The ORIS tool: app, library, and toolkit for quantitative evaluation of non-Markovian systems

Laura Carnevali
Information Engineering Dept.
Univ. of Florence
laura.carnevali@unifi.it

Marco Paolieri
Computer Science Dept.
Univ. of Southern California
paolieri@usc.edu

Enrico Vicario
Information Engineering Dept.
Univ. of Florence
enrico.vicario@unifi.it

ABSTRACT

ORIS is a tool for quantitative modeling and evaluation of concurrent systems with non-Markovian durations. It provides a Graphical User Interface (GUI) for model specification as Stochastic Time Petri Nets (STPNs), validation by interactive simulation, and evaluation by several techniques, computing instantaneous and cumulative rewards. It also provides an open-source Java Application Programming Interface (API) to automate the workflow, and it can be used as a toolkit for derivation and evaluation of STPNs in model driven engineering. As distinguishing features, ORIS implements transient and steady-state analysis of STPNs with underlying Markov Regenerative Process (MRP), and transient analysis of STPNs with underlying Generalized Semi-Markov Process (GSMP). It also implements nondeterministic analysis of Time Petri Nets (TPNs), simulation of STPNs, and solution methods for Continuous-Time Markov Chains (CTMCs) and MRPs with at most one non-exponential timer in each state. The well-engineered software architecture of ORIS supports agile implementation of new STPN features, new modeling formalisms, and new analysis methods.

Keywords

Software tools and libraries, stochastic models, quantitative evaluation, stochastic Petri nets, non-Markovian processes, Markov regenerative processes, model driven engineering.

1. INTRODUCTION

ORIS [24, 23] enables modeling of stochastic systems with multiple concurrent general (i.e., non-Markovian) timers, with bounded or unbounded support, and quantitative evaluation of their underlying stochastic processes. As shown in Fig. 1, ORIS provides a GUI for specification of STPNs [35], their validation by interactive simulation, and their transient or steady-state analysis by different methods making different assumptions on the underlying stochastic process. ORIS also includes SIRIO [29], an open-source API enabling access to the GUI functions and to additional features for symbolic manipulation of multivariate distributions and generalization of the modeling formalism and the solution techniques.

As characterizing feature, ORIS implements the *method of stochastic state classes* [35, 18], enabling quantitative evaluation of non-Markovian models underlying an MRP [21] (if a new *regeneration* is always reached with probability 1, i.e., a state satisfying the Markov condition) or a GSMP: *i*) exact transient and steady-state analysis of STPNs with underlying an MRP that always reaches a new regeneration within a bounded number of state transitions (*bounded regeneration restriction*); *ii*) exact transient analysis of STPNs with bounded number of state transitions within the time limit and approximate transient analysis of STPNs, with not restriction on the occurrence of regenerations.

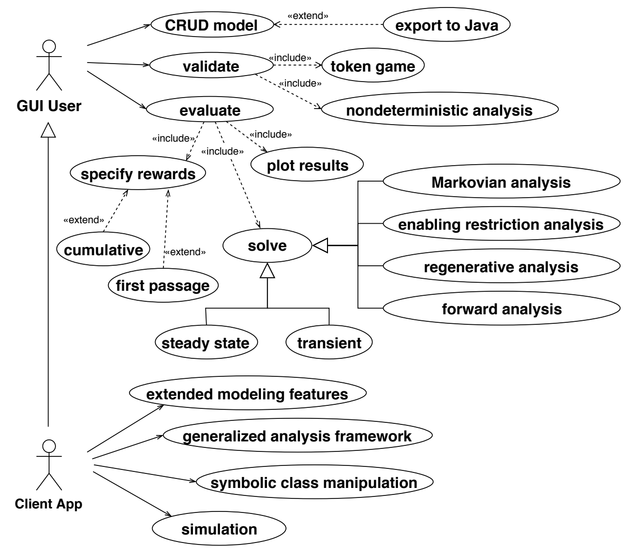


Figure 1: Use-case diagram of the functionalities provided by the GUI and API of the ORIS tool.

ORIS also offers a basic implementation of transient and steady-state analysis of STPNs with underlying CTMC [30] and transient analysis of STPNs with underlying MRP satisfying the enabling restriction [15] (i.e., at most one general timer enabled in each state), which are the focus of tools like PRISM [22], SHARPE [32], TimeNET [37], and GreatSPN [2]. Moreover, ORIS implements transient and steady-state simulation of STPNs, and nondeterministic analysis of the underlying TPNs [34] based on the enumeration of the *state class graph*, also used in qualitative verification tools like Tina [5], Romeo [12], and Uppaal [3].

ORIS has been used, as a GUI or library, to perform quantitative evaluation of stochastic models in various application domains, e.g., design and testing of real-time software [10, 25], performability evaluation of railway signalling systems [7] and of repair procedures for gas and water distribution networks [8, 11], and human activity recognition for ambient assisted living [9, 6]. The flexible and extensible software architecture enables the agile implementation of new model features and analysis algorithms, and facilitates the integration of the SIRIO library into custom software tools and toolchains, supporting model driven engineering approaches where STPNs are instantiated from domain metamodels and analyzed in automated manner for many parameter values. Moreover, ORIS has been used as a support for teaching quantitative modeling and evaluation of non-Markovian systems in master courses of the University of Florence, e.g., "Quantitative Evaluation of Stochastic Models".

The ORIS tool is freely available at oris-tool.org, and a tutorial is available at oris-tool.org/tutorial. The source code of the SIRIO library is available at github.com/oris-tool/sirio under the AGPL licence, and a ready-to-use project is available at github.com/oris-tool/sirio-examples.

This paper is a short version of [24][1], where we recall how to build and analyze STPNs (Sections 2 and 3). Then, we discuss how to use ORIS as a toolkit to support model driven engineering from domain metamodels to runtime analysis (Section 4), through an example from the context of smart transportation. Finally, we draw our conclusions (Section 5).

2. MODEL SPECIFICATION

2.1 Stochastic Time Petri Nets (STPNs)

STPNs [35] model concurrent systems with stochastic temporal parameters. As shown in Fig. 2, they consist of: *transitions* (depicted as vertical bars) modeling activities; *tokens* within *places* (depicted as dots inside circles) modeling the system logical state; and *directed arcs* from *input* places to transitions and from transitions to *output* places (depicted as directed arrows), modeling token moves at the execution of activities. A *marking* assigns a natural number of tokens to each place. A transition is enabled if all its input places contain at least one token; upon firing, it removes one token from each input place and adds one to each output place.

A transition t is termed immediate (IMM) or deterministic (DET) if its time to fire τ is zero or a positive value, respectively. Otherwise, t is termed exponential (EXP) or general (GEN) if τ is a continuous random variable with EXP or GEN Probability Density Function (PDF), respectively.

As stochastic reward nets [31] and stochastic activity networks [28], STPNs support *enabling functions* restricting the enabling of a transition through constraints on token counts. STPNs also support *update functions* specifying additional updates of token counts after a firing, *reset sets* forcing the restart of selected transitions, and *priorities* imposed among IMM or DET transitions. If omitted, default feature values are an always-true enabling function, an identity update function, an empty reset set, weight 1, and priority 0.

Arc cardinalities larger than 1 could be easily introduced by 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 supported by SIRIO, arc cardinalities were not included in the ORIS GUI to reduce model clutter, in contrast with explicit features provided by other tools [37]; instead, arc cardinalities can be modeled in ORIS through enabling and update functions.

[1]When referring to ORIS and SIRIO, please cite [24].

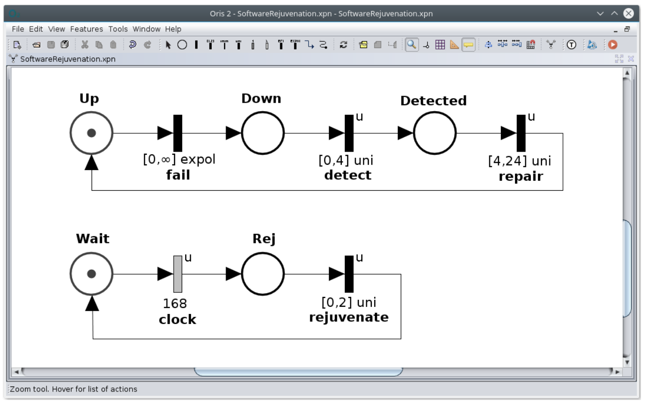

Oris 2 - SoftwareRejuvenation.xpn - SoftwareRejuvenation.xpn
File Edit View Features Tools Window Help
SoftwareRejuvenation.xpn
Up
Down
Detected
u
u
[0,∞] expol
fail
[0,4] uni
detect
[4,24] uni
repair
Wait
Rej
u
u
168
clock
[0,2] uni
rejuvenate
Zoom tool. Hover for list of actions

Figure 2: ORIS GUI: STPN model of software rejuvenation. IMM, DET, EXP, GEN transitions are represented by thin black bars, thick gray bars, thick white bars, black thick bars, respectively, labeled with `e`, `u`, `r` if having non-default value of enabling function, update function, reset set, respectively.

2.2 Structure of concurrency

We recall the model specification workflow using the example of *software rejuvenation* [33, 27, 13, 20] of Fig. 2, where a software system inspired by [14] is restarted periodically to prevent failures due to *software aging* and reduce unavailability intervals. First, feasible behaviours are identified by the concurrency structure and qualitative timing constraints. Specifically, the basic structure of concurrency is represented by the underlying Petri Net (PN) (i.e., places, transitions, enabling and update functions, priorities), capturing the concurrency between the aging process of the software system (transitions `fail`, `detect`, and `repair` model the time required by software aging, failure detection, and unplanned repair, respectively) and the rejuvenation mechanism (transitions `clock` and `rejuvenate` model the time between two consecutive rejuvenations and the rejuvenation time, respectively). Update functions model the interactions between the software system and the rejuvenation mechanism: the update function of transition `clock` flushes places `Up`, `Down`, and `Detected` to represent system switch-off during rejuvenation; the update function of transition `rejuvenate` assigns a token to place `Up` to model system restart after rejuvenation; the update function of transition `detect` flushes place `Wait` to account for disabling of rejuvenation during unplanned repair; and, the update function of transition `repair` adds a token to place `Wait` to model reschedule of rejuvenation.

Firm timing constraints and reset sets further restrict possible behaviors by extending the PN into a TPN [4], determining the set of timed firing sequences, i.e., sequences of transition firings, associated with the time between each pair of consecutive firings. In Fig. 2, we assume that the system specification requires the failure detection time to be lower than 4 h, the repair time to be between 4 and 24 h, the rejuvenation time to be lower than 2 h, and the rejuvenation period to be 168 h (7 days). Thus, `detect`, `repair` and `rejuvenate` have support $[0, 4]$ h, $[4, 24]$ h, and $[0, 2]$ h, respectively, while clock has a deterministic value of 168 h. Conversely, we assume that the failure time is unbounded, thus `fail` has support $[0, \infty)$ h. The initial marking is `Up Wait`, i.e., system up and rejuvenation timer just started.

2.3 Stochastic parameters

Feasible behaviours are associated with a measure of probability by setting distributions and weights of transitions, identifying an STPN that casts the timed firing sequences of the underlying TPN into a probability space [25]. ORIS supports *expolynomial* PDFs [32], i.e., products of exponentials and polynomials, on bounded or unbounded supports, with analytical representation over its domain or piecewise-defined over multiple sub-domains, with EBNF syntax:

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

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

Expolynomials represent common PDFs (e.g., Erlang, uniform) and enable approaches to fit data (e.g., moments [36], shape [19, 26]) obtained in different manners (e.g., estimated from measurements, synthetically generated). In Fig. 2, we assume the failure time was repeatedly measured, with values lower than $x_1 = 72\,\text{h}$ (3 days), $x_2 = 144\,\text{h}$ (6 days), and $x_3 = 216\,\text{h}$ (9 days) with frequency $p_1 = 0.001$, $p_2 = 0.006$, and $p_3 = 0.016$, respectively, and, larger than x_3 with frequency $p_4 = 0.984$ and mean value $672\,\text{h}$ (28 days). These measurements can be modeled by the PDF of transition `fail`, piecewise-defined 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 $x_0 = p_0 = 0$; *ii)* a shifted exponential PDF $f(x) = p_3 \alpha \exp(-\lambda x)$ for $x \in [x_3, \infty)$ with rate $\lambda = 0.002193$, shift $x_3 = 216\,\text{h}$, and $\alpha = 0.003522$ with mean value of $672\,\text{h}$ when $x > x_3$). Other, more complex, expolynomial PDFs could be used. Conversely, we assume no measurements for the duration of failure detection, repair, and rejuvenation, associating transitions `detect`, `repair`, and `rejuvenate` with uniform PDF.

Weights are used to select one of the enabled IMM transitions or DET transitions with the same value, modeling discrete probabilistic choices depending on the logical state (e.g., probability of message losses depending on the channel conditions). Weights are defined as expressions of token counts in the current marking, e.g., `2.0*p1`. Rates of EXP transitions can also be functions of the current marking, modeling durations depending on the logical state (e.g., service rates depending on the number of available servers).

2.4 Model validation

The model is validated by interactive simulation to achieve confidence about its correspondence with the modeling aim, exploring the state space either manually, by selecting the next transition to fire, or automatically, by specifying a number of transition firings or a stop condition (i.e., function of the current marking). To support inspection, the firing probability and the range of firing times of transitions are evaluated by the method of stochastic state classes [18].

3. MODEL EVALUATION

3.1 Rewards and stop conditions

Rewards and *stop conditions* enable the evaluation of 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, supporting the assessment of non-functional requirements of stochastic systems.

Rewards. ORIS supports 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 e combining constants and token counts so as to define a real-valued function over the set of markings, with EBNF syntax:

$$\begin{aligned} c &:= \textit{place id} \mid \textit{constant} \\ e &:= c \mid \texttt{(}\,e\,\texttt{)} \mid e\,\texttt{+}\,e \mid e\,\texttt{-}\,e \mid e\,\texttt{/}\,e \mid e\,\texttt{*}\,e \mid e\,\texttt{\^{}}\,e \mid e\,\texttt{==}\,e \\ &\mid e\,\texttt{!=}\,e \mid e\,\texttt{>}\,e \mid e\,\texttt{>=}\,e \mid e\,\texttt{<}\,e \mid e\,\texttt{<=}\,e \mid e\,\texttt{!=}\,e \mid e\,\texttt{\&\&}\,e \\ &\mid e\,\texttt{||}\,e \mid \texttt{!}\,e \mid \texttt{If(}\,e\,\texttt{,}\,e\,\texttt{,}\,e\,\texttt{)} \mid \texttt{min(}\,e\,\texttt{,}\,e\,\texttt{)} \mid \texttt{max(}\,e\,\texttt{,}\,e\,\texttt{)} \end{aligned}$$

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 $\texttt{If}(e_1, e_2, e_3)$ is a ternary conditional if operator as in Java (i.e., it evaluates to the value of e_2 or e_3 depending on whether e_1 evaluates to TRUE or FALSE, respectively). In particular, $\texttt{If}(\phi, 1, 0)$ returns the probability of ϕ, i.e., the measure of probability of the set of behaviors where the marking satisfies the (state) property ϕ, e.g., if $\texttt{p}_1$ and $\texttt{p}_2$ are names of two places, the reward expression $\texttt{If}(\texttt{p}_1 + \texttt{p}_2 > 0, 1, 0)$ evaluates the probability of states where at least one token is contained in $\texttt{p}_1$ or $\texttt{p}_2$. Since e_1 and e_2 are expressions, the ternary operator also permits more complex forms implementing an if-then-else logic, with possible nesting. A reward can also be defined as a simple expression e, which is a shorthand of the form $\texttt{If}(\text{TRUE}, e, 0)$ and evaluates the expected value of e, e.g., the reward expression $\texttt{p}_1 + \texttt{p}_2$ evaluates the expected value of the sum of tokens in $\texttt{p}_1$ and $\texttt{p}_2$.

ORIS supports the evaluation of the expected value of rewards during the transient evolution of the stochastic process and at steady-state. Let $p_{ij}(t) = P\{M(t) = j \mid X_0 = i\}$ and $\overline{p}_{ij} = \lim_{t\to\infty} p_{ij}(t)$ be, respectively, the transient and steady-state probabilities for each initial regeneration $i \in \mathcal{R}$ (initial marking with times to fire sampled independently) and each marking $j \in \mathcal{M}$. 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 $C_r^i(t) = \int_0^t I_r(t)\, dt$ up to t.

Rewards can be used to evaluate a variety of non-functional requirements, e.g., for the model of Fig. 2, a measure of *system availability* can be expressed as $r(m) = \texttt{Up}$, which returns the probability to be in a correctly functioning state.

Stop Conditions. *Stop conditions* turn states that satisfy a Boolean predicate into absorbing states where all transitions are disabled and the system sojourns indefinitely, enabling the evaluation of quantitative measures where only a subset of execution paths is of interest. For example, *system reliability* (i.e., the "probability that the system will continuously perform its intended function during a specified period of time $[0, T]$") can be computed using a stop condition that evaluates to TRUE when the system is down, i.e., `Up==0`. Then, the instantaneous expected value of the reward $r(m) = \texttt{Up}$ at time T excludes execution paths where the system is up at T but has visited a down state before.

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

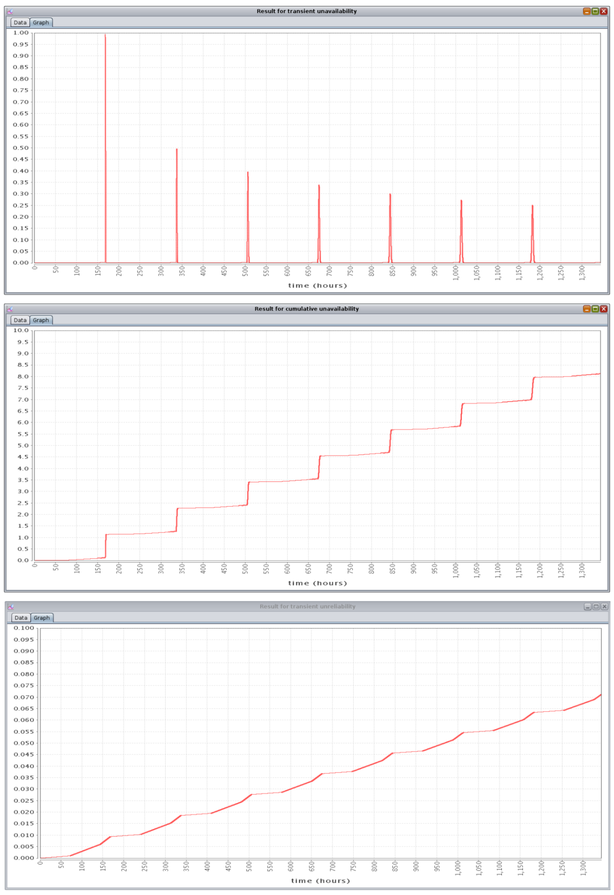


Figure 3: ORIS GUI: for the STPN model of Fig. 2, plots show transient rewards providing the transient unavailability (top), cumulative unavailability (center), and transient unreliability (bottom).

Stop conditions enable evaluation of reach-avoid objectives equivalent to a *bounded until operator* $\phi_1 \mathcal{U}^{[0,t]} \phi_2$ of probabilistic model checking [25], which specifies the set of 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 `!`ϕ_1 `||` ϕ_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 t traversing only safe states). Probabilistic until and *probabilistic existence* TRUE $\mathcal{U}^{[0,t]} \phi_2$ are the most frequent specification patterns for quality and dependability requirements of software-intensive systems in domains like automotive systems and railway signalling [17].

Example. The model of Fig. 2 is analyzed by regenerative transient analysis with time limit $t_{max} = 1344$ h (8 weeks) and step size $k = 0.005$ h, so that probabilities are evaluated for all $t = 0, k, 2k, \ldots, \lfloor \frac{t_{max}}{k} \rfloor k$. We evaluate the *transient unavailability*, i.e., the probability that the system is not working at time t, by computing the instantaneous reward `Down>0||Detected>0||Rej>0`, with initial marking `Up Wait`. We also evaluate the *cumulative unavailability*, i.e., the expected outage time within $[0, t]$. As shown in Fig. 3, unavailability is very high (nearly 0.993) at time 168 h, since this is the schedule time of the first rejuvenation and the probability that the system fails sooner 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.

We also compute the *transient unreliability*, i.e., transient probability that the system has failed at least once by time t, by computing the instantaneous reward `Down>0` with stop condition `Down>0` and initial marking `Up Wait`. As shown in Fig. 3, system reliability is low: it could be improved with more frequent rejuvenations, though reducing availability.

3.2 Analysis engines

ORIS provides a suite of analysis engines (see Fig. 1) implementing different solution methods while leveraging the same input format for rewards and stop conditions. Each 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. Complexity factors of each engine are extensively discussed in [24].

Markovian Engine. It implements standard methods for transient and steady-state analysis of CTMCs [30], requiring STPNs with only EXP and IMM transitions, also known as Generalized Stochastic Petri Nets (GSPNs) [1]. To overcome this limitation, each GEN transition can be replaced (prior to analysis) with the sequence of EXP transitions obtained by a phase-type approximation, e.g., through PhFit [19].

Enabling Restriction Engine. It implements transient analysis for MRPs under enabling restriction [16], requiring STPNs with at most one GEN transition enabled in each state. It 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 the MRP, which are used to solve a system of Volterra integral equations [21].

Regenerative Engine. It implements transient and steady-state analysis of MRPs with multiple GEN transitions enabled in each state [18]. Steady-state analysis requires the bounded regeneration restriction, which can be checked for an STPN by the (terminating) algorithm for nondeterministic analysis of the underlying TPN. Transient analysis can lift this restriction by allowing an error in the enumeration of MRP subgraphs [18]. The implementation leverages enumeration of stochastic state classes, which encode (symbolically) the joint PDFs of transition timers after each firing.

Forward Engine. It provides transient analysis of STPNs without restrictions on the occurrence of regenerations [18]. The implementation enumerates a single graph of stochastic state classes until the earliest firing times of transitions along each sequence surpass the time bound. To reduce the number of classes, the user can specify a truncation error, comprising the guaranteed error bound between approximate and exact probabilities. A truncation error is required if the STPN allows cycles of transitions firing in zero time [18].

Nondeterministic Engine. It implements nondeterministic analysis of the state space of STPNs [34]. The implementation encodes the dense set of timed states reached by an STPN 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.

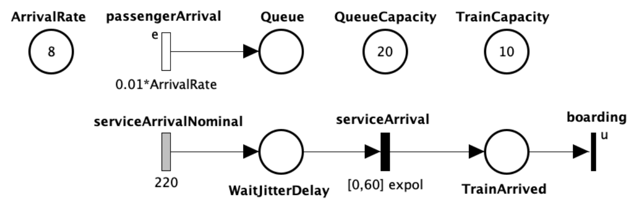


Figure 4: STPN model of a passenger queue.

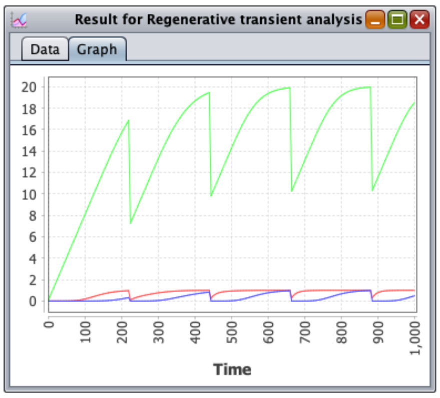
4. MODEL-DRIVEN ENGINEERING

During early design, as well as implementation and integration stages, stochastic models provide a valuable means to assess non-functional requirements of a system through the evaluation of quantitative performance metrics (e.g., throughput, waiting time, rejection rate) and dependability attributes (e.g., availability, reliability, maintainability, security). Advancements in tools that generate and evaluate these models have enabled model-driven engineering of complex and cyberphysical systems ranging from real-time and self-adapting software components, to critical infrastructures for telecommunication, transportation, or power/water/gas distribution.

Notably, while most existing approaches are limited to stochastic models governed by EXP random variables, ORIS supports models with deterministic timers (e.g., timeouts) and non-EXP durations (e.g., delays in train networks). At the same time, its Java library (SIRIO) and toolkit facilitate the development of domain-specific tools exploring different system designs and parameters. To illustrate this approach, we consider the case study of passenger queues at tram stops.

Domain Model and System Requirements. The transportation network includes *lines* and *stations*: trains leave from the initial station of a line at regular time intervals, arriving at each station after a constant delay (the nominal time to travel from the initial station), plus a random *jitter*; the time between consecutive arrivals of passengers to a station is also a random variable. A global admission policy controls the number of passengers allowed to board the train, so that passengers with similar waiting times can be admitted at the following stations; and, if the queue is longer than a given threshold, passengers abandon the station. We evaluate the number of passengers waiting at a station, the probability that at least some passenger will be denied boarding, and the probability that a newly arrived passenger will abandon the station, which might be used to support early evaluation of the impact of social distancing measures during a pandemic.

STPN Model. To model the passenger queue at a station, we define the model illustrated in Fig. 4: passengers arrive with exponential interarrival time (transition `passengerArrival`) with rate equal to 0.01 times the number of tokens in place `Arrival`; trains arrive after a deterministic delay equal to 220 (transition `serviceArrival`) plus a jitter distributed over $[0, 60]$ according to either a uniform PDF, or the PDF $f(x) = C[3\exp(-x/10) + (x/10)\exp(-x/10)]$ where $C = 40-100\exp(-6)$. The number of passengers allowed boarding is equal to the number of tokens in place `TrainCapacity` (i.e., transition `boarding` has the update function "`Queue = max(0, Queue-TrainCapacity)`"). When `QueueCapacity` passengers are already waiting, newly arrived passengers abandon the station (i.e., `passengerArrival` has enabling function "`Queue < QueueCapacity`").



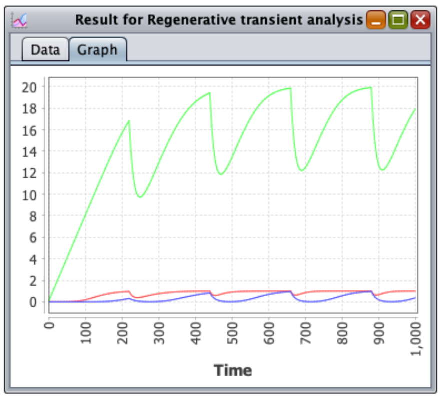


Figure 5: Queue size (green), probability of boarding denial (red), and queue abandonment (blue) for uniform (top) and expolynomial jitter (bottom).

Metrics Evaluation. The metrics of interest can be evaluated at each time t using regenerative transient analysis and rewards "`Queue`" (expected number of passengers waiting for a train), "`If(Queue>TrainCapacity,1,0)`" (prob. of at least one boarding denial upon train arrival), and "`If(Queue==QueueCapacity,1,0)`" (prob. of queue abandonment upon passenger arrival). We repeat the evaluation of these transient rewards for our two choices of jitter PDF, using a time step $\Delta t = 5$ for $t \in [0, 1000]$. As shown in Fig. 5, all metrics drop sharply near the expected arrival times of the trains: the expected number of waiting passengers oscillates between 10 and 20, while the probability of boarding denials (red) is higher than that of queue abandonment (blue). Using steady-state regenerative analysis, we can evaluate the value of these metrics at steady-state: respectively, 16.8, 0.94 and 0.43 for both jitter PDFs (uniform and the expolynomial). Each analysis completes in under 2 minutes.

5. CONCLUSIONS

This paper illustrates the usual modeling and evaluation workflow in ORIS. Specifically, ORIS can be used as a GUI, supporting quick development of a model and validation of its operation by interactive simulation or evaluation of quantitative metrics (see Sections 2 and 3). It can be used also as a toolkit, supporting export of the model from the GUI as Java code, using the SIRIO API, allowing the user to introduce quantitative evaluation of parametric non-Markovian models into larger software projects (see Section 4).

6. REFERENCES

[1] 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.

[2] 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.

[3] 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.

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

[5] 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.

[6] 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*, 49(4):293–303, 2019.

[7] 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.

[8] 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.

[9] 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.

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

[11] 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.

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

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

[14] 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.

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

[16] 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.

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

[18] 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.

[19] 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.

[20] 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.

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

[22] 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.

[23] ORIS. Homepage. http://www.oris-tool.org, 2021.

[24] M. Paolieri, M. Biagi, L. Carnevali, and E. Vicario. The ORIS Tool: Quantitative Evaluation of Non-Markovian Systems. *IEEE Trans. on Soft. Eng.*, 47:1211–1225, 2021.

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

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

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

[28] W. H. Sanders and J. F. Meyer. Stochastic activity networks: formal definitions and concepts. In *School Europ. Educ. Forum*, pages 315–343. Springer, 2000.

[29] SIRIO. https://github.com/oris-tool/sirio, 2021.

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

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

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

[33] A. van Moorsel and K. Wolter. Analysis of restart mechanisms in software systems. *IEEE TSE*, 32(8):547–558, Aug 2006.

[34] E. Vicario. Static analysis and dynamic steering of time-dependent systems. *IEEE TSE*, 27(8):728–748, Aug. 2001.

[35] E. Vicario, L. Sassoli, and L. Carnevali. Using stochastic state classes in quantitative evaluation of dense-time reactive systems. *IEEE TSE*, 35(5):703–719, Sept./Oct. 2009.

[36] W. Whitt. Approximating a point process by a renewal process, I: Two basic methods. *Operations Research*, 30(1):125–147, 1982.

[37] A. Zimmermann. Modelling and Performance Evaluation with TimeNET 4.4. In *QEST'17*, pages 300–303, 2017.