Reengineering the Editor of the GreatSPN Framework

Elvio Gilberto Amparore

Università di Torino, Dipartimento di Informatica, Italy
amparore@di.unito.it

Abstract. This paper describes the technical challenges around the modernization process of the GreatSPN framework[15], one of the first Petri net frameworks started in the eighties, in particular in the reengineering of its Graphical User Interface and in its general user-friendliness, to account for its large set of functionalities[1].

Keywords: GreatSPN, GUI, Dataflow architecture.

1 Objectives and contributions

It is common opinion that formalisms like Petri nets or Timed Automata are very powerful and yet simple to understand thanks to their graphical representation. A graphic schema is usually simple to specify and grasp. However, graphical formalisms depends on the availability of good graphical editors to be able to gain the full advantages.

Back in 1985, the University of Torino developed the (probably) first documented software package for the analysis of *stochastic Petri nets*, under the name of *Graphical Editor and Analyzer for Timed and Stochastic Petri nets* (GreatSPN) [15]. The framework consisted in a set of tools for the analysis of *Generalized Stochastic Petri nets* (GSPN) [1], and it was supported by a graphical editor, described in [16], for interactively design, validate and evaluate GSPN models. The graphical editor, developed initially on a Sun 3 machine with UNIX BSD 4.2, was based on the X11/Motif toolkit. The simplicity of graphically designing models boosted the usage of GSPNs in various fields, from performance evaluation, telecommunications, biology and more. Other tools/GUIs have then followed, providing nowadays a large base of Petri net tools.

Today, the GreatSPN framework provides a vast collection of solvers developed in a time span of 30 years, that includes solvers optimized for low memory consumption (for the computers of the late '80s), modern model checkers based on decision diagram techniques [3], a stochastic model checker, ODE/SDE[2] solvers, Markov decision process optimizers, DSPN [23] solvers, simulators, and others. While the set of solvers of GreatSPN is very large, the obsolescence of technologies like Motif hurt the usability of the GUI over the years. After having evaluated other GUIs, the group arrived to the decision of renewing the interface of GreatSPN. The modular nature of the framework itself allows to easily replace solvers, modules and the GUI itself, while maintaining

[1] This work has been funded by the *Compagnia di San Paolo*, as a part of the AMALFI (Advanced Methodologies for the Analysis and management of the Future Internet) project.

[2] Ordinary and Stochastic Differential Equations.

the other modules fully working and unchanged. In the end, the modular framework structure proved to be easy to maintain and to develop over such a long timeframe.

This paper describes the reengineered GUI of GreatSPN, with its recent enhancements. The GUI is written in Java, and it is therefore portable to multiple platforms. Enhancements include, among all, support for drawing colored Petri nets and hybrid Petri nets, token game, batch measure specification and processing, and support for multiple solvers/model checkers. The paper describes the overall tool workflow, from the modeling and the verification phase, that allows to edit models, simulate their behaviors, inspect their structural properties, up to the evaluation phase, where performance indexes are computed with numerical solvers and/or simulators, and the computed results are visualized interactively to the user. A prototype of the GUI was briefly described in a short paper [5], centered around its use for stochastic model checking.

The GUI supports multiple formalisms: Generalized Stochastic Petri Nets (GSPN), GSPN with colors (*Stochastic Well-Formed net*, or SWN), Hybrid Petri nets, and Deterministic Timed Automata (DTA). In addition, the application presents a number of unique features, like multipage projects, solution batches, support for template variables in models, LaTeX labels and high quality vector graphics. This new GUI is described here with a focus on various recent additions: parametric measure specification, the support for SWN and hybrid Petri nets, and the integration inside the framework.

The modernization process actually required a process of re-engineering of the GUI around the workflow of GreatSPN. During this process, many limitations of the existing GUI have been removed, like the absence of a SWN token game, the missing support for model checkers, no capacity for drawing Timed Automata, and other. The main contributions provided by the modernized GreatSPN GUI are its improved usability, while keeping the compatibility with the large framework, and its support to multiple formalisms and solvers, which expands the tool usability. Other formalisms and other solvers may be added using the modular tool structure.

Section 2 describes the architecture of the GreatSPN framework. Section 3 and 4 introduce the application interface, and describe briefly the modeling capabilities of the editor. Section 5 shows how the user can simulate the designed GSPNs with the token game, and visualize their structural properties like the minimal P/T semi-flows. Section 6 describes how the designed models can be verified quantitatively with the set of supported solvers. Section 7 shows a simple use case of the tool that illustrates how the GUI can help the user in the process of modeling and analysis. The paper concludes with a comparison of other commonly used GUIs in section 8 and with the section 9 with a brief discussion on the future of GreatSPN.

2 Architecture of GreatSPN

GreatSPN is a large framework made by several interacting components, that has grown over the time to incorporate various Petri net-related features. The framework itself is not made by a large, monolithic tool. Instead, many independent tools interact by sharing data through files in standardized formats, resulting in a *dataflow architecture* approach. Each tool is responsible for reading its own input, written in one or more files, performing the computation, and writing the outputs in other files. The framework actu-

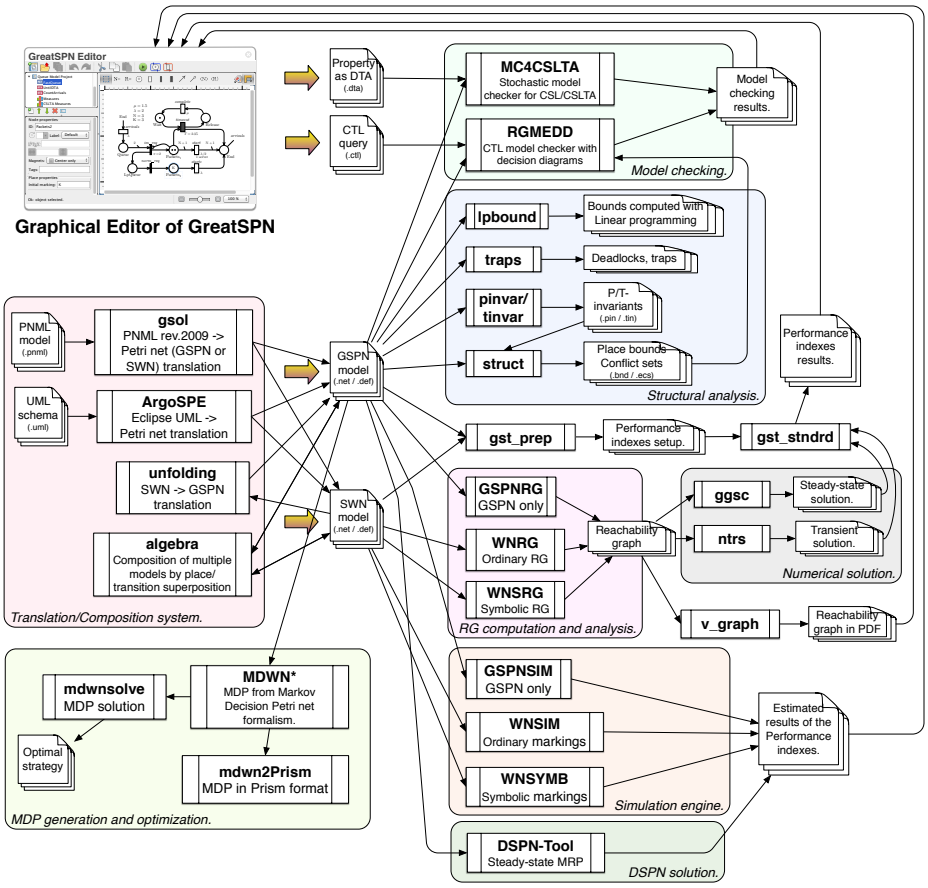


Fig. 1. (Partial) Architecture of the GreatSPN framework, as it is today.

ally contains more than 60 binaries. The advantage of this software architecture is that it allows to easily modify/replace single components, while keeping the rest of the framework unchanged, as long as the input/output formats are observed. While this software architecture is not very modern, it has proven to be very solid and maintainable, such that in the framework many software modules written in its 30 years of development co-exists, without causing too many troubles.

A simplified schema of the GreatSPN framework is shown in Fig. 1, that reports a selection of the various features of GreatSPN. Tools are written in bold, and are grouped in logical modules, according to their function, that span from numerical solutions, structural analysis, MDP support, conversion between multiple formalism, and so on. The graphical editor is the center for drawing the models and their properties. It is responsible for the invocation of various command line tools and for the visualization of the results. Actually, many but not all the command line tools are available from the GUI.

The workflow of GreatSPN was conceived, back in its original design, to be made in three main phases: first the user (the "modeler") designs the Petri net in a textual or graphical way; secondly, structural properties are computed (minimal P/T semi-flows, place bounds, conflict sets, ...) to understand if the model is designed properly and may be solved under numerical analysis or simulation; then the user specifies the measures of interest on the model and calls a command line solvers to do the computation. Several solvers are provided, for different types of models and with different characteristics.

2.1 Reengineering requirements.

In order to create a new GUI that replaces the old one, it must satisfy a set of requirements and constraints imposed by the framework itself. First of all, the GUI is responsible of these tasks:

1. Help the user in the process of drawing Petri net models and other graphical models.
2. Allow the user to call the tools provided by the GreatSPN framework for the structural analysis of Petri net models, to discover potential structural mistakes made in the drawn models.
3. Simplify the process of specifying and computing performance measures, by calling the command-line solvers and providing an understandable visualization of the computed results.

The design choices done to satisfy requirement 1 are explained in sections 3 and 4. Requirements 2 and 3 involve the interaction between the GUI and the command line tools. Command line tools expect precise file formats in input and produce other files as output, since these tools are designed to be non-interactive. Therefore, the tool interaction with the solvers require an explicit serialization of the data for the computation and a deserialization of the results. Many different files are involved in any computation: a partial list of these files is shown in Table 2.1.

Extension	**Content of the file**
.PNPRO	Petri net Project (XML format). Main format of the editor.
.net	Input format of Petri net models for command line solves.
.def	Input performance indexes.
.ctl	Qualitative queries in CTL language.
.dta	Deterministic timed automata for CSL^{TA} model checker.
.pin, .tin	P/T invariants.
.sta	Computed statistics.
.throu	Transition throughputs.
.tpd	Token distributions in places.
.ecs	Extended conflict sets.
.bnd	Upper/lower bounds of tokens in places.
.grg	Reachability graph.

Table 1. File extensions of the input/output files used by the editor and the solvers).

The complete solution process of a Petri net may require the invocation from one to twelve different command line tools, depending on the target measures to be computed. In addition, the reengineered workflow has been improved to support parametric models, i.e. models defined to depend on multiple integer/real parameters, whose values are specified at solution time and not at design time. Parameters are passed as command line arguments, and all command line tools have been modified to support them.

Another design requirement of the GUI is to support new modeling formalisms and functionalities that are not present in the current toolchain. To represent and store these informations, the tool uses a new XML file format for the models. This choice avoids modifying the input file formats of the toolchain. Any command line tool invocation serializes the drawn model in appropriate input formats when needed, leaving the main file format just for the editor. In this way, it is easy to change the editor format without breaking the compatibility with the command line tools of the GreatSPN framework.

Requirements 2 and 3 involve the reconstruction of the modeler workflow of GreatSPN inside the GUI. Examples of how this workflow is implemented graphically are given in sections 5–7.

2.2 Code structure of the new GUI.

The GUI consists of about 55K lines of code written in the Java™ language, plus an optional command line LaTeX engine that runs in the background to format the text labels of the models. The application is cross platform and runs on Windows, MacOSX and Linux. Java package structure is shown in Table 2.2.

Packages	Description
gui	Core GUI structure, main window cycle.
gui.net	Visualization/editing of abstract graphs (Petri nets, automata).
gui.play	Interactive token game.
gui.semiflows	Visualization of minimal P/T semi-flows.
gui.measures	Editing of measures and visualization of computed results.
domain	Data structures.
domain.project	File management, undo/redo facility.
domain.grammar	Unified ANTLRv4 grammar for expressions and measures.
domain.io	Serialization/deserialization in net/def, XML and APNN formats.
domain.values	Expression evaluation engine.
domain.elements.gspn	GSPN elements.
domain.elements.dta	DTA elements.
domain.play	Token game logic.
domain.semiflows	Computation of minimal P/T semi-flows.
domain.measures	Measure specification and tool invocations.
domain.unfolding	Unfolding of colored Petri nets into uncolored ones.

Table 2. Code structure of the Java application.

The core structure of the design view of the GUI is essentially an editor for abstract graphs of nodes and edges. The version described in this paper supports two graph

formalisms: Petri nets and automata. New formalisms can be added by deriving the corresponding base classes in the Java codebase. Adding a new formalism is done by deriving the base classes for the model, the node elements and the edge elements, and by providing the Java panels to edit properties. For instance, the DTA formalism, implemented in the domain.elemens.dta package, involves about 2K lines of code: two Java classes for the DTA locations and edges (the graph nodes), a class for the DTA model in a project, and the property panels for the location and edges. Of course, other part of the application that use DTAs also involve some additional logic. To abstract different syntax of properties, measures, expressions, provided by various solvers, the GUI has a uniform C-like language for expressions. When an expression needs to be passed to a solver, it is converted to the specific syntax expected by the tool. Abstracting expression languages of different solvers allows to support multiple solvers without having to re-specify expressions and measures for different tools. Overall, the complete GreatSPN framework amounts to about 500K lines of code, mostly made by C/C++ programs.

3 Drawing Petri net models

The core feature of the editor is the drawing of Petri net models, centered around the GSPN, the SWN and the Hybrid Petri net formalisms. Figure 2 shows the main application window, taken while editing a colored Petri net model. In the upper-left panel, there is the list of open projects. The editor is designed around the idea of *multi-page* projects. Each project correspond to a file, and is made by several pages. In the current version of the editor, pages can be of three types: Petri net models, DTA models or table of measures. In the lower-left panel of the main window there is the property panel, that shows the editable properties of the selected objects. The central canvas contains the editor of the selected project page, that is in this case a SWN model.

Petri nets are drawn with the usual graphical notation. Transitions may be immediate (thin black bars), exponential (white rectangles) or general (black rectangles). Names, arc multiplicities, transition delays, weights and priorities are drawn as small movable labels near the corresponding Petri net elements. Arcs may be "broken", meaning that only the beginning and the end of the arrows are shown. Color definitions are drawn in textual form, as in the upper right part of the window where two color classes, a composite color domain and two color variables are declared. The editor also supports fluid places and fluid transitions (not shown in the example of Fig. 2), and place partitions for Kronecker-based solutions [11]. The editing process supports all the common operations of modern interactive editors, like undo/redo of every actions, cut/copy/paste of objects, drag selection of objects with the mouse, single and multiple editing of selected objects, etc. Great care has been put to the graphical quality of the resulting Petri net models, to allow for high quality visualization of the net. The interface is designed to avoid modal dialog windows as much as possible, to streamline the use of the GUI.

Figure 3 shows some of the extended features of the Petri net editor. Name labels for elements (places, transitions, constants, etc) may appear in three user-selected modes:

- The label shown is the alphanumeric object identifier, as-is;
- A LaTeX string is used, allowing for more readable models that better express their meanings, like in the simple reaction network of Fig. 3(A) where alphanumeric

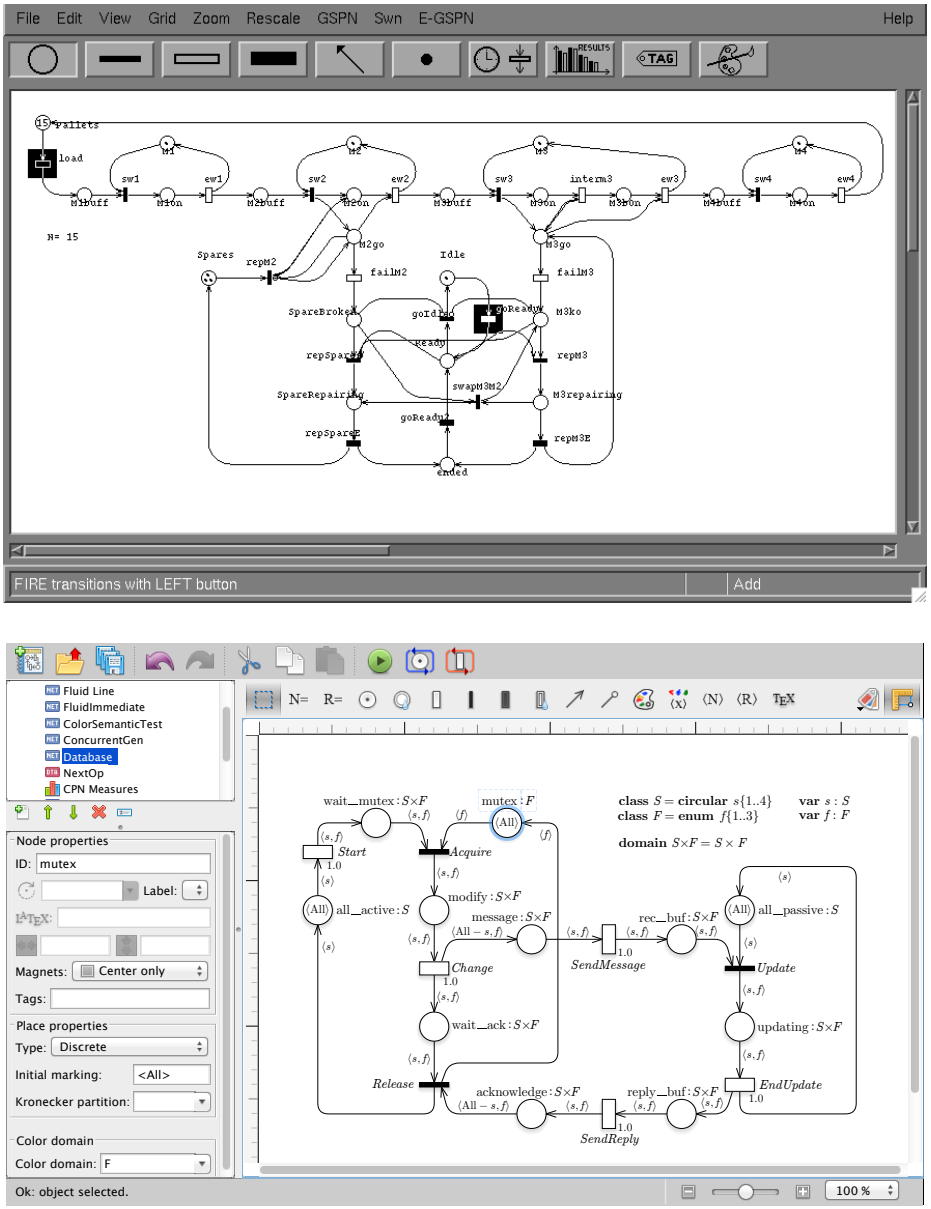
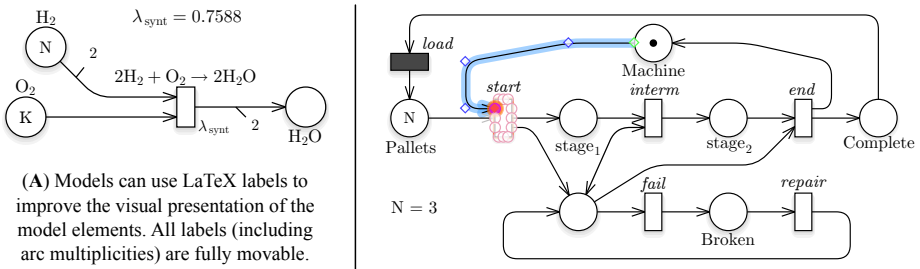


Fig. 2. The old and the new graphical user interfaces of GreatSPN. Screen-capture of the former is taken during the interactive token-game, while the SVG capture of the latter shows the design view with a colored Petri net model.



(**A**) Models can use LaTeX labels to improve the visual presentation of the model elements. All labels (including arc multiplicities) are fully movable.

(**B**) Customizable magnetic points where arrows can attach.

Fig. 3. Some features of the Petri net editor.

transition names are replaced with the represented chemical reaction, and place names represent the chemical species.

– The label is the object identifier automatically formatted in LaTeX with a function that tries to convert common patterns (like `alpha` → α, or `stage1` → stage_1).

Arc arrows point to the center of the attached transitions/places. If this behavior is not satisfactory, the editor provides a set of customizable "magnetic points" drawn on the element perimeters, where the arrows may attach. This behavior is shown in Fig. 3(B), figure that has been taken while dragging the arc arrow with the mouse on the "*start*" transition that has "3 magnets per side". All elements in the model are vector based, which result in high print quality. Printing and PDF exportation of the models are also possible, using the printing facilities of the operating system.

Figure 4 shows a colored model, drawn using the SWN formalism, as it appears in the editor window. Support for SWN has been recently added to the GUI. The model has three objects, located in the upper left part, that represents object declarations. The $\langle N \rangle$ objects declares a parametric integer constant, whose value is decided at verification time. The 'class' declaration defines a color class for the places in the Petri net, as usual in the SWN formalism. Places belonging to this color class are labeled with the place name followed by a colon and the color class name. The third declaration is a color variable named x of color class *Philo*. All expressions are parsed and verified syntactically and semantically on-the-fly, and appear in red if there is some error.

4 Drawing CSL$^{\text{TA}}$ DTAs

The second type of models that can be drawn with the editor are *Deterministic Timed Automata* (DTA), a type of timed automata for the CSL$^{\text{TA}}$ stochastic logic [19]. CSL$^{\text{TA}}$ works by measuring stochastic GSPN behaviors using a DTA. A DTA is an automaton that reads the language of GSPN firing sequences (also called *paths*), and separates accepted and rejected paths. The formal semantic of the DTA can be found in [19] (single clock), and in [14] (with multiple clocks). In few words, the logic provides a stochastic operator: $s_0 \models P_{\bowtie\lambda}(A)$ that is satisfied iff the overall probability of the set of GSPN paths starting in state s_0 and accepted by the DTA A, is $\bowtie \lambda$.

Figure 5 shows three CSL$^{\text{TA}}$ DTAs, drawn with the notation described in [4]. The first DTA describes the CSL [7] path property: Φ_1 *Until*$^{[\alpha,\beta]}$ Φ_2. Locations are drawn

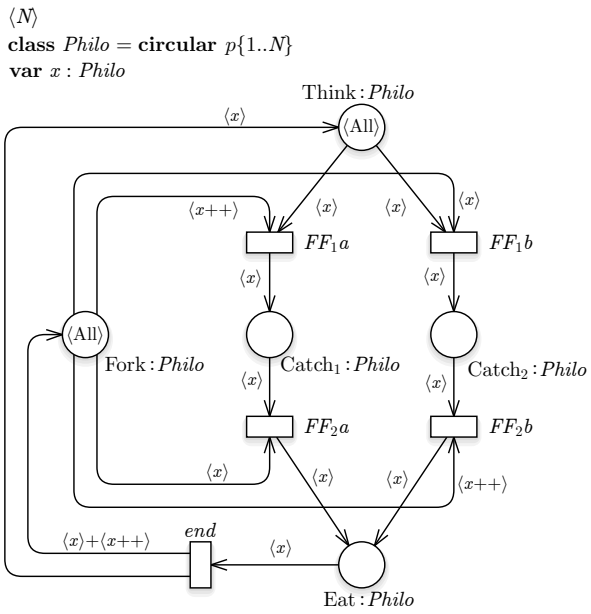
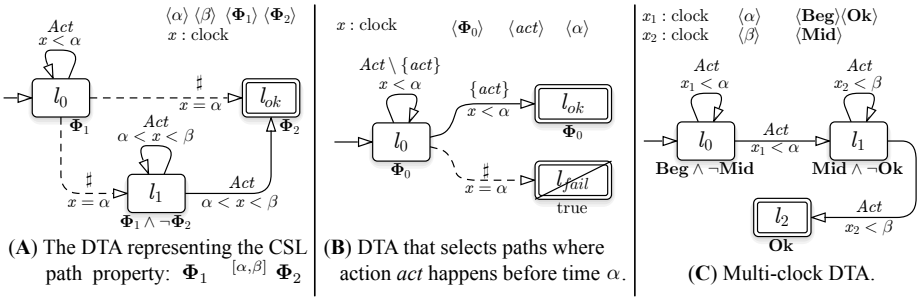


Fig. 4. Model of the *N* dining philosophers drawn in the SWN formalism.

as rounded rectangles, and the state proposition that the GSPN must satisfy while the DTA is in a location is written below the location rectangle, in bold. Initial locations are represented with an entering arrow, and final locations are drawn with a double border. The editor also allows *final rejecting locations*, not included in the original definition, but used in [2]. There are two kinds of edges, *boundary*, drawn dashed, and *inner*, drawn solid. Boundary edges are triggered as soon as the clock condition is satisfied, and are labeled with a ♯. Inner edges specify the set of GSPN actions with which they are synchronized. Each edge also specify a set of clock constraints, and an optional set of clock resets.



(**A**) The DTA representing the CSL path property: $\Phi_1 \ ^{[\alpha,\beta]} \ \Phi_2$

(**B**) DTA that selects paths where action *act* happens before time α.

(**C**) Multi-clock DTA.

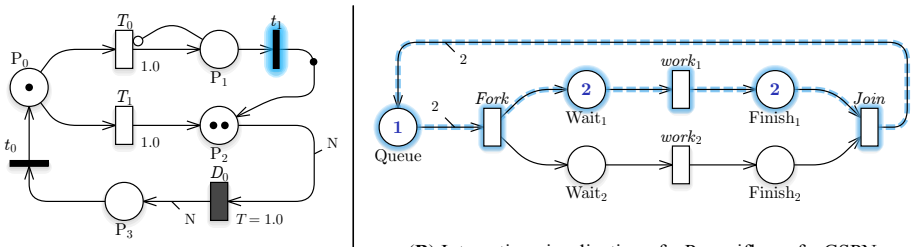
Fig. 5. Some example of DTA models drawn with the editor.

DTAs are parametric, and are not bound a priori to a specific GSPN model. Instead, all state propositions, real clock boundaries and action names are declared as *template variables* (depicted as $\langle var \rangle$). When the DTA is used for computating measures of a GSPN, these parameters are instantiated to boolean conditions, real values and transition names of the GSPN, as we shall see in section 5.1.

Clocks are declared as part of the DTA. The DTAs (A) and (B) of Fig. 5 have a single clock, while the DTA (C) has two clocks. Currently, only single-clock DTAs can be verified numerically. The DTA (B) accepts all the GSPN where a transition *act* fires before time α, while remaining in states that satisfy the condition Φ_0.

5 Interactive simulation and inspection of structural properties

The behavior of Petri nets can be experimented interactively inside the GUI. This is known as the "token game" or "interactive simulation", and works as follows. The editor shows the initial marking of the GSPN, and highlights the set of enable transitions of the model. By click on one of the enabled transitions, the editor responds by firing the tokens from the input to the output places, showing the behavior of the model. The reached marking is then shown, and the user can continue firing new transitions.



(A) Token simulation screenshot taken while firing immediate transition t1.

(B) Interactive visualization of a P-semiflow of a GSPN. P-semiflow multiplicities are written on the places.

Fig. 6. Interface of the interactive GSPN simulation and semi-flows visualization.

Figure 6(A) shows this interactive simulation on a GSPN model, taken during the firing of transition t_1. Tokens removed from input places and added to output places are drawn with a short animation. Token game works in untimed and timed mode. In untimed mode, no age/duration of the events is considered, and no track is kept for the advance of time. In timed mode instead, a time is present, and the time for the transition fire is taken into account. For DSPN models with non-exponential transitions, timing constraints are resolved. The user may also enable a *semi-automatic firing* mode, where interaction is required only if there is a choice between multiple concurrently enabled transitions, or a *random firing* mode where the editor picks the next transition randomly (and the next time in timed simulation), thus simulating without the user interaction. SWN models are supported in interactive mode: instead of black dots, colored tokens are shown as color names. When firing a colored transitions, a list of enabling bounds of the color variables is shown to the user, who can pick the one to fire.

Additionally, the user may visualize the minimal P and T semi-flows that covers a GSPN model, as shown in Fig. 6(B). The user selects the minimal semi-flow that wants to visualize from a list, and the editor highlights the involved places and transitions. Semi-flows are computed with the modified Farkas method of Martinez and Silva [24]. With these tools, the user may inspect the behavior and the structural properties of the Petri net while modeling, which is useful to verify that the model is drawn correctly.

5.1 Interactive CSL^{TA} simulation

An interactive simulation of the path probability operator of the CSL^{TA} logic is, roughly speaking, a system where GSPN firings are checked by the DTA. Each GSPN transition firing has to be matched by a corresponding DTA edge, otherwise the path is rejected. In addition, boundary edges of the DTA (labeled with a $\sharp$ and drawn as dashed arrows) are autonomous and are taken as soon as their timing conditions are met. Before starting the simulation, the template variables of the models are shown as a list of text boxes, that must be filled by the user with appropriate values. Values assigned to the parametric variables of the DTA are shown above the DTA. In the central panel of the window, the GSPN and the DTA are shown side by side.

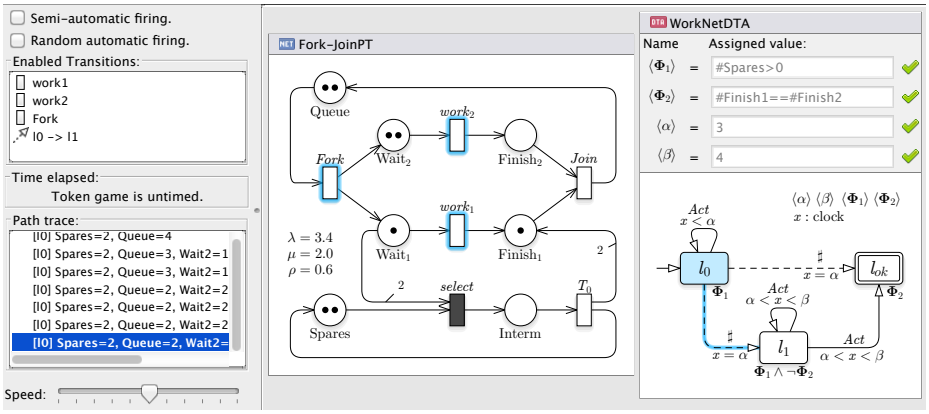


Fig. 7. Interface of the interactive CSL^{TA} model checking simulation.

Figure 7 shows the GUI window for the joint simulation of a GSPN model and a DTA. The list of enabled GSPN transitions and autonomous DTA edges is shown in the upper-left corner. In the lower left corner there is the state of the path trace chosen interactively by the user, starting from the initial marking. Values assigned to the parametric variables are validated while typing, and their correctness is signaled with a green tick mark on the right of the corresponding text boxes. When all values are assigned, the user may press the play button and the joint simulation starts in the initial state of the GSPN and in the initial location of the DTA. Each time the user selects a GSPN transition to fire, a DTA inner edge has to be chosen afterwards to match the GSPN firing. Boundary edges of the DTA may also be independently enabled

(clock condition is evaluated in a timed simulation, and ignored in an untimed one). The simulation ends when the DTA reaches a final location, or when no DTA edge can match a GSPN firing.

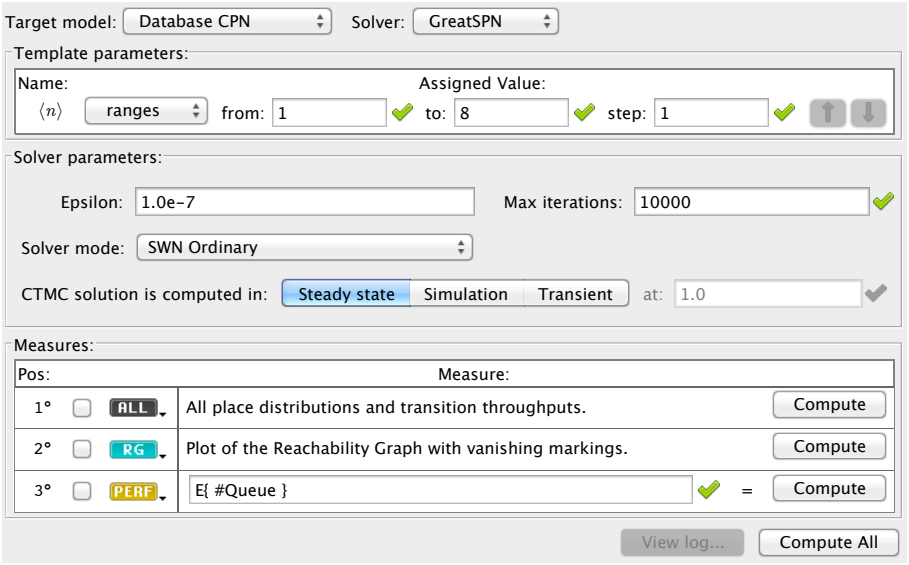


Fig. 8. The interface for specifying and computing a batch of measures.

6 Computing measures

The GUI integrates an interface for specifying, computing and visualizing measures on Petri net models. A project may contain multiple *measure pages*, and each page specifies:

- The target Petri net model;
- The selected numerical solver, from a list of supported solvers;
- The instantiation of the parameters of model, if any;
- Solver-specific parameters and flags;
- A table of target measures that will be computed.

The GUI is currently integrated with three solvers. The first is the GreatSPN toolchain, that can evaluate standard performance measures (mean number of tokens in a place, transition throughputs, etc...) on GSPN/SWN models using an extensively tested implementation. Index can be computed in steady-state or in transient with a numerical solver, or by using a simulator. The second solver is the MC4CSL^{TA} stochastic model checker, that can evaluate standard performance measures for GSPN and DSPN models [6], as well as CSL and CSL^{TA} queries. The third solver is RGMEDD [3], the symbolic CTL model checker of GreatSPN [8].

Figure 8 shows a measure page editor for a GSPN model with one parametric marking parameter $\langle n \rangle$ and with three measures (at the bottom). The GSPN model is evaluated multiple times for different values of n, from 1 to 10 (template parameters section), with increments of 1. The numerical solution is computed by invoking the command-line solvers with the specified solver parameters (solution in steady-state, maximum number of iterations, use the ordinary SWN solution, etc..). The table of measures lists what will be computed. Entry ALL specifies that all basic GSPN measures will be computed, which are the distributions of tokens in each place, and the transition throughputs. RG and TRG specify that the (Tangible) Reachability Graph will be generated by the GreatSPN tools, and a graphical representation will be drawn. Queries in a given language (CTL, CSL, CSL^{TA}, Performance measures) can be specified textually. A PERF query expresses a performance measure on places and transitions, using the syntax of the measure language of GreatSPN.

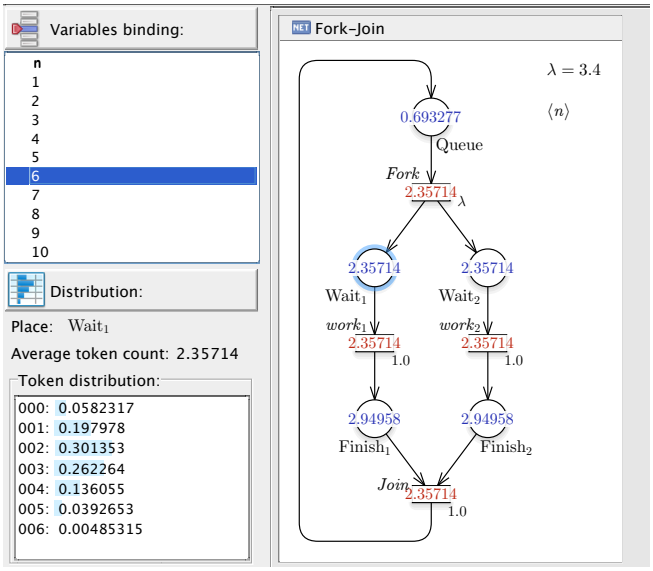


Fig. 9. The interface that shows the ALL results computed on a parametric GSPN in steady-state.

When the user clicks on the "Compute" button, the GUI calls the command-line numerical solvers, and shows the output to the user. To use the command line tools directly, the user can export the GSPN/DTA models as separate files. Currently supported formats are the GreatSPN format and the APNN format [20].

Computed solutions are shown interactively to the user. Figure 9 shows the interface that is used to show the results of the ALL measure, computed with parameter $\langle n \rangle$ that ranges from 1 to 10. Places and transitions show their expected number of tokens and throughputs, respectively, for the value selected by the user (in this case $n = 6$). When the user selects a place, its token distribution is shown in the bottom-left corner.

Template parameters can be bound to a single value, a list of values or a range of values. If the performance measures are computed in transient, it is possible to specify that the transient time t is template variable, thus computing a sequence of solutions at different time steps. This allows the user to setup a batch of parametric tests with a certain degree of flexibility.

7 Use case

We now present a simple use case to show how the tool functionalities can be used to support model analysis with standard performance measures as well as with CSL$^{\text{TA}}$ queries.

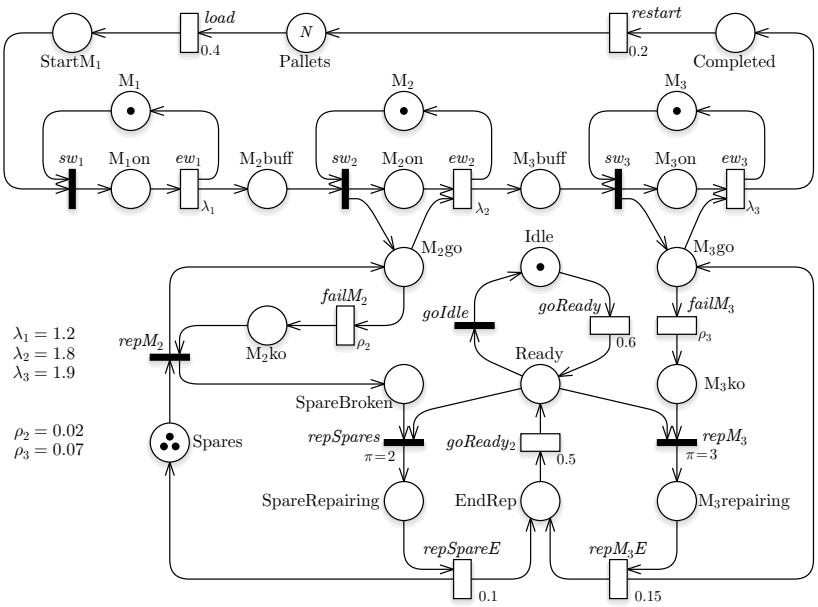
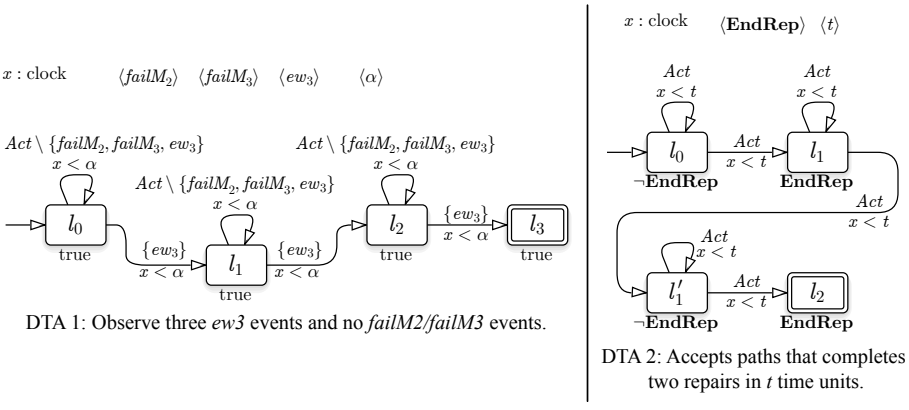


Fig. 10. The GSPN of a Flexible Manufacturing System (FMS).

Figure 10 shows an instance of a *Flexible Manufacturing System* (FMS) taken from [9], modeled with the GUI and exported as PDF. The model represents a system where N pallets are treated in a sequence of three machines, M_1, M_2 and M_3. Each machine can treat one pallet at a time. Machine 2 and 3 are subject to breakages, and a repairman continuously checks the machine for repairs. Machine 2 has a set of spare parts that can be used to replace the broken parts, without losing work time. Machine 3 instead always requires a stop to do the repair. The model is parametric in the number N of circulating pallets.

Figure 11 depicts two DTAs drawn with the GUI that express two path properties on the FMS model. The first accepts the system behaviors where three completion events



DTA 1: Observe three *ew3* events and no *failM2/failM3* events.

DTA 2: Accepts paths that completes two repairs in t time units.

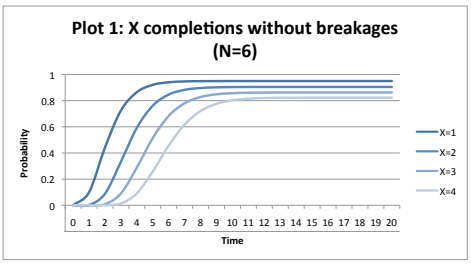
Fig. 11. The Two DTAs used in the analysis of the FMS model.

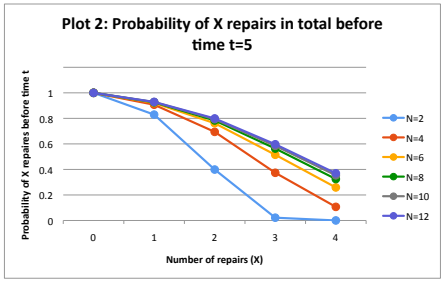
ew_3 are observed before time α, having not seen any failure of the machines M_2 and M_3. The second DTA accepts the paths where at least two repairs have been completed in t time units.

To carry on the analysis from the editor, it is sufficient to create a new set of measures for the FMS model, that are parametric in the number N of pallets and on the time t of the DTA. Figure 12 shows the results of the analysis of the FMS model by computing the steady state solution and the model checking of the two DTAs. Datas are computed using the GUI, and then exported from the GUI in Excel Open XML format to plot the diagrams.

				GMRES	
N	States	Transitions	Visited Vanishings	Iterations	Time
2	259	734	324	161	0,01
4	1460	5565	2414	258	0,08
6	4858	21280	9246	325	0,34
8	12225	58047	25420	380	1,04
10	25861	129426	57168	501	2,67
12	48594	252369	112354	634	6,25
14	83780	447220	200474	793	15,02
16	135303	737715	332656	1297	41,91
18	207575	1150982	521660	1854	122,34

State space of the FMS net and time/iterations required to achieve convergence of the steady-state solution.





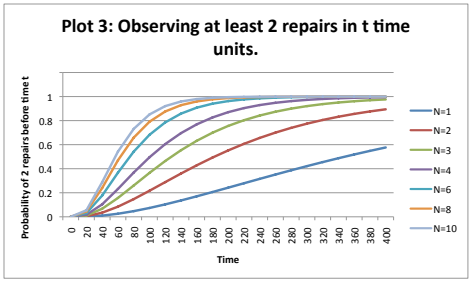


Fig. 12. Results of the FMS model analysis, visualized in Excel.

The table in the upper-left corner shows the size of the FMS model state space, the number of transitions, the number of vanishing states visited by the numerical solver, and the iterations/time needed to compute the steady state solution on a 2.4Ghz Intel machine with accuracy of 10^{-7}.

Plot 1 represents the result of DTA 1 on the FMS with N=6 pallets for 1,2,3,4 completions (the DTA of Figure 11 represents the case X=3). The plot shows the probability of having enough time to do X completions, and for large values of t converges to the probability of observing a failure. Plot 2 shows the results of DTA 1 by variating the number of circulating pallets N with a fixed time $t = 5$. The x-axis plots the number of completions, while the y-axis shows the overall probability of completing X tasks before time t. A larger number of pallets increases the throughput of the system, resulting in an increased probability. Finally, Plot 3 shows the results of DTA 2, i.e. the probability of observing two repairs in t time units. Time is plotted on the x-axis and probability on the y-axis, for various numbers of circulating pallets. Since the machine may break when it is under usage, the probability increases for higher values of N.

8 Related work

While the GreatSPN framework with the new interface provides a solid base for editing, verifying and computing quantitative/qualitative measures of GSPN/SWN models, there are other tools that provide similar features. Before reimplementing a new GUI, we have explored various alternatives. An (incomplete) list is given.

Möbius : The aims of the Möbius tool [18], developed at the University of Illinois, are similar to those of GreatSPN. It supports multiple formalisms, multiple solvers, and provides a complete analysis workflow, from design to verification to the numerical solution. It supports analysis of discrete and continuous time Markov chains, Markov regenerative processes and a powerful simulator. However, the central formalism is the SAN, not the stochastic Petri net, so it is not directly suitable for GreatSPN (even if SAN nets can be converted to GSPN). In addition, no SWN and no stochastic model checking is available.

Snoopy : The tool Snoopy [21] is a proprietary software developed at Cottbus TU. It provides a unified editor for Petri net models, with support for hierarchical composition and multiple solvers. Snoopy supports hybrid Petri nets (HPN), colored Petri nets, as well as other extensions. The main solver is Marcie, based on MTBDD/MTIDD (decision diagram variants) techniques.

Coloane : Coloane [17] provides support for both Petri net and Timed Automata, but is currently not focused on stochastic formalisms. It is designed to provide a GUI around the standard PNML format [12], an XML-based exchange format for Petri net models.

CPNtool : CPN is a powerful toolkit for the design and evaluation of colored Petri nets [22]. The formalism adopted by CPN includes a color algebra, expressions in ML. The tool is supported by a flexible simulation engine. The specific mix of ML code

and Petri net graphics is very compact and powerful, but unfortunately is far from what GreatSPN solvers expect. In addition, the tool has some portability concerns. Therefore,a conversion between CPNtool models and GreatSPN models appears difficult.

TimeNet A successor of the DSPN-express tool developed at TU Berlin, TimeNET is a modern tool for editing stochastic and colored Petri nets. It is still being developed, and it has been recently updated with heuristic optimization techniques [13].

The specific characteristic of the GreatSPN models, and the vast number of solvers lead to the decision of designing a specific GUI for it. Additionally, some features like the DTA specification and the support to the CSL^{TA} stochastic logic are, to the best of our knowledge, a unique feature of the GreatSPN GUI, and are not found on other tools. The tool is available at `http://www.di.unito.it/~greatspn/index.html`, in the "New Java GUI" section, either as a part of GreatSPN or as a standalone version. A virtual machine with all the tools installed is also available, at request.

9 Conclusions and Future works

This paper presents an in-depth analysis of a new graphical user interface for the GreatSPN framework. The GUI is designed around a complete workflow for the modeling of Petri nets and DTAs, and includes graphical interactive analysis, specification of measures, computation and interactive visualization of the results, and an integration with multiple solvers/simulators/model checkers/optimizer/translators including a CSL^{TA} stochastic model checker and GreatSPN. A small use case has been also presented, to show the effectiveness of the GUI modeling capabilities and analysis with measures from the user point of view.

Since the tool architecture is scalable and customizable, we plan to extend the tool in various directions. First of all, the Petri net formalism can be augmented to support other extensions, like compositional formalism. Similarly, DTAs can be extended to cover more complete statistical control automata, like Linear Hybrid Automata [10]. Solvers and file formats of other framework can also be considered, like PNML, and there is an ongoing work to support solvers[11] of the APNN-Toolbox of TU-Dortmund.

References

1. Ajmone Marsan, M., Conte, G., Balbo, G.: A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems. ACM Transactions on Computer Systems 2, 93–122 (May 1984), `http://doi.acm.org/10.1145/190.191`
2. Amparore, E., Donatelli, S.: Improving and assessing the efficiency of the MC4CSL^{TA} model checker. In: EPEW 2013 (2013)
3. Amparore, E.G., Beccuti, M., Donatelli, S.: (Stochastic) model checking in GreatSPN. In: Application and Theory of Petri Nets, LNCS, vol. 8489, pp. 354–363. Springer (2014)
4. Amparore, E.G.: State, action, and path properties in Markov chains. Ph.D. thesis, Dipartimento di Informatica, Università di Torino, Italy (2013)
5. Amparore, E.G.: A New GreatSPN GUI for GSPN Editing and CSLTA Model Checking. In: Norman, G., Sanders, W. (eds.) Quantitative Evaluation of Systems, Lecture Notes in Computer Science, vol. 8657, pp. 170–173. Springer International Publishing (2014)

6. Amparore, E.G., Donatelli, S.: Revisiting the Iterative Solution of Markov Regenerative Processes. Numerical Solution of Markov Chains (NSMC) (2010)
7. Aziz, A., Sanwal, K., Singhal, V., Brayton, R.: Model-checking continuous-time Markov chains. ACM Trans. Comput. Logic 1(1), 162–170 (2000)
8. Babar, J., Beccuti, M., Donatelli, S., Miner, A.S.: GreatSPN Enhanced with Decision Diagram Data Structures. In: Lilius, J., Penczek, W. (eds.) Petri Nets. LNCS, vol. 6128, pp. 308–317. Springer (2010)
9. Balbo, G., Beccuti, M., De Pierro, M., Franceschinis, G.: First Passage Time Computation in Tagged GSPNs with Queue Places. The Computer Journal (2010), first published online July 22, 2010.
10. Ballarini, P., Djafri, H., Duflot, M., Haddad, S., Pekergin, N.: HASL: An expressive language for statistical verification of stochastic models. In: Proceedings of VALUETOOLS'11. pp. 306–315. Cachan, France (May 2011)
11. Bause F. and Buchholz P. and Kemper P.: Hierarchically combined queueing petri nets. In: In Proc. 11th Int. Conf. on Analysis and Optimization of Systems, Discrete Event Systems, Sophie-Antipolis. pp. 176–182. Sophie-Antipolis,France (1994)
12. Billington, J., Christensen, S., Van Hee, K., Kindler, E., Kummer, O., Petrucci, L., Post, R., Stehno, C., Weber, M.: The petri net markup language: Concepts, technology, and tools. In: Proceedings of the 24th International Conference on Applications and Theory of Petri Nets. pp. 483–505. ICATPN'03, Springer-Verlag, Berlin, Heidelberg (2003)
13. Bodenstein, C., Zimmermann, A.: Timenet optimization environment: Batch simulation and heuristic optimization of scpns with timenet 4.2. In: Proceedings of the 8th International Conference on Performance Evaluation Methodologies and Tools. pp. 129–133. VALUETOOLS '14, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), ICST, Brussels, Belgium, Belgium (2014)
14. Chen, T., Han, T., Katoen, J.P., Mereacre, A.: Quantitative Model Checking of Continuous-Time Markov Chains Against Timed Automata Specifications. Logic in Computer Science, Symposium on 0, 309–318 (2009)
15. Chiola, G.: A software package for the analysis of generalized stochastic petri net models. In: International Workshop on Timed Petri Nets. pp. 136–143. IEEE Computer Society, Washington, DC, USA (1985)
16. Chiola, G.: A Graphical Petri Net Tool for Performance Analysis. In: Third Int. Workshop on Modeling Techniques and Performance Evaluation. pp. 323–333. Paris, France (1987)
17. Coloane webpage. https://coloane.lip6.fr/
18. Courtney, T., Daly, D., Derisavi, S., Gaonkar, S., Griffith, M., Lam, V., Sanders, W.: The Mobius modeling environment: recent developments. In: International Conference on Quantitative Evaluation of Systems (QEST). pp. 328–329 (2004)
19. Donatelli, S., Haddad, S., Sproston, J.: Model checking timed and stochastic properties with CSL^{TA}. IEEE Trans. Softw. Eng. 35(2), 224–240 (2009)
20. F. Bause, P.K., Kritzinger., P.: Abstract Petri nets notation. In: Petri Net Newsletter. vol. 49, pp. 9–27 (1995)
21. Heiner, M., Herajy, M., Liu, F., Rohr, C., Schwarick, M.: Snoopy: A unifying petri net tool. In: Application and Theory of Petri Nets, LNCS, vol. 7347 (2012)
22. Jensen, K., Kristensen, L.M.: Coloured Petri Nets: Modelling and Validation of Concurrent Systems. Springer Publishing Company, Incorporated, 1st edn. (2009)
23. M. Ajmone Marsan, Chiola, G.: On Petri nets with deterministic and exponentially distributed firing times. In: Advances in Petri Nets. vol. 266/1987, pp. 132–145. Springer Berlin / Heidelberg (1987)
24. Martinez, J., Silva, M.: A simple and fast algorithm to obtain all invariants of a generalized Petri net. In: Selected Papers from the First and the Second European Workshop on Application and Theory of Petri Nets. pp. 301–310. Springer-Verlag, London, UK, UK (1982)