An introduction to the ORIS tool

Marco Biagi, Laura Carnevali, Enrico Vicario
University of Florence, Italy
firstname.lastname@unifi.it

Marco Paolieri
University of Southern California
paolieri@usc.edu

ABSTRACT

ORIS provides a graphical interface to draw Petri nets, analysis engines for different classes of underlying stochastic process, and visualization of reward-based metrics. It also includes a Java API for model definition and analysis, which can be used to carry out parametric performance studies. ORIS implements methods for steady-state and transient analysis of Semi-Markov Processes (SMPs), Markov Regenerative Processes (MRPs), Generalized Semi-Markov Processes (GSMPs), and Continuous-Time Markov Chains (CTMCs).

KEYWORDS

Tools, Stochastic Petri Nets, Markov Regenerative Processes, Non-Markovian Processes, Transient Analysis, Steady-State Probabilities

ACM Reference Format:
Marco Biagi, Laura Carnevali, Enrico Vicario and Marco Paolieri. 2017. An introduction to the ORIS tool. In *VALUETOOLS 2017: 11th EAI International Conference on Performance Evaluation Methodologies and Tools, December 5–7, 2017, Venice, Italy*. ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3150928.3158361

1 INTRODUCTION

ORIS is a software tool for the modeling and evaluation of stochastic systems governed by timers (e.g., interarrival or service times, failure times, repair times, timeouts) with general probability density functions (PDFs). It provides a graphical user interface to model systems using stochastic Petri nets, a popular formalism where the state is represented by *tokens* contained inside *places*, and *transitions* (depicted as bars) move tokens between places when their timers expire. Notably, *ORIS supports models where multiple timers with general PDF can be concurrently enabled in each state.*

Over the past 10 years, this challenging goal has been the subject of several works, starting with foundational results on the integration of general PDFs over DBM zones [4, 15], and then addressing different metrics for specific classes of underlying stochastic process: transient analysis of GSMPs [9] and MRPs [8], steady-state analysis of MRPs [10], probabilistic model checking of until operators in GSMPs [9] and MRPs [11] (SMPs are a subclass of MRPs). Recent work leveraged state-space analysis [14] to detect the class of the underlying process and apply the most efficient technique, or even different techniques to separate components of the state space [2] (also implementing well-established methods for transient analysis of CTMCs and MRPs with at most one general timer [7]).

VALUETOOLS 2017, December 5–7, 2017, Venice, Italy

ACM ISBN 978-1-4503-6346-4/17/12.
https://doi.org/10.1145/3150928.3158361

The flexibility of models with multiple general timers allowed the application of ORIS to the analysis of performance and reliability in many domains, including railway signaling [3], power grids [1], gas distributions networks [6] and phased-mission systems [5].

2 OVERVIEW

2.1 Models

ORIS adopts Stochastic Time Petri Nets (STPNs) as a graphical formalism to specify stochastic systems. STPNs include all the standard building blocks of stochastic Petri nets: places contain tokens that control the enabling of transitions through input and inhibitor arcs [15]. When a transition becomes enabled, it samples a time to fire according to a given PDF; the transition with minimum time to fire is the next discrete event: it "fires" by removing a token from each input place and adding a token to each output place. To improve modeling convenience, STPNs introduce *enabling functions* that control the enabling of transitions (e.g., "$\texttt{place}_1 + \texttt{place}_2 > 3$", as in stochastic activity networks and stochastic reward nets [12]), *update functions* to express complex token moves at the firing of a transition (e.g., $\mathsf{p}_1 \leftarrow \mathsf{p}_1 * \mathsf{p}_2 + \mathsf{p}_3$), and *reset sets* to force some transitions (if enabled) to resample their times to fire.

PDFs of transitions can be exponential, deterministic, expolynomial or immediate (drawn as thick white, thick gray, thick black, or thin black bars, respectively). *Expolynomials* [13] are a general class of PDFs obtained from products of exponentials and polynomials, on bounded or unbounded supports; they include uniform and Erlang PDFs as special cases. To resolve ties between immediate (i.e., with zero duration) or deterministic transitions with minimum but equal time to fire, STPNs include *priorities* (only the transitions with highest priority are considered) and *weights* (transitions are selected with probability proportional to their weights).

2.2 Metrics

A marking $m\colon P \to \mathbb{N}_{\geqslant 0}$ assigning a token count $m(p)$ to each place $p \in P$ defines the discrete state of the STPN at a specific time. The firing of transitions governs the evolution of the marking: tokens are added or removed from places, causing the enabling (or disabling) of other transitions and the sampling of their times to fire. The marking of an STPN over time is a stochastic process $\{m(t), t \geqslant 0\}$, which we require to have finite state space M. Quantitative metrics are defined from transient and steady-state probabilities, i.e., from $p_i(t) = P(\{m(t) = i\})$ and from $\overline{p}_i = \lim_{t\to\infty} p_i(t)$ for each $i \in M$.

The user can define a *reward* $r\colon M \to \mathbb{R}$, i.e., a real-valued function of markings such as "$(\mathsf{p}_1+\mathsf{p}_2)/2$" that is evaluated by ORIS (substituting place names with the number of contained tokens) to compute the instantaneous expected reward $I_r(t) = \sum_{i\in M} r(i)\, p_i(t)$ at each time t, its steady-state value $\overline{I}_r = \lim_{t\to\infty} I_r(t) = \sum_{i\in M} r(i)\, \overline{p}_i$ or its cumulative value over time $C_r(t) = \int_0^t I_r(t)\, dt$.

2.3 Analysis Engines

The software architecture of ORIS decouples the graphical editor from the underlying analysis engines. Given the many variants of Petri net features (enabling/update functions, priorities, weights, reset sets), ORIS was developed with extensibility in mind: new features can be defined by implementing specific interfaces, so that they can be introduced in the graphical editor and made available to the analysis engines. In turn, analysis engines implement a specific interface that allows them to cooperate with the graphical interface, i.e., to collect analysis options from the user, to start/stop analysis runs, to record and display analysis logs, and to show time series and tabular results. The available analysis engines include:

Non-deterministic Analysis, to produce a compact representation of the dense set of timed states that can be reached by the model. The state space is displayed as a directed graph, where edges represent transition firings while nodes are *state classes* [14] comprising a marking and a DBM zone of timer values. This analysis is useful to debug STPN models and ensure that their state space M is finite.

Transient and Regenerative Analysis, to compute transient probabilities in GSMPs and MRPs, respectively. These methods evaluate trees where edges are labeled with transitions and their firing probabilities, while nodes are *stochastic state classes* [8] comprising a marking, the PDF of timers, and their support (a DBM zone). For a given time limit T, the enumeration proceeds until the tree covers the transition firings of the STPN by time T with probability greater than $1-\epsilon$, where $\epsilon > 0$ is an *error* term. While standard transient analysis enumerates a single, very large tree of events, regenerative analysis avoids the enumeration of repeated subtrees rooted in the same *regeneration point* (where all general timers are reset or have been enabled for a deterministic time). A *time step* is used to select equispaced time points where transient probabilities are evaluated (directly or by solving Markov renewal equations).

Regenerative Steady-State Analysis, to compute steady-state probabilities in MRPs (and thus SMPs and CTMCs) with irreducible state space. This method uses trees of stochastic state classes between regeneration points to compute steady-state probabilities of markings: expected sojourn times in each tree are combined with the steady-state probability of regenerations at their roots [10]. As for transient analysis, this method can be applied to STPNs allowing multiple general timers enabled in each state.

Transient Analysis under Enabling Restriction, to compute transient probabilities in MRPs that allow at most one general transition enabled in each state [7].

Uniformization, to compute transient probabilities of CTMCs. This method implements uniformization using Fox and Glynn's algorithm to compute Poisson probabilities [7].

Engines support instantaneous (transient or steady-state) and cumulative (transient) rewards. In addition, the user can specify a *stop condition*, i.e., a Boolean predicate on markings such as `(error == 1) || (goal == 1)`, that is used to halt the STPN. This feature can be used to compute first-passage probabilities [8] or reach-avoid objectives equivalent to bounded until operators [11].

2.4 Graphical User Interface and Java API

ORIS provides a Graphical User Interface (GUI) to edit STPN models with the aid of features common in CAD tools, such as zoom, undo/redo, and aligning elements to a fixed grid, with each other (horizontally or vertically), or so as to evenly space them. STPNs models can be saved as XML files and later reopened in the editor, or exported as SVG images. To facilitate the use of the Java API, the GUI editor can export an STPN as "Java code": this function generates a Java file (e.g., `Fischer.java`) that contains the statements required to build the STPN through the ORIS API, such as:

```
public class Fischer {
  public static void build(PetriNet pn, Marking m) {
    Place id = pn.addPlace("id");
    Place writing1 = pn.addPlace("writing1");
    Place waiting1 = pn.addPlace("waiting1");
    Transition write1 = pn.addTransition("write1");
    pn.addPrecondition(writing1, write1);
    pn.addPostcondition(write1, waiting1);
    write1.addFeature(
      StochasticTransitionFeature.newUniformInstance(
        new OmegaBigDecimal("0"),
        new OmegaBigDecimal("1")));
    write1.addFeature(new PostUpdater("id=1", pn));
    m.addTokens(writing1, 1); // (rest omitted)
  }
}
```

Once a model is built using the Java API, it can be analyzed by invoking the available engines and the results (e.g., steady-state distributions, transient time-series, or state class graphs) can be saved to file or displayed using the same functions available in the GUI editor, such as plots and tables. While building and analyzing models in the GUI editor can be very useful to quickly iterate over different model designs and analyze their behavior, the Java API is crucial to carry out extensive parametric studies where the stochastic parameters are varied, or different policies are compared.

3 A CASE STUDY: FISCHER'S PROTOCOL

ORIS is available at `http://www.oris-tool.org` with a tutorial. We illustrate the use of the tool on Fischer's protocol [10, 11], which ensures mutual exclusion using atomic read and write operations on a shared communication variable, and waiting periods before entering the critical section. Fig. 1 illustrates an STPN of the protocol where n processes $P_1, P_2, \ldots, P_n$ access a critical section using the shared variable `id` taking values $0, 1, \ldots, n$. When `id` $= 0$, a process P_i that becomes ready (i.e., after transition $\texttt{arrival}_i$ fires and places a token in $\texttt{ready}_i$) can attempt to access the critical section, performing the write operation $\texttt{id} \leftarrow i$, waiting for a time not lower than the maximum write time of other processes, and then reading `id` again: if $\texttt{id} = i$, it can access the critical section and write $\texttt{id} \leftarrow 0$ on exit; otherwise ($\texttt{id} \neq i$) it must wait until $\texttt{id} = 0$ to attempt again. Mutual exclusion is guaranteed by the fact that the waiting time $\texttt{wait}_i$ is greater than the maximum writing time $\texttt{write}_i$ of other processes, so that concurrent writes to `id` are detected (by transition $\texttt{readOther}_i$) and only the last process that finishes its write can enter the critical section (with $\texttt{readSelf}_i$).

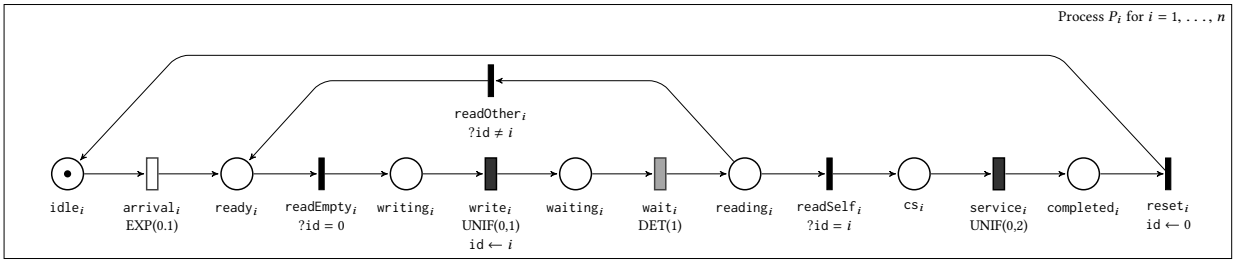

Figure 1: STPN model of three processes accessing a critical section with Fischer's mutual exclusion protocol.

The underlying stochastic process of this STPN is an MRP: multiple $\texttt{write}_i$ and $\texttt{wait}_i$ transitions with general PDF (uniform and deterministic, respectively) can be enabled at the same time but, when a process enters the critical section, the others are either idle (thus enabling the exponential transition $\texttt{arrival}_i$) or waiting in $\texttt{ready}_i = 1$ for id to become 0; the process encounters a regeneration point, where all general timers are newly enabled. This STPN can thus be analyzed using the regenerative analysis engines (transient or steady-state) with error term $\epsilon = 0$: the results are exact (up to numerical errors).

First, we consider the probability that, from an initial state where three processes are idle ($\texttt{id} = 0$ and $\texttt{idle}_i = 1$ for $i = 1, 2, 3$), P_1 reaches the critical section. This metric can be computed by evaluating the instantaneous reward "$\texttt{cs}_1$" (place $\texttt{cs}_1$ contains at most one token) under the stop condition "$\texttt{cs}_1 == 1$." We computed this metric also from the alternative initial state where $\texttt{id} = 0$, $\texttt{ready}_1 = 1$, and $\texttt{idle}_i = 1$ for $i = 2, 3$ to compare with case when P_1 is the first process that becomes ready. Transient probabilities were evaluated after steps of size 0.1 (this parameter is also used to solve Markov renewal equations: large step sizes can affect accuracy). Results obtained with the GUI are illustrated in Fig. 2.

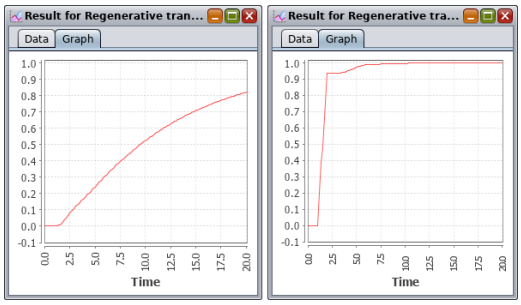

Figure 2: First-passage probability of states where $\texttt{cs}_1 == 1$.

Then, we export the model as a Java class and carry out a parametric study of performance at the steady-state (note that the state-space is irreducible). We run steady-state regenerative analysis with reward "$\texttt{cs}_1 + \cdots + \texttt{cs}_n$" for $\texttt{service}_i \sim \text{UNIF}(0, x)$, x ranging from 0.5 to 100.5, and $n = 2, 3, 4, 5$, which gives the results of Fig. 3. This reward represents the utilization of the critical section, which saturates when the number of processes or the service time increase.

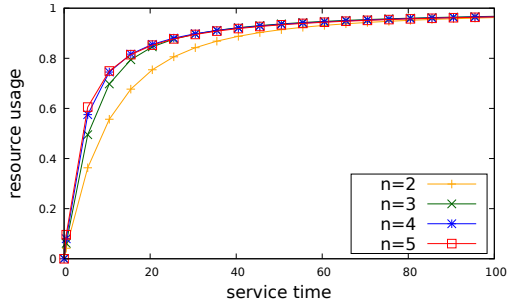

Figure 3: Steady-state utilization of the critical section.

REFERENCES

[1] A. Avritzer, L. Carnevali, L. Happe, A. Koziolek, D. S. Menasche, M. Paolieri, and S. Suresh. 2014. A Scalable Approach to the Assessment of Storm Impact in Distributed Automation Power Grids. In *QEST'14 (LNCS)*, Vol. 8657. Springer, 345–367. https://doi.org/10.1007/978-3-319-10696-0_27

[2] M. Biagi, L. Carnevali, M. Paolieri, T. Papini, and E. Vicario. 2017. Exploiting non-deterministic analysis in the integration of transient solution techniques for MRPs. In *QEST'17*. 20–35. https://doi.org/10.1007/978-3-319-66335-7_2

[3] M. Biagi, L. Carnevali, M. Paolieri, and E. Vicario. 2017. Performability evaluation of the ERTMS/ETCS – Level 3. *Transportation Research Part C: Emerging Technologies* 82 (2017), 314–336. https://doi.org/10.1016/j.trc.2017.07.002

[4] L. Carnevali, L. Grassi, and E. Vicario. 2009. State-Density Functions over DBM Domains in the Analysis of Non-Markovian Models. *IEEE Trans. Softw. Eng.* 35, 2 (March 2009), 178–194. https://doi.org/10.1109/TSE.2008.101

[5] L. Carnevali, M. Paolieri, K. Tadano, and E. Vicario. 2013. Towards the quantitative evaluation of phased maintenance procedures using non-Markovian regenerative analysis. In *EPEW'13 (LNCS)*, Vol. 8168. Springer, 176–190. https://doi.org/10.1007/978-3-642-40725-3_14

[6] L. Carnevali, M. Paolieri, F. Tarani, and E. Vicario. 2013. Quantitative Evaluation of Availability Measures of Gas Distribution Networks. In *VALUETOOLS'13*. ACM, 145–154. https://doi.org/10.4108/icst.valuetools.2013.254411

[7] R. German. 2000. *Performance Analysis of Communication Systems with Non-Markovian Stochastic Petri Nets*. Wiley.

[8] A. Horváth, M. Paolieri, L. Ridi, and E. Vicario. 2012. Transient analysis of non-Markovian models using stochastic state classes. *Perform. Eval.* 69, 7-8 (July 2012), 315–335. https://doi.org/10.1016/j.peva.2011.11.002

[9] A. Horváth, L. Ridi, and E. Vicario. 2010. Transient Analysis of Generalised Semi-Markov Processes Using Transient Stochastic State Classes. In *QEST'10*. IEEE, 231–240. https://doi.org/10.1109/QEST.2010.37

[10] S. Martina, M. Paolieri, T. Papini, and E. Vicario. 2016. Performance evaluation of Fischer's protocol through steady-state analysis of Markov regenerative processes. In *MASCOTS'16*. Springer, 355–360. https://doi.org/10.1109/MASCOTS.2016.72

[11] M. Paolieri, A. Horváth, and E. Vicario. 2016. Probabilistic Model Checking of Regenerative Concurrent Systems. *IEEE Trans. Softw. Eng.* 42, 2 (Feb 2016), 153–169. https://doi.org/10.1109/TSE.2015.2468717

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

[13] K. S. Trivedi and R. Sahner. 2009. SHARPE at the age of 22. *SIGMETRICS Perform. Eval. Rev.* 36, 4 (March 2009), 52–57. https://doi.org/10.1145/1530873.1530884

[14] E. Vicario. 2001. Static Analysis and Dynamic Steering of Time-Dependent Systems. *IEEE Trans. Softw. Eng.* 27, 8 (Aug. 2001), 728–748. https://doi.org/10.1109/32.940727

[15] E. Vicario, L. Sassoli, and L. Carnevali. 2009. Using Stochastic State Classes in Quantitative Evaluation of Dense-Time Reactive Systems. *IEEE Trans. Softw. Eng.* 35, 5 (Sept./Oct. 2009), 703–719. https://doi.org/10.1109/TSE.2009.36