PeabraiN: A PIPE Extension for Performance Estimation and Resource Optimisation – Tool paper –

Ricardo J. Rodríguez, Jorge Júlvez, José Merseguer
Dpto. de Informática e Ingeniería de Sistemas
Universidad de Zaragoza, María de Luna 1, 50018 - Zaragoza, Spain
Email: {rjrodriguez, julvez, jmerse}@unizar.es

***Abstract*—Many discrete systems with shared resources from different artificial domains (such as manufacturing, logistics or web services) can be modelled in terms of timed Petri nets. Two studies that may result of interest when dealing with such a systems are the performance evaluation (or completed jobs per unit of time) and the resource optimisation. Exact performance evaluation, however, may become unachievable due to the necessity of an exhaustive exploration of the state-space. In this context, a solution can be to estimate the performance by computing bounds. Resource optimisation leverages a budget and distributes resources in order to maximise the system performance. In this paper, we present PeabraiN, a collection of PIPE tool-compliant modules for performance estimation and resource optimisation based on bounds computation for Stochastic Petri Nets. The algorithms supporting the modules make an intensive use of linear programming techniques and therefore their computational complexity is low. Besides, other PN properties, such as structural enabling bound at a transition, structural marking bound at a place or visit ratios computation, are added to PIPE tool as well.**

***Index Terms*—Performance evaluation, Petri nets, Software performance, Discrete Event Systems**

I. Introduction

Many discrete systems can be modelled in terms of Stochastic Petri Nets (SPNs) [1]. Such systems may need the use of shared resources. Two studies that are often of interest are: (i) the performance evaluation (or *throughput*, defined as completed jobs per unit of time), and (ii) the resource optimisation, i.e., to have optimally sized the number of shared resources in the system.

Exact performance evaluation may become unachievable, in terms of computation time, due to the need of an exhaustive exploration of the state-space. Normally, the larger the system, the bigger its state-space. An alternative is to estimate the performance by computing performance bounds [2]–[4].

Resource optimisation is another master key when designing these systems. When resources are not well-dimensioned [5], it may happen that either the throughput is constrained by lack of available resources (then performance is lower than it could be), or there are idle resources (then money has been squandered).

In this paper, we present PeabraiN, a collection of PIPE [6] tool-compliant modules for performance estimation and resource optimisation based on bounds computation for SPNs. The algorithms supporting such modules, which have been previously published [2], [4], [7], [8], intensively use linear programming (LP) techniques, then assuring low computational complexity. Besides, other modules have been added for computing other properties based in LP techniques, such as the computation of structural enabling(marking) bound at a transition(place). Visit ratios computation and SPN simulation analysis modules have been integrated to PIPE as well.

We studied different choices for implementation: implementation of a stand-alone MATLAB application; extension of the GreatSPN [9] tool; extension of the HISim tool; and to develop modules to be integrated in PIPE tool.

The only tool for performance bound computation, as far as our knowledge, is GreatSPN, which computes lower and upper throughput bounds of transitions. The extension of GreatSPN was finally rejected because of the programming language paradigm used, and its platform dependency. All bound computation algorithms presented in this paper were initially developed for MATLAB. Nevertheless, a final deployment of this solution was ruled out by the dependency of a proprietary software library (namely, MATLAB Component Runtime library). A solution deployed over HISim was rejected when we figured out the easiness of extension through modules directly over PIPE tool. For resource optimisation, as far as our knowledge, there does not exist any tool.

PIPE was chosen because (i) we need only SPNs and not other PN extensions, (ii) it uses the standard PN file format, Petri Net Markup Language (PNML) [10], so it allows an interchange of files between different PNML-compliant tools, (iii) PIPE facilitates a user-friendly GUI editor, (iv) it is multiplatform, and (v) it is open source.

The remainder of this paper is organised as follows. Section II introduces the theory concepts behind the tool. Section III presents the PeabraiN framework design and the features added to PIPE tool. An illustrative example showing the benefits of using PeabraiN is introduced in Section IV. Section V presents the installation requirements and the tool availability. Lastly, Section VI outlines some conclusion and future work in PeabraiN.

II. Theory Overview behind PeabraiN

This section introduces concepts and references for the algorithms implemented in PeabraiN.

A. Preliminary Concepts

A Petri net [11] is a 4–tuple $\mathcal{N} = \langle P, T, \mathbf{Pre}, \mathbf{Post} \rangle$, where P and T are disjoint non-empty sets of *places* and *transitions*, and $\mathbf{Pre}$ ($\mathbf{Post}$) are the pre–(post–)incidence non-negative integer matrices of size $|P| \times |T|$. The *pre-* and *post-set* of a node $v \in P \cup T$ are respectively defined as ${}^\bullet v = \{u \in P \cup T | (u,v) \in F\}$ and $v^\bullet = \{u \in P \cup T | (v,u) \in F\}$, where $F \subseteq (P \times T) \cup (T \times P)$ is the set of directed arcs. A Petri net is said to be *self-loop free* if $\forall p \in P, t \in T \; t \in {}^\bullet p$ implies $t \notin p^\bullet$. *Ordinary* nets are Petri nets whose arcs have weight 1. The *incidence matrix* of a Petri net is defined as $\mathbf{C} = \mathbf{Post} - \mathbf{Pre}$.

A vector $\mathbf{m} \in \mathbb{Z}_{\geq 0}^{|P|}$ which assigns a non-negative integer to each place is called *marking vector* or *marking*. A *Petri net system*, or *marked Petri net* $\mathcal{S} = \langle \mathcal{N}, \mathbf{m_0} \rangle$, is a Petri net $\mathcal{N}$ with an *initial marking* $\mathbf{m_0}$.

A transition $t \in T$ is *enabled* at marking $\mathbf{m}$ if $\mathbf{m} \geq \mathbf{Pre}(\cdot, t)$, where $\mathbf{Pre}(\cdot, t)$ is the column of $\mathbf{Pre}$ corresponding to transition t. A transition t enabled at $\mathbf{m}$ can *fire* yielding a new marking $\mathbf{m'} = \mathbf{m} + \mathbf{C}(\cdot, t)$ (*reached* marking). This is denoted by $\mathbf{m} \xrightarrow{t} \mathbf{m'}$. A sequence of transitions $\sigma = \{t_i\}_{i=1}^{n}$ is a *firing sequence* in $\mathcal{S}$ if there exists a sequence of markings such that $\mathbf{m_0} \xrightarrow{t_1} \mathbf{m}_1 \xrightarrow{t_2} \mathbf{m}_2 \ldots \xrightarrow{t_n} \mathbf{m}_n$. In this case, marking $\mathbf{m}_n$ is said to be *reachable* from $\mathbf{m_0}$ by firing σ, and this is denoted by $\mathbf{m}_0 \xrightarrow{\sigma} \mathbf{m}_n$. The *firing count vector* $\boldsymbol{\sigma} \in \mathbb{Z}_{\geq 0}^{|T|}$ of the firable sequence σ is a vector such that $\boldsymbol{\sigma}(t)$ represents the number of occurrences of $t \in T$ in σ. If $\mathbf{m_0} \xrightarrow{\sigma} \mathbf{m}$, then we can write in vector form $\mathbf{m} = \mathbf{m_0} + \mathbf{C} \cdot \boldsymbol{\sigma}$, which is referred to as the *linear* (or *fundamental*) *state equation* of the net.

Two transitions t, t' are said to be in *structural conflict* if they share, at least, one input place, i.e., ${}^\bullet t \cap {}^\bullet t' \neq \emptyset$. Two transitions t, t' are said to be in *effective conflict for a marking* $\mathbf{m}$ if they are in structural conflict and they are both enabled at $\mathbf{m}$. Two transitions t, t' are in *equal conflict* if $\mathbf{Pre}(\cdot, t) = \mathbf{Pre}(\cdot, t') \neq \mathbf{0}$.

A Petri net is said to be *strongly connected* if there is a directed path joining any pair of nodes of the graph. A *state machine* is a particular type of ordinary Petri net where each transition has exactly one input arc and exactly one output arc, that is, $|t^\bullet| = |{}^\bullet t| = 1, \forall t \in T$.

A *p-semiflow* is a non-negative integer vector $\mathbf{y} \geq \mathbf{0}$ such that it is a left anuller of the net's incidence matrix, $\mathbf{y}^\intercal \cdot \mathbf{C} = 0$[1]. A p-semiflow $\mathbf{v}$ is *minimal* when its support, $\|\mathbf{v}\| = \{i | \mathbf{v}(i) \neq 0\}$, is not a proper superset of the support of any other p-semiflow, and the greatest common divisor of its elements is one.

A process Petri net (PPN) [12] is a strongly connected self–loop free Petri net $\mathcal{N} = \langle P, T, \mathbf{Pre}, \mathbf{Post} \rangle$ where (i) $P = P_0 \cup P_S \cup P_R$ is a partition such that $P_0 = \{p_0\}$ is the *process-idle place*, $P_S \neq \emptyset, P_S \cap P_0 = \emptyset, P_S \cap P_R = \emptyset$, P_S is the set of *process-activity places* and $P_R = \{r_1, \ldots, r_n\}$, $n > 0, P_R \cap P_0 = \emptyset$ is the set of *resources places*; (ii) the subnet $\mathcal{N'} = \langle P \setminus P_R, T, \mathbf{Pre}, \mathbf{Post} \rangle$ is a strongly connected state machine, such that every cycle contains p_0; (iii) for each $r \in P_R$, there exist a unique minimal p-semiflow associated to r, $\mathbf{y}_r \in \mathbb{N}^{|P|}$, fulfilling: $\|\mathbf{y}_r\| \cap P_R = \{r\}, \|\mathbf{y}_r\| \cap P_S \neq \emptyset, \|\mathbf{y}_r\| \cap P_0 = \emptyset$ and $\mathbf{y}_r(r) = 1$; and (iv) $P_S = \bigcup_{r \in P_R} (\|\mathbf{y}_r\| \setminus \{r\})$.

A *Stochastic Petri Net* system (SPN) is a pair $\langle \mathcal{S}, \delta \rangle$ where $\mathcal{S} = \langle P, T, \mathbf{Pre}, \mathbf{Post}, \mathbf{m_0} \rangle$ is a Petri net system and $\delta : T \to \mathbb{R}^+$ is a positive real function such that $\delta(t)$ is the mean of the exponential firing time distribution associated to each transition $t \in T$. If $\delta(t) > 0$, then transition t is a timed transition. Otherwise, i.e., $\delta(t) = 0$, transition t is an immediate one. It will be assumed that all transitions in conflict are immediate.

The vector of visit ratios expresses the relative throughput of transitions in the steady state. The visit ratio $\mathbf{v}(t)$ of each transition $t \in T$ normalised for transition t_i , $\mathbf{v}^{t_i}(t)$, is expressed as $\mathbf{v}^{t_i}(t) = \dfrac{\chi(t)}{\chi(t_i)} = \boldsymbol{\Gamma}(t_i) \cdot \chi(t), \; \forall t \in T$, where $\boldsymbol{\Gamma}(t_i) = \dfrac{1}{\chi(t_i)}$ represents the *average inter-firing time* of transition t_i and $\chi(t)$ is the steady-state throughput of transition t.

The vector of visit ratios $\mathbf{v}$ exclusively depends on the structure of the net and on the routing rates when the t-semiflows of a PN system are assumed to be freely related [2]. Thus, the vector of visit ratios $\mathbf{v}$ normalised for transition t_i, $\mathbf{v}^{t_i}$, can be calculated by solving the following linear system of equations [2]:

$$\begin{aligned} \begin{pmatrix} \mathbf{C} \\ \mathbf{R} \end{pmatrix} \cdot \mathbf{v}^{t_i} &= 0 \\ \mathbf{v}^{t_i}(t_i) &= 1 \end{aligned} \tag{1}$$

where $\mathbf{R}$ is a matrix containing the rates $\mathbf{r}(t)$ associated to transitions in equal conflict.

B. Description of Algorithms

PeabraiN provides two main features which implement (i) an iterative algorithm for performance estimation based on linear bound computation [4], [8] and (ii) a heuristic method to distribute shared resources in order to enhance the system performance as much as possible [8].

The algorithm in Fig. 1 [4], [8] shows the main steps for computing an estimation of the performance, more precisely, an upper throughput bound in a more accurate way (i.e., more closer to the system performance) than other algorithms [2]. It needs, as input, the SPN system to analyse and the degree of precision ($\varepsilon > 0$) to be achieved, and returns the improved upper throughput bound. Such an algorithm is based on the computation of p-semiflows. Initially, it considers the p-semiflow with lowest throughput, and its associated subnet is called initial bottleneck. Then, in each iteration the subnet associated to the p-semiflow that is potentially more constraining than the others is added to the bottleneck, and

[1] In the sequel, we omit the transpose symbol in the matrix products for clarity.

Input: $\langle \mathcal{S}, \delta \rangle$, ε
Output: Θ

1: *Compute the slowest p-semiflow* $\mathbf{y}$ *(initial bottleneck) and its throughput* Θ;
2: $\Theta' = 2 \cdot \Theta$; {Initial Θ' value to force first iteration}
3: **while** $\frac{\Theta' - \Theta}{\Theta'} \geq \varepsilon$ **and** *there are p-semiflows to be added* **do**
4: *Given a p-semiflow* $\mathbf{y}$*, compute the next slowest p-semiflow* $\mathbf{y}'$ *connected to* $\mathbf{y}$
5: $\Theta' = \Theta$
6: $\Theta =$ *Compute throughput of subnet associated to the p-semiflow* $\mathbf{y}'$
7: **end while**

Figure 1. Iterative algorithm for computing upper throughput bounds.

Input: $\langle \mathcal{S}, \delta \rangle$, R, p_0, $budget$, c
Output: n

1: *Compute the slowest p-semiflow* $\mathbf{y}$ *(initial bottleneck) and its throughput* Θ;
2: **while** *there is money to be spent* **and** *we have not reached the process-idle place* **do**
3: *Compute the next resource increment by solving a LP problem*
4: *Given resource costs and a maximum budget, compute the cost of adding more resources in the way indicated by the solution of the previous LP problem, and store such a distribution in* n
5: **end while**
6: **if** *not all resources have been incremented* **and** *there is remaining money to be spent* **then**
7: *Report there is still money to be spent*
8: **end if**

Figure 2. Resource optimisation heuristic algorithm.

after that, the throughput is calculated. Note that such addition in each iteration is restricting the behaviour of the system, what implies a lower throughput. The iteration stops when no significant improvement of the bound is achieved.

The algorithm in Fig. 2 is a heuristic procedure for resource optimisation, it gauges the number of resources a system should allocate. More precisely, given a cost for each resource and a fixed budget, the algorithm computes how many instances of each resource are needed ensuring two premises: the budget is not exceeded, and the throughput is maximised. As input, it requires a PPN representing the system, the vector of resources R, the process-idle place p_0 (see Section II-A), the total $budget$ and a vector of costs c, which associates a cost to each resource in the PPN. As output, the algorithm provides which resources and in which quantity they have to be incremented.

C. Other Implemented Features

Lastly, `PeabraiN` adds other features to PIPE as side effect from the algorithms above mentioned:

- **Lower (upper) throughput bound.** The algorithms given in [7] for computing the lower (and upper) throughput bound for a SPN, in terms of LP problems, have been implemented. The PN structure needs to fulfil a set of conditions so that the computation of performance bound has some meaning, namely (i) the PN must be structurally live, (ii) structurally bounded, (iii) have a home state and (iv) its vector of visit ratios must have a unique solution. As some of these properties are already fulfilled depending on the PN subclasses, and moreover some of them are NP-decidability problems, we just automatically check the latter property.
- **Slowest p-semiflow.** The LP problem presented in [7] allows to compute the slowest p-semiflow of a PN, and its throughput which is an upper performance bound for the real system performance. The PN must fulfil the same conditions than in the previous algorithm.
- **Structural marking** and **structural enabling.** The structural marking of a place p, and the structural enabling of a transition t, can be computed by using LP problems [7]. Such algorithms work for any kind of PN.
- **Visit ratios.** The vector of visit ratios v of a PN, normalised for a transition $t \in T$ can be computed as described in Section II-A.
- **SPN Simulation.** A simulator for SPNs using the Gillespie's stochastic simulation algorithm [13] has been implemented. It performs a set of replications of the simulation, and estimates the average throughput with a given confidence interval level and error accuracy.

III. `PeabraiN` Framework Design and Features

`PeabraiN` is made of a set of modules compliant with PIPE-tool modules. As PIPE, `PeabraiN` has been implemented in `Java`, and it uses the same libraries as PIPE, and additionally, the Java Interface for LP solvers (Java ILP)[2] library, the Stochastic Simulation in Java (SSJ)[3] library, the Java Matrix (JAMA)[4] library for performing computational operations in matrices and LP solver-specific interface for Java. Hence, such a collection of modules perfectly fits in PIPE tool.

`PeabraiN` has been designed as a closed architecture by layers, i.e., each layer only calls methods of the immediate lower layer modules (see Fig. 3). Each of these three layers matches with each component of the Model-View-Controller (MVC) architectural pattern. It has been developed on the top of the Java Runtime Environment (JRE) and some other external libraries as indicated above.

The *data layer* contains classes representing the information needed for the algorithms to execute. For instance, the `PetriNetModel` class represents a PN in its matrix form, and implements several methods related to PN (such as getting the initial marking at a place or getting the rate of some transition). The rest of the classes in this layer represent

[2]http://javailp.sourceforge.net/
[3]http://www.iro.umontreal.ca/~simardr/ssj/indexe.html
[4]http://math.nist.gov/javanumerics/jama/

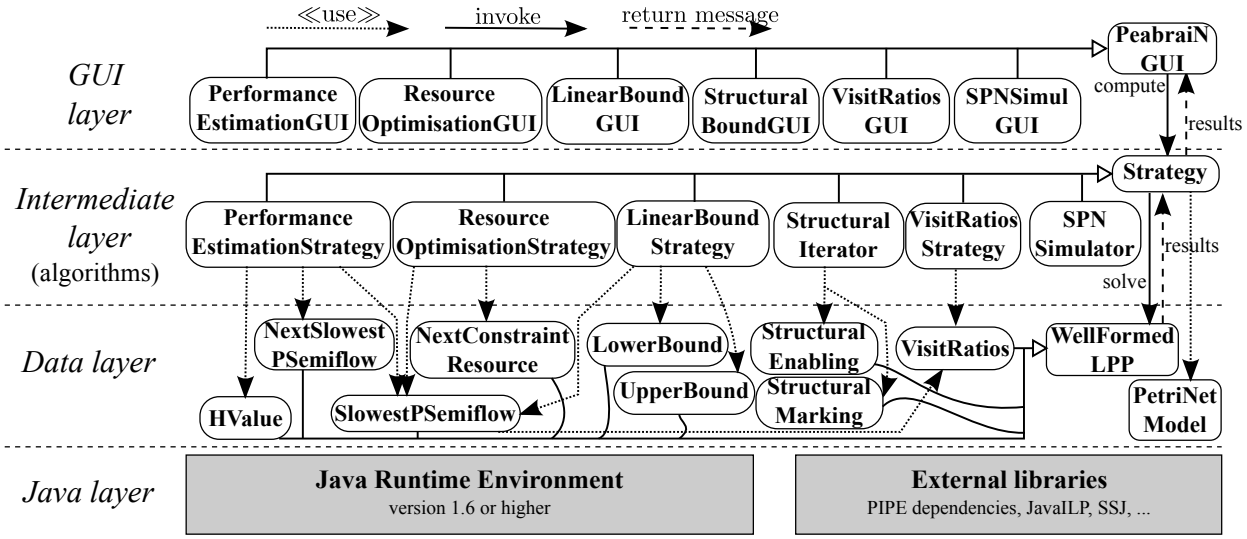

Figure 3. `PeabraiN` software architecture.

constraints, they are needed either for the LP problems of the algorithms in Figs. 1 and 2 or for the features presented in Section II-C. The `WellFormedLPP` class is a super-class of the rest of classes in this layer.

The *intermediate layer* encloses the classes that implement the algorithms and features explained in Section II. Solid arrows mean that a class invokes methods of another class, while dashed arrows represent the method return messages. For instance, `PerformanceEstimationStrategy` implements algorithm in Fig. 1, and invokes the LP problem for computing the slowest p-semiflow and the LP problem for computing the next slowest p-semiflow. `Strategy` is a super-class that allows all its child-classes to manage the `PetriNetModel` in matrix form. The classes in this layer call the `solve` method of the classes in the data layer through the `WellFormedLPP` class. The dotted arrows connect a class in this layer with the classes in the data layer that it actually uses.

Finally, the *GUI layer* has classes which create the graphic interfaces for collecting, from the user, information for execution of the algorithms and also to show the results. They invoke the classes in the intermediate layer. For example, Fig. 6(b) is an instance of the `ResourceOptimisationGUI` class, it allows to introduce the necessary parameters and after computation shows the results and the execution time.

Fig. 4 shows the integration of `PeabraiN` in PIPE. PIPE is extended through modules, and each module must implement the `IModule` interface. Besides, the open architecture depicted in Fig. 4 shows how the PIPE-data layer and `PeabraiN`-data layer are related. Each `PeabraiN` module creates a matrix representation of the current PN model, which is in PNML (PIPE format). We do not use PNML in our data layer because the algorithms work with the matrix representation.

Each `PeabraiN` module in the GUI layer in Fig. 4 will create an instance of the class with its same name in Fig. 3. For example, `PerformanceEstimationModule` creates `PerformanceEstimationGUI` to allow the user to introduce ϵ (input parameter in algorithm in Fig. 1). Fig. 5 illustrates the interactions between the user, PIPE, and `PeabraiN` when executing.

In brief, the new modules added to PIPE are:

- **Performance Estimation.** It needs as input the degree of precision, ϵ, to be achieved. Note that the lower the value of ϵ, the longer it takes to finish. The module reports about the components of the p-semiflow in each iteration step and its throughput, computed by simulation.
- **Resource Optimisation.** This feature enacts an optimal distribution of resources in a shared-resource PN for a given budget and resource costs, trying to optimise the system performance. It needs the process-idle place, that is, the place which represents the workload in the system (i.e., incoming customers or requests), the maximum of budget to be spent and the cost of each resource. Once the input data are validated, it computes and reports about the needed increment of resources, the rest of budget to be assigned and informs if there is more choice of improvement.
- **Linear Bound.** This feature allows to compute the upper and lower performance bound for a given transition and the slowest p-semiflow of the PN.
- **Structural Enabling.** This feature allows the computation of the structural enabling bound for a given transition or for all the transitions.
- **Structural Marking.** This feature allows the computation of the structural marking bound for a given place or for all the places.
- **Visit Ratios Computation.** It needs a transition, used as the normalised transition, for the visit ratios computation. As well as the result, information about the uniqueness of its solution is also given.
- **SPN Simulation Analysis.** Input data are either the

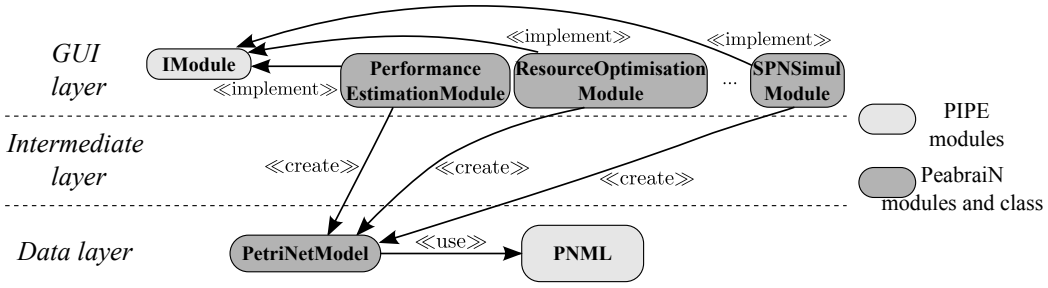

Figure 4. Integration of PeabraiN in the PIPE tool.

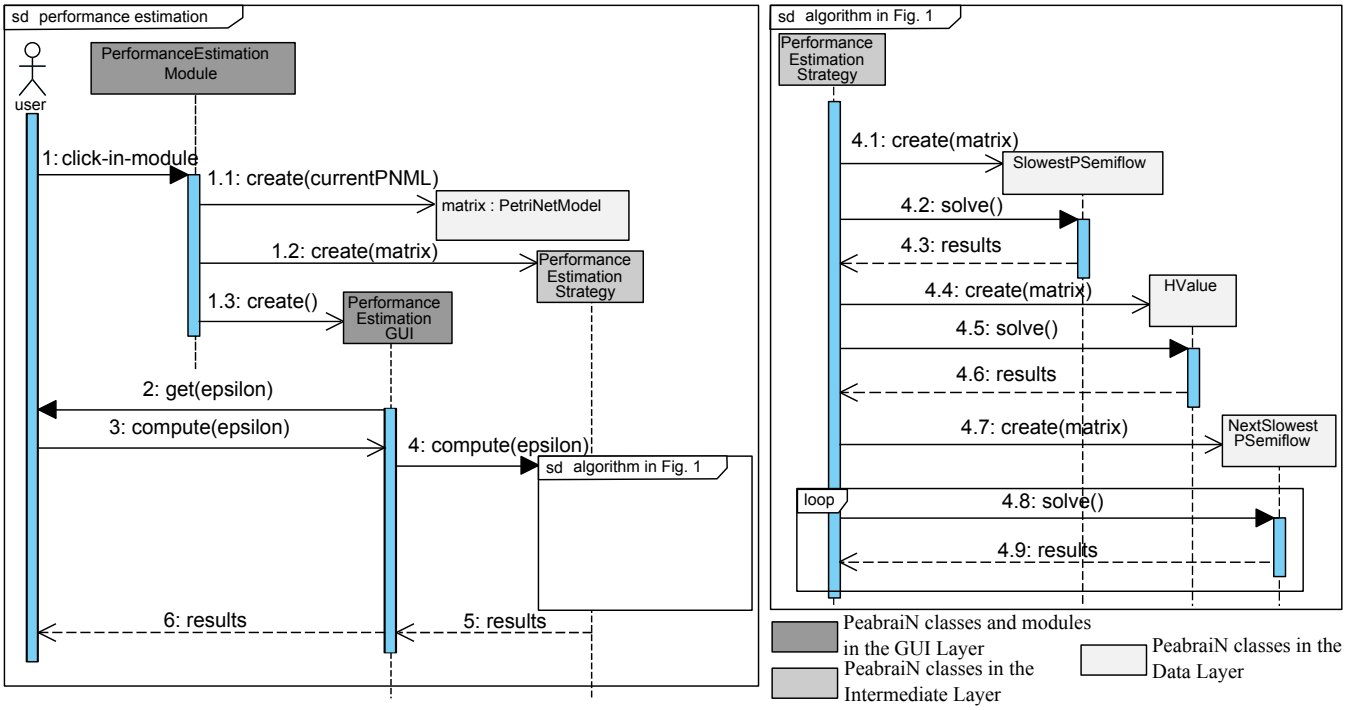

Figure 5. UML Sequence Diagram for executing performance estimation module.

maximum simulation time, or the confidence level and error accuracy to be achieved. When the simulation finishes, the module informs about the estimated throughput (computed for transition t_0 by default) of the PN, the confidence level, the error accuracy achieved and the execution time.

IV. Example on Using PeabraiN

Let us illustrate the use of PeabraiN by an example. Let us consider a pop-corn cinema store where once customers are attended and they have their pop-corn bucket, they are asked for some beverage. Fig. 6(a) depicts a PN modelling such a store. The PN marking represents the number nC of customers (initial marking of the process-idle place, p_0), the number nW of workers attending the customers (initial marking of p_2) and the number nD of beverage dispensers. The exponential transitions are represented by a white box, whilst immediate transitions are black boxes. The think time of the customers is represented by transition T_0, which follows an exponential distribution of mean δ_{T_0} minutes. The amount of time for attending a customer is represented by transition T_2, which follows an exponential distribution of mean $\delta_{T_2} = 2$ minutes. The customer's decision is represented by the place p_5 and its outgoing arcs: either transition t_4 is fired (then the customer does not want any beverage), or transition t_5 is fired. In the latter case, once some beverage dispenser is available, it is used. Such a use is represented by T_7 and takes, on average, $\delta_{T_7} = 1$ minute to complete. Finally, T_9 represents the customer's payment, which takes, in terms of time, about 2 minutes, i.e., $\delta_{T_9} = 2$.

Let us suppose an expected number of customers $nC = 10$, an initial budget of $\$80,000$, the cost of a new hiring is $\$6,000$ and a beverage dispenser has a cost of $\$250$. With this configuration, the resource optimisation procedure gives as result that 3 new hirings should be done in order to attend such an incoming number of customers and hence to maximise the performance. Fig. 6(b) depicts a snapshot of the results as reported by PeabraiN. It reports about the remaining

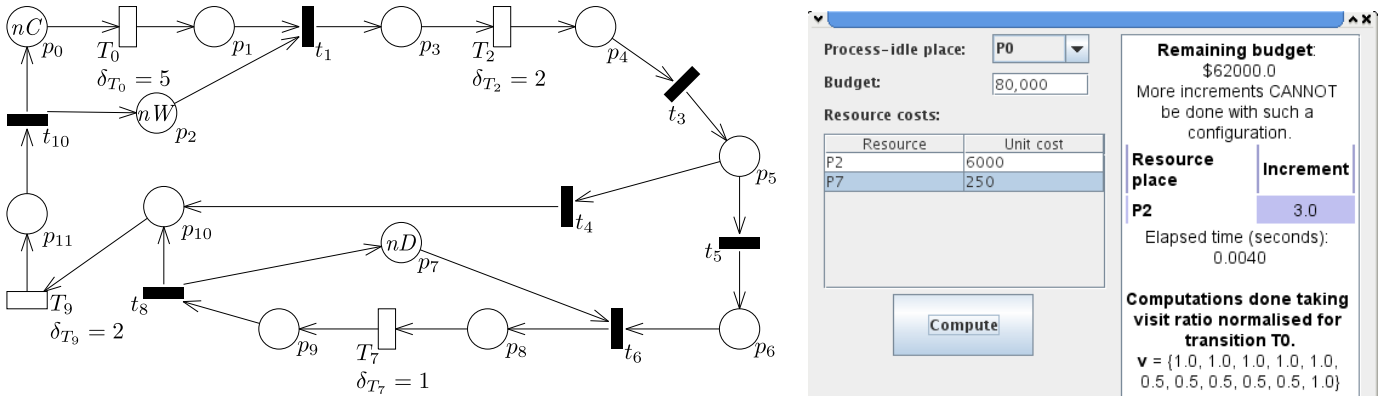

Figure 6. (a) Example of a pop-corn cinema store and (b) a snapshot of execution results (resource optimisation).

money to be spent, the number of instances of resources to be incremented and the elapsed time in computation. Besides, for this configuration it reports that no more further improvement can be done with the remaining budget. This means that even if we keep incrementing resources, the bottleneck of the system is the number of customers. Therefore, if resources are incremented they will be idle.

V. Tool Availability and Installation Requirements

`PeabraiN` has been developed with the Eclipse IDE under Linux environment, and successfully tested on Linux and Windows environments. `PeabraiN` needs to have installed in the host machine the following software to execute:

- a JRE version 1.6 (or higher); and
- an LP solver, namely the GNU Linear Programming Kit (GLPK) and its associated library for binding with Java.

Currently, we are working on an automatic detection of installed LP solvers in the host machine, and then automatically configure `PeabraiN` to work with them. Even `PeabraiN` is designed for working with several LP solvers, such as CPLEX or `lpsolve` among others, our initial thought was GLPK as it is free software under GNU General Public License (GNU GPL).

There exists a web page:

http://webdiis.unizar.es/GISED/?q=tool/peabrain

where further information about tool requirements and installation steps, tool binaries and sources can be found. `PeabraiN` is released under GNU GPL version 3 license.

VI. Conclusion and Future Work

We have developed `PeabraiN`, a collection of PIPE tool-compliant modules for performance estimation and resource optimisation based on bounds computation for Stochastic Petri Nets. Moreover, other features have been added to PIPE, such as structural enabling bound at a transition, structural marking bound at a place, visit ratios computation or SPN simulation analysis.

As future work, we plan to add the choice of LP solver by the user, the automatic detection of installed LP solver for automatic configuration of the tool, and to allow to change the simulation parameters, such as the transient observations to be discarded. Besides, we intend to perform more experiments with larger benchmarks to show its applicability, and to compare the performance of our tool with other broadly used tools, such as `GreatSPN` or `TimeNET`.

Acknowledgements

This work was partially supported by Fundación Aragón I+D, DGA (CONAID), and Spanish project DPI2010-20413.

References

[1] M. Molloy, "Performance Analysis Using Stochastic Petri Nets," *IEEE T. Comput.*, vol. C-31, no. 9, pp. 913–917, sept. 1982.

[2] J. Campos and M. Silva, "Structural Techniques and Performance Bounds of Stochastic Petri Net Models," *Lecture Notes in Computer Science*, vol. 609, pp. 352–391, 1992.

[3] Z. Liu, "Performance Bounds for Stochastic Timed Petri Nets," in *Proceedings of the 16th ICATPN*. Springer-Verlag, 1995, pp. 316–334.

[4] R. J. Rodríguez and J. Júlvez, "Accurate Performance Estimation for Stochastic Marked Graphs by Bottleneck Regrowing," in *Proceedings of the 7th EPEW*, ser. LNCS, vol. 6342. Springer, 2010, pp. 175–190.

[5] E. M. Goldratt and J. Cox, *The Goal: A Process of Ongoing Improvement*. North River Press, 1986.

[6] P. Bonet, C. Llado, R. Puijaner, and W. Knottenbelt, "PIPE v2.5: A Petri Net Tool for Performance Modelling," in *Proceedings of the 23rd Latin American Conference on Informatics (CLEI)*, Costa Rica, 2007.

[7] J. Campos and M. Silva, "Embedded Product-Form Queueing Networks and the Improvement of Performance Bounds for Petri Net Systems," *Performance Evaluation*, vol. 18, no. 1, pp. 3–19, July 1993.

[8] R. J. Rodríguez, J. Júlvez, and J. Merseguer, "On the Performance Estimation and Resource Optimisation in Process Petri Nets," *IEEE T. Syst. Man. Cy. A.*, submitted for publication.

[9] S. Baarir, M. Beccuti, D. Cerotti, M. D. Pierro, S. Donatelli, and G. Franceschinis, "The GreatSPN tool: recent enhancements," *SIGMETRICS Perform. Eval. Rev.*, vol. 36, no. 4, pp. 4–9, 2009.

[10] L. M. Hillah, E. Kindler, F. Kordon, L. Petrucci, and N. Tréves, "A primer on the Petri Net Markup Language and ISO/IEC 15909-2," *Petri Net Newsletter*, vol. 76, pp. 9–28, 2009.

[11] T. Murata, "Petri Nets: Properties, Analysis and Applications," in *Proceedings of the IEEE*, vol. 77, no. 4, April 1989, pp. 541–580.

[12] Z. Banaszak and B. Krogh, "Deadlock Avoidance in Flexible Manufacturing Systems with Concurrently Competing Process Flows," *IEEE T. Robotic. Autom.*, vol. 6, no. 6, pp. 724–734, dec 1990.

[13] D. T. Gillespie, "A General Method for Numerically Simulating the Stochastic Time Evolution of Coupled Chemical Reactions," *Journal of Computational Physics*, vol. 22, no. 4, pp. 403–434, 1976.