among the last M end-to-end messages have been lost due to handovers (in Fig. 6, $M = 4$); and, *iii*) a submodel accounting for the *periodic* generation and transmission of PRs and the subsequent computation and transmission of MAs. The latter submodel also represents the event (modeled by transition `failure`) that, among the last M end-to-end messages, those that are not lost due to handovers are lost due to burst noise or connection losses, causing an emergency braking (without conditioning on the fact that no emergency braking occurred before). In particular, the probability of such event (i.e., the weight of transition `failure`) is computed from the probability that burst noise or connection losses impair a given number of consecutive end-to-end messages, which in turn is derived analytically. Finally, probability distributions are derived by fitting the ERTMS/ETCS specification with (possibly piecewise) uniform PDFs or with DET values.

The model is solved by regenerative transient analysis using SIRIO, computing an upper bound on the first-passage time distribution of a spurious emergency braking as the instantaneous expected value of the reward `Failure` with stop condition `Failure==1`. The analysis incurs medium complexity mainly due to the value of M, the number of concurrent GEN and DET transitions, and the length of behaviors between consecutive regenerations. Since the system behavior is recurrent over the hyper-period of periodic message releases and periodic arrivals at cell borders, the upper-bound on the probability that a spurious emergency stop occurs within a hyper-period is used as the parameter of a geometric distribution to derive an upper bound $\tilde{\beta}(M,t)$ on the first-passage probability that M consecutive losses occur within a time interval $[0,t]$ of arbitrary duration.

The reduced computational complexity makes it possible to perform a sensitivity analysis with respect to the number M of consecutive tolerated losses and the headway distance T_{h} between trains. Fig. 7 plots $\tilde{\beta}(M,t)$ for increasing values of M and T_{h}. Specifically, $\tilde{\beta}(M,t)$ significantly decreases by one order of magnitude as T_h increases by $6\,\mathrm{s}$, which corresponds to an increase of M by 1, being at time $t \sim 1\,\mathrm{h}$ in the order of $4.3 \cdot 10^{-1}$, $1.5 \cdot 10^{-2}$, and $6.9 \cdot 10^{-4}$ for $T_{\mathrm{h}} = 60\,\mathrm{s}$ ($M = 2$), $T_{\mathrm{h}} = 66\,\mathrm{s}$ ($M = 3$), and $T_{\mathrm{h}} = 72\,\mathrm{s}$ ($M = 4$), respectively. The obtained results could be used to select the delay T_{h} so as to achieve a trade-off between the utilization of the railway line and the probability that a train is stopped within $1\,\mathrm{h}$; in turn, the delay T_{h} corresponds to a headway distance, which determines the maximum number M of consecutive tolerated losses. Overall, results prove that the headway distance must be significantly larger than the braking distance to effectively limit the expected number of spurious emergency stops.

4.2.2 Performability evaluation of gas distribution networks

During a repair procedure, affected components of a gas distribution network are isolated by opening or closing some valves, and the consequent deviation from nominal network operation is mitigated by regulating the supply pressure [17]. Given that repair steps take random durations, valve and pressure control is performed at stochastic times, yielding different completion times of the procedure phases and different risk that pressure becomes insufficient for demand. Evaluation of the network performability is relevant in the applicative perspective, and it permits illustrating how ORIS can be used to analyze workflows with concurrent steps that have firmly bounded non-Markovian duration and may be suspended for a deterministic time (i.e., non-working hours).

To facilitate the procedure modeling, the approach of [14] extends STPNs implemented in the SIRIO library with two transition features (not included in the present distribution): *i*) *fickle functions*, which shift the time to fire of persistent transitions by a deterministic value upon a transition firing, representing suspension and resumption of activities at deterministic time points according to work shifts of personnel; and, *ii*) *marking-dependent CDFs*, which sample the time to fire at newly-enabling from a different CDF depending on the marking, modeling pressure restoration with duration that depends on the number of completed regulation steps. GEN distributions of repair times are assumed to be derived from data that could be actually captured by the increasingly

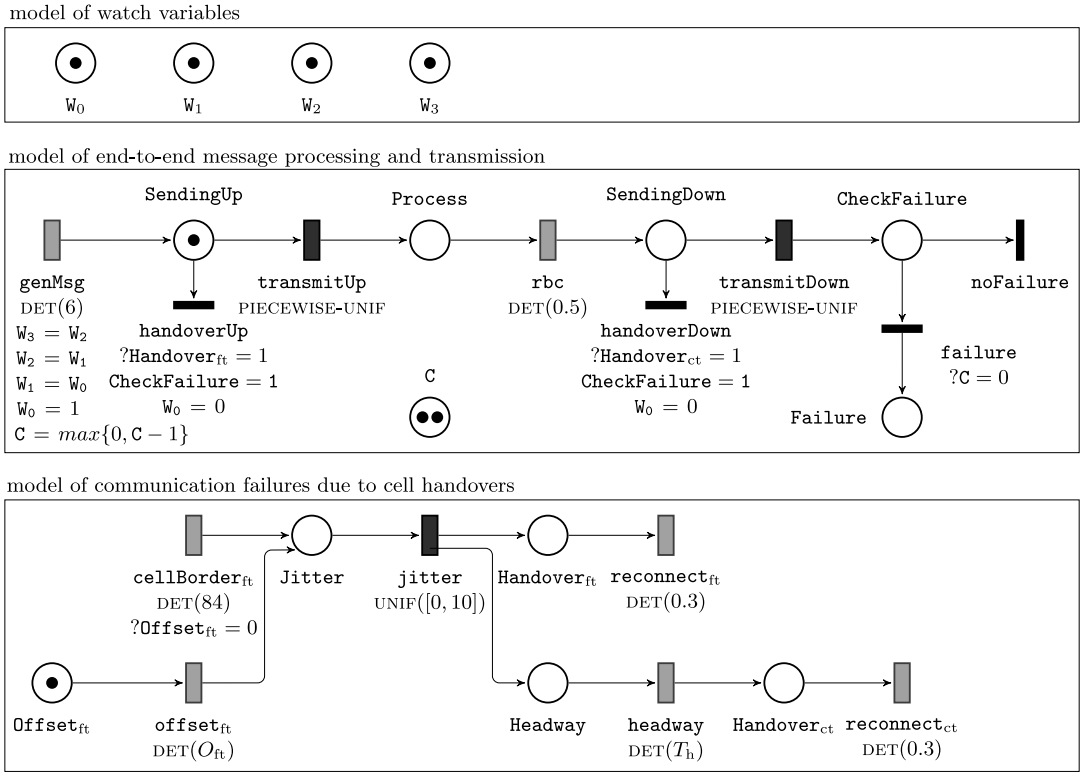

Figure 6: STPN model of the ERTMS/ETCS-L3 railway signalling case study analyzed in [13].

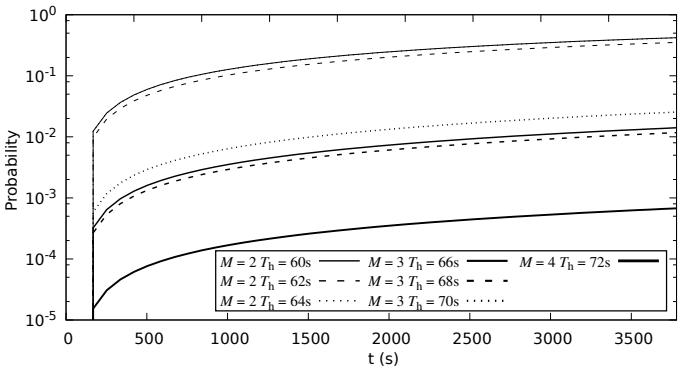

Figure 7: Railway signalling case study [13]: first-passage time distribution of a spurious emergency braking computed for different values of the number M of consecutive tolerated losses and of the headway distance T_{h} between trains.

available smart devices, such as the probability that duration is lower than a value v within a bounded $[\min, \max]$ interval, which can be fit by an expolynomial PDF of the form $f(x) = \alpha(x - \min)(\max - x)e^{-\lambda x}$, with $\alpha, \lambda \in \mathbb{R}_0^+$ such that $\int_{\min}^{\max} f(x)dx = 1$ and $\int_{\min}^{v} f(x)dx = p$.

To evaluate the network performability, the procedure model of [14] is solved through forward transient analysis (since the number of reached regenerations is low), with (limited) complexity mainly due to the number of concurrent GEN and DET transitions and on the length (i.e., number of discrete events) of behaviors where their enabling intervals overlap. On the one hand, the CDF of the completion time of the n-th phase of the procedure, with $n \in \{1, \ldots, N\}$, can be computed as a first-passage probability, i.e., as the instantaneous expected value of the reward $\texttt{EndPhase}_n$ with stop condition $\texttt{EndPhase}_n\texttt{==1}$ (where place $\texttt{EndPhase}_n$ receives a token upon completion of the n-th phase). On the other hand, since the supply pressure can be controlled independently of the network history, and a steady state is reached after any discrete change of the fluid-dynamic process, the expected value of a measure of Low-Pressure Risk (LPR) over time can be derived by combining the LPR experienced in each fluid-dynamic state (computed by fluid-dynamic analysis of the network) with transient probabilities of such states, i.e., as the instantaneous expected value of the reward $\sum_{\gamma \in \Gamma} \texttt{If}\,(e(m, \gamma), \mathrm{LPR}_\gamma, 0)$ where Γ is the set of fluid-dynamic states, LPR_γ is the LPR measure in $\gamma \in \Gamma$, and $e(m, \gamma)$ evaluates to TRUE if marking m represents fluid-dynamic state γ and to FALSE otherwise.

Thanks to the limited computational complexity of forward transient analysis of the procedure model, performability evaluation of a real gas distribution network with actual load profiles can be repeated for different values of various model parameters, notably supporting selection of the procedure start time that minimizes the overall LPR.

4.2.3 *Performability evaluation of water distribution networks*

Performability evaluation of repair procedures with non-Markovian durations and daily work schedules is addressed also in [25], in the context of water distribution networks, evaluating the expected Demand Not Served (DNS) over time, a relevant indicator of the lack of service experienced by users. Though the problem is similar to that of [14] in the applicative perspective, the underlying mathematics is much different (since behavior of pumps depends on node pressures and tank levels, and, in turn, tank levels depend

on the history of incoming fluxes and served demand), providing the opportunity to illustrate an advanced use of SIRIO, beyond the capabilities of the available engines.

The approach of [25] solves the problem by integrating quantitative evaluation of the procedure timing, specified by a model similar to that of [14], with fluid-dynamic analysis of the network. Specifically, the joint PDF f of the times when the procedure affects the network behavior is derived either by stochastic simulation or symbolic analysis of the procedure model, and fluid-dynamic analysis of the network is then repeated for each sample extracted from f. To derive the analytical form of f, the SIRIO API is extended by enriching stochastic state classes with *clock variables* (not included in the present distribution) permitting evaluation of the time spent in each network topology. In so doing, f can be derived by enumerating the transient tree of stochastic classes and by directly manipulating their PDFs through projections and linear transformations (without performing any type of analysis), with very limited computational complexity.

Experimentation on a real water distribution network points out the convenience of the analytical solution over the simulated one when a high degree of accuracy is needed in the evaluation of the expected DNS over time.

4.3 Activity Recognition

Activity Recognition (AR) aims at filling the gap between high-level operations performed by humans and low-level data collected by sensors, with relevant application in Ambient Assisted Living (AAL). This context allows testing ORIS with modeling and analysis of asynchronous phenomena, requiring a model sufficiently accurate to capture observed behaviors without overfitting, and sufficiently simple to permit evaluation at run-time after each observed event.

In [20], [11], AR is addressed by exploiting not only the type of the observed events but also the *continuous-time* duration of activities and inter-event times. To this end, an STPN model of activities and observable events is derived and enhanced according to the observed statistics, associating events with occurrence probabilities and durations with GEN distributions. In particular, in this context, sample mean and coefficient of variation of durations are fit with expolynomial distributions over $[0, \infty)$, so as to avoid false negatives that would be produced by distributions with bounded support whenever the observed duration is outside that support.

At run-time, SIRIO is used to analyze the model after each time-stamped event, deriving the set of plausible states and their likelihoods. Specifically, forward transient analysis is performed from each plausible state with local stop criterion being the occurrence of an observable event, with limited computational complexity thanks to the small depth of transient trees. Then, the probability that an activity is performed at time t conditioned to the sequence of events observed until t is derived as the instantaneous expected value of the reward $\texttt{If}(e(m,a),1,0)$, where $e(m,a)$ is a function of token counts that evaluates to TRUE if marking m represents the execution of activity a and to FALSE otherwise. By exploiting a Forward-Backward procedure, offline AR is also supported, computing the probability that an activity is performed at time t based on all the events before and after t.

Experiments performed on a AAL dataset [66] of low-level sensor data used to predict high-level human activities show that the considered continuous-time approach achieves performance measures comparable to state-of-the-art discrete-time approaches while significantly improving applicability. In fact, the approach is less prone to overfitting (only few parameters have to be to fit) and it can be similarly applied in domains where the observed phenomena have different timescales.

5 Conclusions

ORIS represents a unique solution for quantitative modeling and analysis of non-Markovian models. It includes a new graphical editor and a new Java library supporting transient and steady-state analysis of MRPs with concurrent GEN timers, or even GSMPs with concurrent GEN timers. ORIS also implements standard solution techniques for transient and steady-state analysis of CTMCs, transient analysis of MRPs under the enabling restriction, and state-space enumeration of models with nondeterministic durations.

Instantaneous or cumulative state-based rewards can be defined to compute performance measures. Through the Java API, model definition and analysis can be automated, allowing the user to carry out extensive performance studies (e.g., varying multiple parameters of the model). The software architecture, designed to implement new features of Petri models and new analysis methods, makes ORIS also a flexible research tool to evaluate novel solutions for discrete-event systems. Over the years, ORIS has been successfully used in a variety of contexts and application domains, for instance as a GUI to evaluate performability measures in railway signaling systems [18], as an API to perform sensitivity analysis of maintenance procedures in gas distribution networks [21], and also as a tool to implement a new solution technique for transient analysis of non-Markovian models [10].

The integration of probabilistic model checking techniques for regenerative stochastic systems [55] is planned for future development in ORIS.

References

[1] GSM-R Interfaces: Class 1 Requirements.

[2] M. Ajmone Marsan, G. Conte, and G. Balbo. A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems. *ACM Trans. Comput. Syst.*, 2(2):93–122, May 1984.

[3] E. G. Amparore, G. Balbo, M. Beccuti, S. Donatelli, and G. Franceschinis. *30 Years of GreatSPN*, chapter In: Principles of Performance and Reliability Modeling and Evaluation: Essays in Honor of Kishor Trivedi, pages 227–254. Springer, Cham, 2016.

[4] E. G. Amparore and S. Donatelli. MC4CSLTA: an efficient model checking tool for CSLTA. In *QEST'10*, pages 153–154, 2010.

[5] E. G. Amparore and S. Donatelli. A component-based solution for reducible Markov regenerative processes. *Perform. Eval.*, 70(6):400–422, 2013.

[6] G. Behrmann, A. David, and K. G. Larsen. A tutorial on UPPAAL. In *SFM-RT'04*, number 3185 in LNCS, pages 200–236. Springer–Verlag, September 2004.

[7] G. Behrmann, A. David, K. G. Larsen, P. Pettersson, and W. Yi. Developing UPPAAL over 15 years. *Softw. Pract. Exper.*, 41(2):133–142, Feb. 2011.

[8] B. Berthomieu and M. Diaz. Modeling and Verification of Time Dependent Systems Using Time Petri Nets. *IEEE Trans. Softw. Eng.*, 17(3):259–273, 1991.

[9] B. Berthomieu, P.-O. Ribet, and F. Vernadat. The tool TINA – construction of abstract state spaces for Petri Nets and Time Petri Nets. *International Journal of Production Research*, 42(14), 2004.

[10] M. Biagi, L. Carnevali, M. Paolieri, T. Papini, and E. Vicario. Exploiting non-deterministic analysis in the integration of transient solution techniques for Markov regenerative processes. In *QEST'17*, pages 20–35, 2017.

[11] M. Biagi, L. Carnevali, M. Paolieri, F. Patara, and E. Vicario. A continuous-time model-based approach for activity recognition in pervasive environments. *IEEE Trans. on Human-Machine Systems*, accepted for publication.

[12] M. Biagi, L. Carnevali, M. Paolieri, and E. Vicario. An introdution to the ORIS tool. In *VALUETOOLS'17*. ACM, 2017.

[13] M. Biagi, L. Carnevali, M. Paolieri, and E. Vicario. Performability evaluation of the ERTMS/ETCS - Level 3. *Transportation Research Part C: Emerging Technologies*, 82:314–336, 2017.

[14] M. Biagi, L. Carnevali, F. Tarani, and E. Vicario. Model-based quantitative evaluation of repair procedures in gas distribution networks. *ACM Trans. on Cyber-Physical Systems*, 3(2):19:1–19:26, Dec. 2018.

[15] G. S. Blair, N. Bencomo, and R. B. France. Models@run.time. *IEEE Computer*, 42(10):22–27, 2009.

[16] G. Bucci, L. Carnevali, L. Ridi, and E. Vicario. Oris: a tool for modeling, verification and evaluation of real-time systems. *Int. Journal on Softw. Tools for Techn. Transfer*, 12(5):391–403, Sept. 2010.

[17] E. Cagno, F. Caron, M. Mancini, and F. Ruggeri. Using AHP in determining the prior distributions on gas pipeline failures in a robust bayesian approach. *Reliability Engineering & System Safety*, 67(3):275–284, 2000.

[18] L. Carnevali, F. Flammini, M. Paolieri, and E. Vicario. Non-Markovian Performability Evaluation of ERTMS/ETCS Level 3. In *EPEW'15*, volume 9272 of *LNCS*, pages 47–62. Springer, 2015.

[19] L. Carnevali, L. Grassi, and E. Vicario. State-density functions over DBM domains in the analysis of non-Markovian models. *IEEE Trans. Softw. Eng.*, 35(2):178–194, Mar. 2009.

[20] L. Carnevali, C. Nugent, F. Patara, and E. Vicario. A Continuous-Time Model-Based Approach to Activity Recognition for Ambient Assisted Living. In *QEST'15*, pages 38–53. Springer, 2015.

[21] L. Carnevali, M. Paolieri, F. Tarani, and E. Vicario. Quantitative Evaluation of Availability Measures of Gas Distribution Networks. In *VALUETOOLS'13*, pages 145–154. ACM, 2013.

[22] L. Carnevali, L. Ridi, and E. Vicario. A framework for simulation and symbolic state space analysis of non-Markovian models. In *SAFECOMP'11*, volume 6894 of *LNCS*, pages 409–422. Springer, 2011.

[23] L. Carnevali, L. Ridi, and E. Vicario. Sirio: A Framework for Simulation and Symbolic State Space Analysis of non-Markovian Models. In *QEST'11*, pages 153–154, Sep. 2011.

[24] L. Carnevali, L. Ridi, and E. Vicario. A quantitative approach to input generation in real-time testing of stochastic systems. *IEEE Trans. Softw. Eng.*, 39(3):292–304, March 2013.

[25] L. Carnevali, F. Tarani, and E. Vicario. Performability evaluation of water distribution systems during maintenance procedures. *IEEE Trans. on Systems, Man and Cybernetics: Systems*, to appear.

[26] E. Çinlar. Markov renewal theory: A survey. *Management Science*, 21(7):727–752, 1975.

[27] H. Choi, V. G. Kulkarni, and K. S. Trivedi. Markov regenerative stochastic Petri nets. *Perform. Eval.*, 20(1-3):337–357, May 1994.

[28] G. Ciardo, R. German, and C. Lindemann. A characterization of the stochastic process underlying a stochastic Petri net. *IEEE Trans. Softw. Eng.*, 20(7):506–515, July 1994.

[29] G. Ciardo, R. L. Jones, III, A. S. Miner, and R. I. Siminiceanu. Logic and stochastic modeling with SMART. *Perform. Eval.*, 63(6):578–608, June 2006.

[30] C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool KRONOS. In *Hybrid systems III*. 1066, Springer, 1995.

[31] C. Dehnert, S. Junges, J.-P. Katoen, and M. Volk. A storm is coming: A modern probabilistic model checker. In *CAV'17*, pages 592–600. Springer, 2017.

[32] D. L. Dill. Timing assumptions and verification of finite-state concurrent systems. In *AVMFSS'89*, volume 407 of *LNCS*, pages 197–212. Springer, 1990.

[33] ERTMS/ETCS - System Requirements Specification.

[34] B. L. Fox and P. W. Glynn. Computing poisson probabilities. *Commun. ACM*, 31(4):440–445, 1988.

[35] E. Gafni and M. Mitzenmacher. Analysis of timing-based mutual exclusion with random times. *SIAM Journal on Computing*, 31(3):816–837, 2001.

[36] G. Gardey, D. Lime, M. Magnin, and O. Roux. Roméo: a tool for analyzing Time Petri Nets. *CAV'05*, 2005.

[37] S. Garg, A. Puliafito, M. Telek, and K. Trivedi. Analysis of preventive maintenance in transactions based software systems. *IEEE Trans. on Computers*, 47(1):96–107, Jan 1998.

[38] S. Garg, A. Puliafito, M. Telek, and K. S. Trivedi. Analysis of software rejuvenation using Markov Regenerative Stochastic Petri Net. In *ISSRE'95*, pages 180–187, 1995.

[39] R. German. Iterative analysis of Markov regenerative models. *Perform. Eval.*, 44(1-4):51–72, 2001.

[40] R. German and C. Lindemann. Analysis of stochastic Petri nets by the method of supplementary variables. *Perform. Eval.*, 20(1-3):317–335, May 1994.

[41] R. German, D. Logothetis, and K. S. Trivedi. Transient analysis of Markov regenerative stochastic Petri nets: a comparison of approaches. In *PNPM'95*, pages 103–112, 1995.

[42] L. Grunske. Specification patterns for probabilistic quality properties. In *ICSE'08*, pages 31–40. ACM, May 2008.

[43] A. Hessel, K. G. Larsen, M. Mikucionis, B. Nielsen, P. Pettersson, and A. Skou. Testing real-time systems using UPPAAL. In *Formal Methods and Testing*, pages 77–117, 2008.

[44] A. Horváth, M. Paolieri, L. Ridi, and E. Vicario. Transient analysis of non-Markovian models using stochastic state classes. *Perform. Eval.*, 69(7-8):315–335, July 2012.

[45] A. Horváth and M. Telek. PhFit: A General Phase-Type Fitting Tool. In *Computer Performance Evaluation, Modelling Techniques and Tools (TOOLS'02)*, pages 82–91, 2002.

[46] Y. Huang, C. M. R. Kintala, N. Kolettis, and N. D. Fulton. Software Rejuvenation: Analysis, Module and Applications. In *International Symposium on Fault-Tolerant Computing*, pages 381–390, 1995.

[47] J.-P. Katoen, I. S. Zapreev, E. M. Hahn, H. Hermanns, and D. N. Jansen. The ins and outs of the probabilistic model checker MRMC. *Perform. Eval.*, 68(2):90–104, Feb. 2011.

[48] S. Kounev. Performance modeling and evaluation of distributed component-based systems using queueing Petri nets. *IEEE Trans. Softw. Eng.*, 32(7):486–502, July 2006.

[49] M. Krichen and S. Tripakis. Black-box conformance testing for real-time systems. In *SPIN'04*, pages 109–126, 2004.

[50] V. Kulkarni. *Modeling and analysis of stochastic systems*. Chapman & Hall, 1995.

[51] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: verification of probabilistic real-time systems. In *CAV'11*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.

[52] L. Lamport. A Fast Mutual Exclusion Algorithm. *ACM Trans. Comput. Syst.*, 5(1):1–11, Jan. 1987.

[53] S. Martina, M. Paolieri, T. Papini, and E. Vicario. Performance Evaluation of Fischer's Protocol through Steady-State Analysis of Markov Regenerative Processes. In *MASCOTS'16*, pages 355–360. Springer, 2016.

[54] ORIS Tool. Homepage. http://www.oris-tool.org, 2019.

[55] M. Paolieri, A. Horváth, and E. Vicario. Probabilistic Model Checking of Regenerative Concurrent Systems. *IEEE Trans. Softw. Eng.*, 42(2):153–169, Feb 2016.

[56] C. A. Paterson and R. Calinescu. Observation-enhanced qos analysis of component-based systems. *IEEE Transactions on Software Engineering*, 2018.

[57] P. Reinecke, T. Krauß, and K. Wolter. Phase-Type Fitting Using HyperStar. In *EPEW'13*, pages 164–175, 2013.

[58] F. Salfner and K. Wolter. Analysis of service availability for time-triggered rejuvenation policies. *Journal of Systems and Software*, 83(9):1579 – 1590, 2010.

[59] W. H. Sanders and J. F. Meyer. Stochastic Activity Networks: Formal Definitions and Concepts. In *Lectures on Formal Methods and Performance Analysis*, volume 2090 of *LNCS*, pages 315–343. Springer, 2001.

[60] L. Sassoli and E. Vicario. Closed form derivation of state-density functions over DBM domains in the analysis of non-Markovian models. In *QEST'07*, pages 59–68. IEEE CS, 2007.

[61] SIRIO Library. https://github.com/oris-tool/sirio, 2018.

[62] W. J. Stewart. *Introduction to the Numerical Solution of Markov Chains*. Princeton University Press, 1995.

[63] M. Telek and A. Horváth. Transient analysis of Age-MRSPNs by the method of supplementary variables. *Perform. Eval.*, 45(4):205–221, Aug. 2001.

[64] K. S. Trivedi. *Probability and statistics with reliability, queuing, and computer science applications*. John Wiley and Sons, New York, 2001.

[65] K. S. Trivedi and R. Sahner. SHARPE at the Age of Twenty Two. *SIGMETRICS Perform. Eval. Rev.*, 36(4):52–57, Mar. 2009.

[66] T. van Kasteren, G. Englebienne, and B. J. A. Kröse. Activity recognition using semi-markov models on real world smart home datasets. *JAISE*, 2(3):311–325, 2010.
[67] A. van Moorsel and K. Wolter. Analysis of restart mechanisms in software systems. *IEEE Trans. Softw. Eng.*, 32(8):547–558, Aug 2006.
[68] E. Vicario. Static analysis and dynamic steering of time-dependent systems. *IEEE Trans. Softw. Eng.*, 27(8):728–748, Aug. 2001.
[69] E. Vicario, L. Sassoli, and L. Carnevali. Using stochastic state classes in quantitative evaluation of dense-time reactive systems. *IEEE Trans. Softw. Eng.*, 35(5):703–719, Sept./Oct. 2009.
[70] W. Whitt. Approximating a point process by a renewal process, I: Two basic methods. *Operations Research*, 30(1):125–147, 1982.
[71] N. Wolovick, P. R. D'Argenio, and H. Qu. Optimizing probabilities of real-time test case execution. In *ICST'09*, pages 446–455, 2009.
[72] A. Zimmermann. Modelling and Performance Evaluation with TimeNET 4.4. In *QEST'17*, pages 300–303, 2017.
[73] A. Zimmermann and G. Hommel. A train control system case study in model-based real time system desig. In *IPDPS'03*, page 118, 2003.

Marco Paolieri is a Senior Research Associate at the University of Southern California, Los Angeles, USA. He received his Ph.D. in Computer Science, Systems, and Telecommunications (2015) and his M.S. in Computer Engineering (2011) from the University of Florence, Italy. His research interests focus on stochastic modeling and quantitative evaluation of performance and reliability in concurrent and distributed systems.

Marco Biagi received the M.S. degree in Computer Engineering from the University of Florence, Italy, in 2011. He worked as a Software Engineer for five years in an IT company and is now a student in the Ph.D. program in Smart Computing at the University of Florence, Italy. His research activity mainly focuses on quantitative approaches for diagnosis, prediction, and action scheduling in partially observable systems. He is a member of the Software Technology Laboratory (STLab) of the University of Florence, Italy.

Laura Carnevali is Assistant Professor of Computer Science (with tenure track) at the School of Engineering, University of Florence, Italy. She received the M.S. degree in Computer Engineering and the Ph.D. degree in Informatics, Multimedia, and Telecommunications Engineering from the University of Florence, Italy, in 2006 and 2010, respectively. Her research is focused on correctness verification and performance evaluation of real-time systems, with focus on stochastic characterization of timed models.

Enrico Vicario (M'95) is a Full Professor of Computer Science and Head of the Information Engineering Department at the University of Florence, Italy. He holds a Ph.D. in Informatics and Telecommunications Engineering received from the University of Florence in 1994. He works in the area of Software Engineering, with focus on applications and methods of quantitative evaluations and on software architectures and methodologies.

APPENDIX: EXTENDING THE SIRIO LIBRARY

SIRIO collects libraries for analysis and simulation of TPNs and STPNs, designed to facilitate the implementation of new modeling features and new solution techniques. We illustrate this concept discussing how SIRIO can be extended to support *fickle functions* [25], which enable the representation of suspension of activities for a deterministic time. Specifically, the STPN syntax (Definition 1) is extended by associating each transition $t \in T$ with a fickle function $I(t) : \mathcal{M} \times T \to \mathbb{R}_{>0}$ which, in turn, associates each marking and (fired) transition with a real value. Moreover, in the derivation of state $s_{i+1} = \langle m_{i+1}, \vec{\tau}_{i+1} \rangle$ reached from state $s_i = \langle m_i, \vec{\tau}_i \rangle$ through the firing of transition γ_{i+1}, the STPN semantics (Definition 3) is extended by adding the deterministic value $I(t)(m_{i+1}, \gamma_{i+1})$ to the time to fire of each transition t that is persistent in state s_{i+1}. In so doing, a transition can be suspended for a deterministic amount of time upon the firing of a given transition in a given marking.

Modeling. In SIRIO, the Petri net type of the model is not encoded statically, but it is defined dynamically by the *features* associated with its basic elements, allowing agile maintainability and evolution of code. Therefore, according to the UML class diagram shown in Fig. 8, a new class `FickleFunction` is defined that implements the `TransitionFeature` interface, with a map associating pairs ⟨`Marking`, `Transition`⟩ with a `BigDecimal` value.

Evaluation. The analysis engines rely on a framework orchestrated by the class `Analyzer`, implementing a generic algorithm (Algorithm 1) for enumeration of the state-space of discrete-event systems. Specifically, a *succession evaluator* is repeatedly invoked to enumerate the children of a parent stochastic state class, and its behavior can be customized through *pre-processors* (applied *before* the successors of a stochastic state class are computed) and *post-processors* (applied *after* a stochastic state class is computed). A specific *enumeration policy* can be used to extract stochastic state classes from a queue and enumerate their successors. Moreover, a *global stop criterion* permits halting the whole state-space enumeration, while a *local stop criterion* permits terminating enumeration of the successors of specific classes.

The implementation of Algorithm 1 is customized using composition rather than inheritance, so that the implementation of new solution techniques amounts to the implementation of delegate components for `Analyzer`. As shown in Fig. 9, each instance of `Analyzer` is configured with an instance of `State`, providing the initial state for the enumeration, and with an instance of `AnalyzerComponentsFactory`, providing the objects required to determine enabled events, select the next event to analyze, compute successor states, and halt the enumeration from the current state or globally. Enumeration yields an instance of `SuccessionGraph`, which represents a labeled graph of `State` instances. To allow for heterogeneous kinds of analysis, instances of `State` support the dynamic addition of objects implementing the `StateFeature` interface.

Hence, a new class `FickleSuccessionEvaluator` is defined which implements the `SuccessionEvaluator` interface: for each enabled transition whose `FickleFunction` has the current marking and fired transition as map entry, the time to fire is shifted by the corresponding map value.

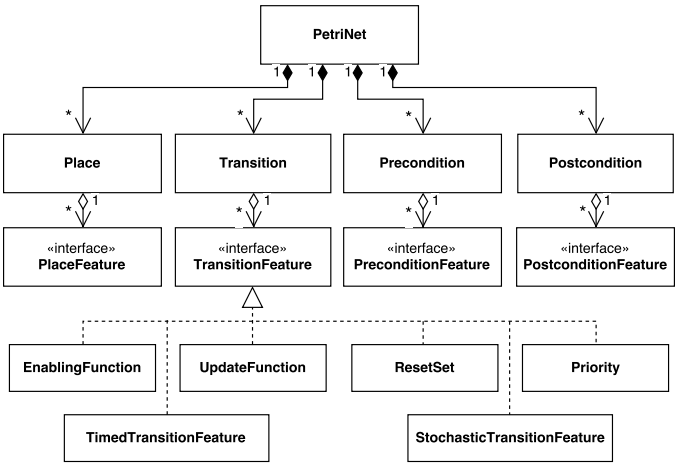

Figure 8: Extensible Petri net model of the SIRIO library.

Algorithm 1 State-space exploration from the initial class Σ_0

$(\cdot, \cdot, \Sigma_0) = \textsc{InitialSuccessionBuilder}()$
$(\cdot, \cdot, \Sigma_0) = \textsc{PostProcessor}((\cdot, \cdot, \Sigma_0))$
$Q = \{(\cdot, \cdot, \Sigma_0)\}$
$G = \varnothing$
while $Q \neq \varnothing \wedge \neg$ GlobalStop() **do**
  select and remove a succession (Σ_p, t_i, Σ) from Q
  $(\Sigma_p, t_i, \Sigma) = \textsc{PreProcessor}((\Sigma_p, t_i, \Sigma))$
  if $\textsc{NewSuccession}((\Sigma_p, t_i, \Sigma)) \wedge \neg$LocalStop() **then**
    $G = G \cup \{(\Sigma_p, t_i, \Sigma)\}$
    for each $t \in \textsc{NextEvents}(\Sigma)$ **do**
      $(\Sigma, t, \Sigma') = \textsc{SuccessionEvaluator}(\Sigma, t)$
      $(\Sigma, t, \Sigma') = \textsc{PostProcessor}((\Sigma, t, \Sigma'))$
      $Q = Q \cup \{(\Sigma, t, \Sigma')\}$
      if GlobalStop() **then**
        break
      end if
    end for
  end if
end while

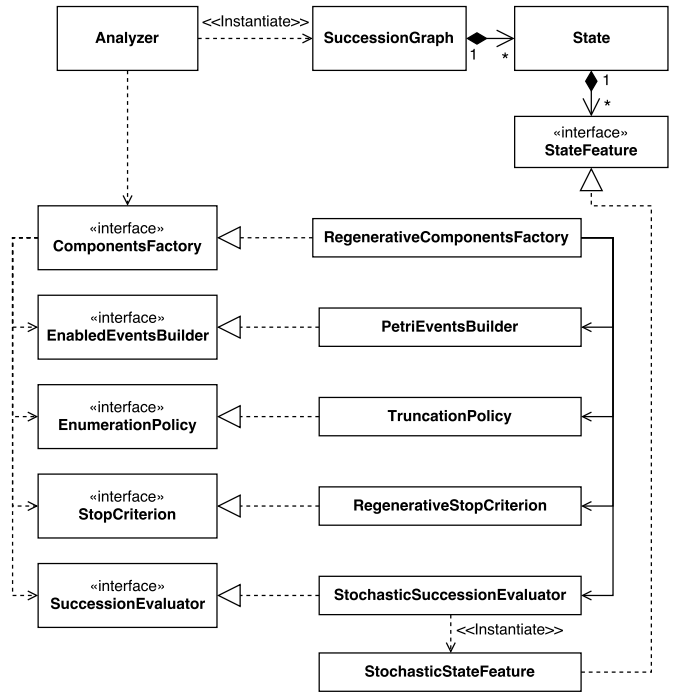

Figure 9: Analyzer framework of the SIRIO library.