Design and Implementation of a WEB-based non-Markovian Stochastic Petri Net Tool

A. Horváth †, A. Puliafito *, M. Scarpa §, M. Telek †, O. Tomarchio *

* Istituto di Informatica e Telecomunicazioni
Università di Catania, 95125 Catania, Italy

§ Dipartimento di Informatica
Università di Torino, 10149 Torino, Italy

† Department of Telecommunications
Technical University of Budapest, 1521 Budapest, Hungary

* ap@iit.unict.it; § scarpa@di.unito.it; † {horvarth,telek}@hit.bme.hu

Abstract

This paper describes the design and the implementation of a new modeling tool for the analysis of non-Markovian stochastic Petri nets *(SPN)*. This tool, called WebSPN, provides a discrete time approximation of the stochastic behaviour of the marking process which results in the possibility to analyze a wider class of *PN* models with *prd, prs* and *pri* concurrently enabled generally distributed transitions. WebSPN relaxes some of the restrictions present in currently available packages thus widening the field of applicability of PNs. A Web-centered view has been adopted in its development in order to make it easily accessible from any node connected with the Internet as long as it possesses a Java-enabled Web browser. Sophisticated security mechanisms have also been implemented to regulate the access to the tool which are based on the use of public and private electronic keys.

Keywords: Non-Markovian *SPN*, transient analysis, Java technology, World Wide Web, security, discrete-time Markov chain.

1 Introduction

The analytical approach to the evaluation of systems is being increasingly viewed as an integral part of the process of design, analysis and tuning of computer systems. Analytical models give results whose accuracy depends on the designer's ability and on the level of detail of the model employed. Furthermore, once the model has been developed, its solution is generally very quick, so an accurate analysis of the system can be made with the variation of all the model parameters.

Special model specification techniques are needed that help analysts to describe their systems in such a way that the models can be understood at the level of the system designer, rather than at the mathematical level. Software environments that support these specification techniques for analytical models are needed. Such environments (tools) should allow for the easy specification and efficient solution of the models. Furthermore, they should allow for the control of the numerical solution of the models as well as for a suitable presentation of the results.

Petri nets are commonly viewed as a valid tool for the qualitative and quantitative study of systems. Many Petri nets modeling tools have been proposed or developed recently (e.g. ESP [7], GSPN [4], SPNP [5], DSPNExpress [14], TimeNet [10], UltraSAN [6]).

Some of the above tools have also implemented the possibility of including some non-Markovian features thus extending the range of applicability of PNs. Their main limitations regard the kind and number of generally distributed (GEN) transitions and their associated preemption policy. A very limited number of simultaneously enabled GEN transitions is allowed. And usually it reduces to only one. Further, the *preemptive repeat different* (*prd*) policy is the only adopted. The *preemptive resume* (*prs*) and the recently proposed *preemptive repeat identical* (*pri*) policies [2], although very powerful, are not yet implemented. The first restriction can be relaxed by the analytical results available for the analysis of *PN* with non-overlapping *prs* general transitions [3], and there is an active research to find the proper way to analyze *PN* with concurrently active general transitions [14, 15].

Some analytical results for the analysis of non-Markovian PNs have been recently presented which make use of Markov regenerative theory but, as far as we know, an automatic procedure based on this approach has not been produced yet. The only possible approach for the analysis of *PN* models, with *prs* and *prd* general transitions, is the Phase type *(PH)* approximation. With this technique, the marking process of the non-Markovian *SPN* is approximated by an expanded Markov chain. The main drawback of the *PH* approximation consists in the very large state space of the expanded Markov chain, mainly if the random firing times have a low coefficient of variation. The *pri* policy is not captured with the *(PH)* approximation.

In this paper we present a new modeling tool for the analysis of non-Markovian stochastic Petri nets that relax some of the restrictions present in currently available modeling packages. This tool, called *WebSPN*, provides a discrete time approximation of the stochastic behaviour of the marking process which results in the possibility to analyze a wider class of Petri net models with *prd, prs* and *pri* concurrently enabled generally distributed transitions. A Web-centered view has been adopted in its development in order to make it easily accessible from any node connected with the Internet as long as it possesses a Java-enabled Web browser. WebSPN also provides sophisticated security mechanisms to regulate the accesses which are based on the use of public and private electronic keys.

The rest of the paper is organized as follows: in Section 2 we will review the main concepts of non-Markovian stochastic Petri nets and introduce the *prd, prs* and *pri* preemption policies. Section 3 briefly outlines the proposed discretization approach. Section 4 shows how it is possible to use the Java technology for the Web sharing of

WebSPN. Section 5 provides an application example which is solved through WebSPN and reports some comparative numerical results. Conclusions are given in Section 6.

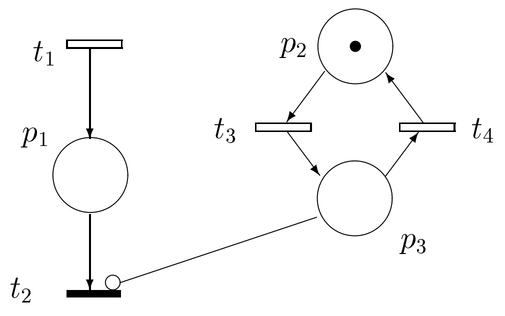

Figure 1: Petri net model of one server

2 Introducing Petri Nets and Preemption Policies

Here we give only a brief definition of timed Petri Net with generally distributed transitions, and an intuitive explanation on the behavior of timed transitions in precence of different kind of memory policy.

A timed Petri net is a tupla PN=$(\mathcal{P}, \mathcal{T}, \mathcal{G}, \mathcal{A}, \mathcal{I}, \mathcal{O}, \mathcal{M})$ where: $\mathcal{P}$ is the set of places; $\mathcal{T}$ is the set of transitions; $\mathcal{G}$ is the set of random variables γ_g associate to transitions; $\mathcal{A}$ is the set of age variables a_g associate to transitions; $\mathcal{I}, \mathcal{O}$ and $\mathcal{H}$ are respectively the set of input, output and inhibitor functions ($\mathcal{I} \subset \mathcal{P} \times \mathcal{T}$, $\mathcal{O} \subset \mathcal{T} \times \mathcal{P}$, $\mathcal{H} \subset \mathcal{P} \times \mathcal{T}$), providing their multiplicity; $\mathcal{M}$ is the set of marking M_i: a marking is a tupla, whose cardinality is $||P||$, recording the number of token in each of the place in P.

A transition $t \in \mathcal{T}$ is enabled when the number of tokens in each input place is greater than multiplicity of input arcs, and the number of the tokens in each inhibitor input place is less then multiplicity of inhibitor arcs. The firing of an enabled transition removes as much tokens as the multiplicity of input arcs from input places, and adds as much tokens as the multiplicity of output arcs to the output places. The firing of an enabled transition, in a given marking M_i, generate another marking M_j. M_j is said directly reachable from M_i ($M_i \rightarrow M_j$).

Starting from an initial marking M_0, the transitive closure of $\rightarrow$ generate the reachability graph $RG(M_0)$ (the set of all reachable marking from M_0).

A consistent way to introduce memory into a *SPN* is provided in [1] and extended in [3]. Each timed transition t_g is assigned a general random firing time γ_g with a cumulative distribution function $G_g(t)$. A clock, associated to each individual transition, counts the time in which the transition has been enabled. An *age variable* a_g associated to the timed transition t_g keeps track of the clock count. A timed transition fires as soon as the memory variable a_g reaches the value of the firing time γ_g.

A timed transition has to be characterized both in terms of the distribution function of the random firing time and also of its behaviour when a preemption occurs. Thus, a preemption policy is required to fully describe the behaviour of a timed transition. In this paper we prefer an informal approach to the definition of preemption policies through the example of Figure 1. A more rigorous definition can be found in [19].

The Petri net on Figure 1 models a server with exponential arrivals (transition t_1) and general service time (transition t_2). Waiting customers are represented by the tokens in place p_1. The server is randomly preempted by higher priority jobs (transition t_3) for an exponentially distributed amount of time (transition t_4), as shown by the inhibitor arc from place p_3 to transition t_2.

When a customer arrives to a server, a specific service requirement γ_g has to be completed. The amount of computation required is sampled from the distribution function $G_g(t)$ of the service time. The optimal case is when the server is able to complete the job before an interruption occurs. However, the server may be interrupted after having processed only a portion of the submitted job. In this case the whole behaviour is strongly affected by the preemption policy and the whole performances will depend on the strategy adopted to deal with the preempted job, as described in the following:

- The server drops the customer it was dealing with before the interruption. It means that it looses the portion of work a_g already completed. It starts with a new customer which has a new work requirement, i.e. a new sample from the same distribution is taken. Of course, the server starts serving this new customer from the beginning.

- The server goes back to the preempted customer who still maintains the original work requirement γ_g. The server does not loose the portion of work a_g already executed, but it resumes the work from the point it was interrupted.

- The server also returns to the same customer who still has the same work requirement γ_g. But this server looses the portion of work a_g already completed and starts the service of the same customer from the beginning.

According to [19], the previous policies are referred to as *preemptive repeat different (prd)*, *preemptive resume (prs)* and *preemptive repeat identical (pri)*, respectively. Note that in [1] the authors indicated the *prd* and *prs* type policies as enabling and age type. The *pri* policy was introduced for the first time in [2]. The *prd* policy is the only considered in the available tools modeling non-Markovian *SPN* [14, 10, 6].

From the previous discussion it is clear that the main difficulty in analysing stochastic Petri nets with general transitions is related to the fact that the underlying discrete state marking process is not a CTMC anymore, as its future evolution depends on the past history. This is true also if the Petri net model contains only exponential transitions, but a *pri* policy is adopted in a preemptable one. In this case the memoryless property is destroyed due to the fact that the previously sampled value has to be maintained. This is why in the following we indicate as general (MEM) transitions both the generally distributed transitions (including the deteministic ones) and the exponentially distributed transitions of *pri* type. For a transition with exponentially distributed firing time the *prd* and the *prs* policies have the same effect, due to the memoryless property. We denote these transitions as EXP transitions.

3 Algorithm description

In this section we provide a brief outline of the discretization approach implemented in WebSPN for the transient and steady state analysis of a non-Markovian SPN. This presentation is intented to give an intuitive explanation of the algorithm; a detailed description can be found in [13].

The algorithm we have developed is based on a time discretization approach which allows to deal with the *prs*, *prd* and *pri* preemption policies previously discussed.

The approximation of the continuous time model at equispaced discrete time points involves the analysis of the system behaviour over a time interval based on the system state at the beginning of the interval and the past history of the system. The length of the constant discretization interval must be chosen as a function of the random time variables of the model in such a way that the piece-wise discretized functions provide a sufficiently accurate approximation of the corresponding continuous functions.

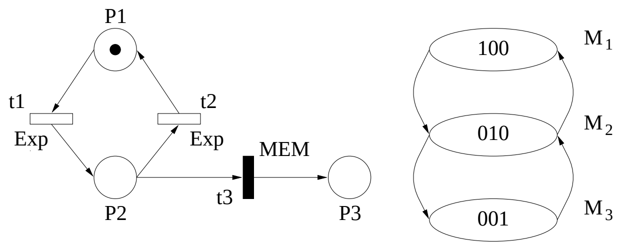

Figure 2: PN with one MEM transition

Let consider the Petri net shown in Figure 2. This Petri net model represents a system that moves between two conditions: a fully operative state (token in place $P2$), where some work is produced, and a failure state (token in place $P1$), where the system does not produce any work. Transition $t3$ models the time required for completing all the work submitted to the system, and it is assumed to be generally distributed. The EXP transitions $t1$ and $t2$ are used to describe the changes in the system state from operational to failed, and vice versa.

The algorithm assumes a Discrete Time Phase Type (DT-PH) as an approximation of the firing time of each MEM transition. In particular Figure 3 shows the DTPH expansion used to approximate the stochastic behavior of the MEM transition $t3$. There is not any particular reason to use this specific DTPH; it is assumed only to explain the way this technique works.

Based on the Petri net behavior, the algorithm generates a *discrete time* markov chain (DTMC) which approximate the *continuous* marking process. When transition $t3$ has a *prd* or *prs* policy, the state of the chain is a couple (i, u) with i the index of a marking $(M_i \in RG(M_1))$ and u a phase of the DT-PH approximation of the MEM transition. Thus, u is used to capture the "memory" effect that is necessary to model the general distribution. $u = \diamond$ denotes that the process is in a state where the general transition is disabled (i.e. it has no memory); taking into exam the DT-PH expansion of Figure 3, transition $t3$ is enabled if $1 \leq u \leq N = 4$.

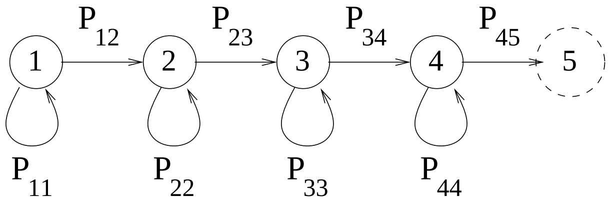

Figure 3: The DT-PH approximation of the firing time of t_3

Figure 4 gives the DTMC that is constructed to approximate the stochastic behaviour of the net when $t3$ is of *prd* kind. The chain is derived from the reachability graph. State $(1, \diamond)$ corresponds to the initial marking (100) shown in Figure 2, in which the MEM transition is not active.

When the marking process is in state $M_1 = 100$, only transition $t1$ is enabled and, assumed an appropiately small time step δ, only two events are possible: transition $t1$ fires with probability $\delta\lambda_1$, transition $t1$ does not fire with probability $1 - \delta\lambda_1$. The former event is approximated in the DTMC with the transition $(1, \diamond) \to (2, 1)$, the latter with transition $(1, \diamond) \to (1, \diamond)$.

In marking $M_2 = 010$, transitions $t2$ and $t3$ become enabled. We need more states to model the behaviour of the net in this marking because of the generally distributed transition $t3$, and the DT-PH model is used for this purpose.

From each state $(2, i)$ in the DTMC, the discrete process moves to a state carachterized from a different index marking due to the firing of an enabled transition. If this event does not happen a transition $(2, i) \to (2, j)$ between two states of the DT-PH occurs with probability $P_{ij}(1 - \delta\lambda_2)$, which considers the event that neither the Exp transition $t2$, nor the MEM one fire in a time-interval of lenght δ.

If transition $t2$ fires, with probability $\delta\lambda_2$, the DTMC transits into state $(1, \diamond)$. If transition $t3$ fires, with probability $P_{i,5}(1 - \lambda_2\delta)$, the process reaches state $(3, \diamond)$.

This consideration does not take into account the case when in marking M_2 both the enabled transitions, $t2$ and $t3$, fire in a δ interval. The probability of this event can be considered by splitting this quantity between arcs from state $(2, 4)$ to states $(1, \diamond)$ and $(3, \diamond)$, because these two transitions are in conflict as shown in Figure 4.

State $(3, \diamond)$ is an absorbing one because there are not transitions is enabled in marking M_3.

If a preemptive resume policy is adopted, the DTMC structure has to be organized in order to keep track of the amount of time the *prs* transition was enabled before being preempted. This is because the transition has to restart from the same point once it becomes enabled again. For this purpose we have to expand the state space of the DTMC. Figure 5 shows the DTMC that approximates the behaviour of the net depitched in Figure 2 when $t3$ implements a *prs* policy. It can be observed that as many states corresponding to marking (100) are needed as many phases the DT-PH structure has. In fact, we have to guarantee that if the MEM transition becomes disabled in the u-th phase of the DT-PH structure, the process has to enter the same phase once the transition gets enabled again.

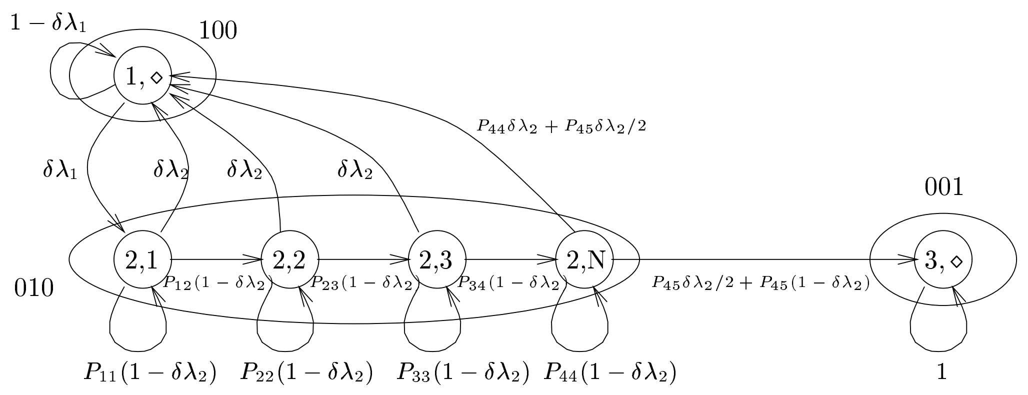
$1-\delta\lambda_1$
100
$1,\diamond$
$P_{44}\delta\lambda_2 + P_{45}\delta\lambda_2/2$
$\delta\lambda_1$
$\delta\lambda_2$
$\delta\lambda_2$
$\delta\lambda_2$
001
010
2,1
2,2
2,3
2,N
$3,\diamond$
$P_{12}(1-\delta\lambda_2)$
$P_{23}(1-\delta\lambda_2)$
$P_{34}(1-\delta\lambda_2)$
$P_{45}\delta\lambda_2/2 + P_{45}(1-\delta\lambda_2)$
$P_{11}(1-\delta\lambda_2)$
$P_{22}(1-\delta\lambda_2)$
$P_{33}(1-\delta\lambda_2)$
$P_{44}(1-\delta\lambda_2)$
1

Figure 4: DTMC approximation of the Petri net on Figure 2 with *prd* transition

Probabilities associated to the arcs are computed similarly to the *prd* case.

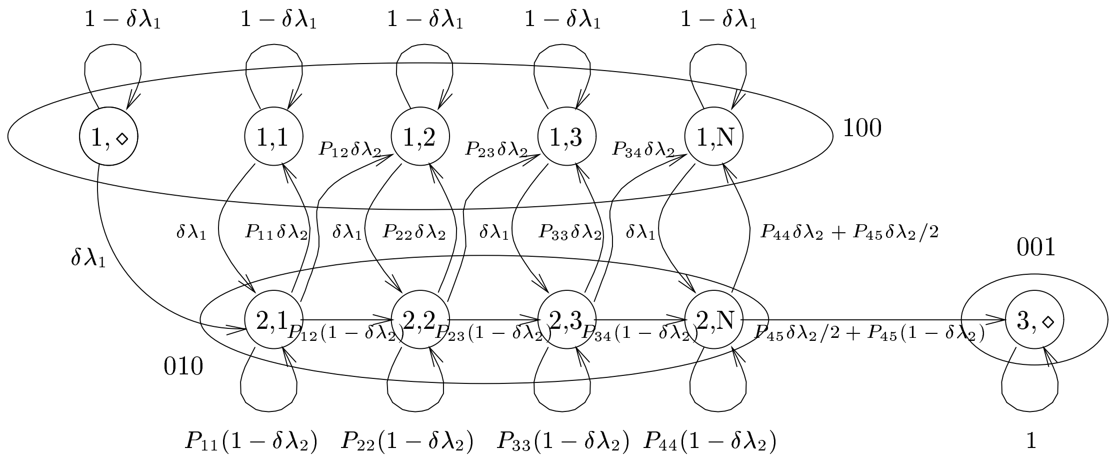

Figure 5: DTMC approximation of the Petri net with *prs* transition

If (i, u) is the current state of the DTMC and the MEM transition is disabled in marking M_i, the chain enters the u-th phase of the DT-PH structure once the transition is enabled again.

If a *pri* policy is assumed, an interrupted job must be repeated with an identical work requirement. To cover this case, a further expansion of the state space is required than in the case of *prs* policy.

The tuple (i, u, w) is used in order to describe the current state of the process where i indicates a marking, u is the age variable, and w is the sampled value. The tuple $(i, 0, w)$ means that the *pri* transition was disabled in one of the states of the w-th column, and after getting enabled again the process will enter the first state of this column. The tuple $(i, \diamond, \diamond)$ is used for the states where the process has no memory, in other words the marking itself completely determines the state of the process.

Figure 6 shows the DTMC approximation of the Petri net shown in Figure 2. The process starts from the state in the top of the figure (state $(1, \diamond, \diamond)$). If the *pri* transition becomes enabled, the associated random variable is sampled and the process enters the first state of one of the columns corresponding to marking (010). For example, if the sampled value of the firing time is 4δ (its probability is P_4), then the next state will be the first state of the column that has 4 states. The transition will fire if it is not disabled in the subsequent 4 time slots. If it gets disabled, the process enters the state shown in the figure to the left of the column corresponding to marking (100). Once the *pri* transition gets enabled again, the process enters the first state of the column that was previously left, thus guaranteing that the random variable associated to the firing time of the *pri* transition will not be resampled.

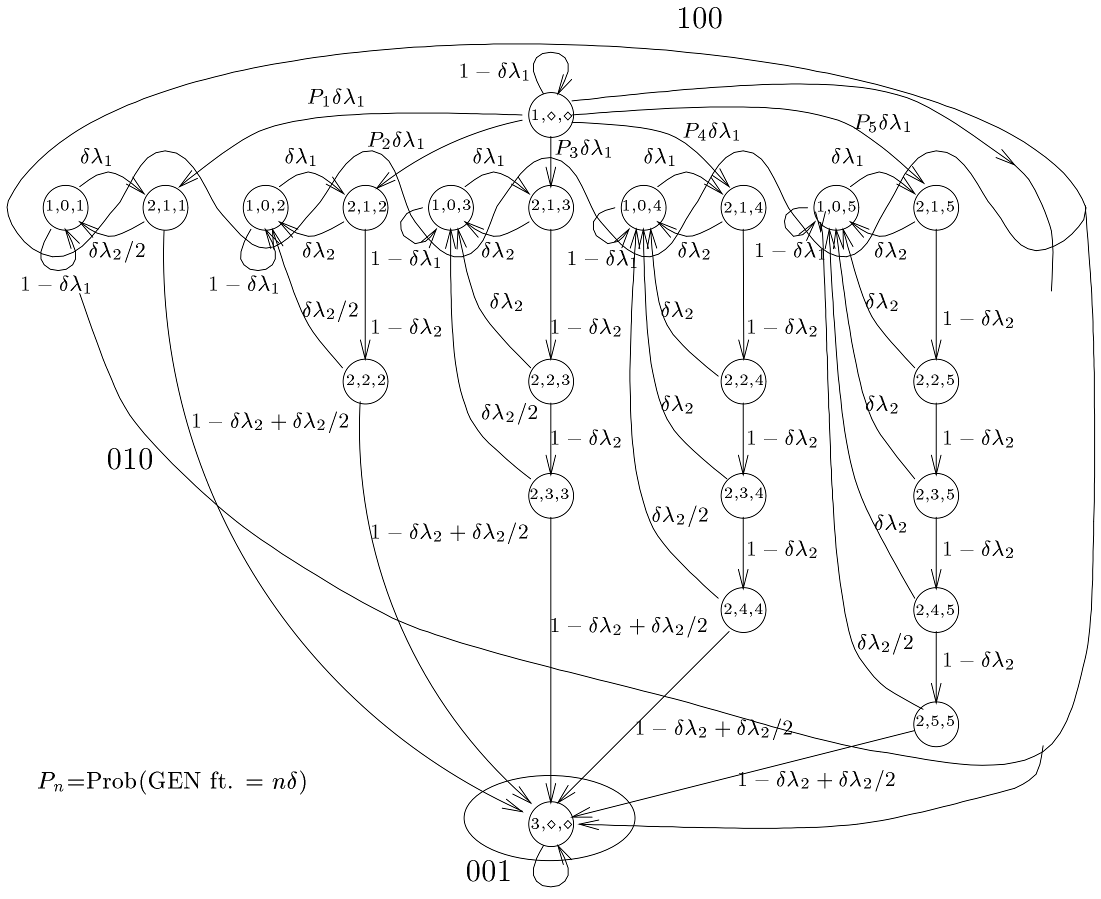

Figure 6: DTMC approximation of the Petri net with *pri* transition

Based on these considerations an elementary step of the approximation method is as follows:

- analysis of the behavior of the marking process inside a single time interval. This analysis is based on the the associated age variables at the beginning of the interval and on the state reached by the system at the end of the previous interval;
- evaluation of the values of the associated variables at the end of the current time interval.
- determination of the transition probabilities over the reachable states of the discretized state space at a fixed time (the lenght of the discretization interval).

The system behaviour is approximated by a Discrete Time Markov Chain (DTMC) over an expanded state space determined by the cross product of the system states (the markings of the Petri net) and the discretized values of the associated age variables. This approach is very similar to the *PH* expansion method [7] in several senses. The main difference is that, in this case, the system behavior is approximated by an expanded DTMC while in the *PH* approximation case an expanded CTMC is obtained. The present approach inherits some similarities also from the supplementary variable approach [9], since the supplementary (age) variables are constrained to assume values in a discretized set.

The main steps of the implemented solution method are the following:

- generation of the reachability graph (with tangible and vanishing states), and reduction of the reachability graph to tangible states, only;
- generation and analysis of the expanded DTMC;
- evaluation of the final measures at the net level, based on the solution of the expanded DTMC.

A detailed description of the discretization algorithm is provided in [13].

4 WebSPN: Design Issues

In this section we present the tool WebSPN which provides an implementation of the discretization approach described in the previous section and allows to analyze a wider class of *PN* models with *prd, prs* and *pri* concurrently enabled generally distributed transitions. WebSPN adopts a Web-centered approach in order to make it easily accessible from any node connected with the Internet as long as it possesses a Java-enabled Web browser [12, 20].

To make the tool available only to authorized users, adequate security mechanisms based on public and private key algorithms are included, which provide authentication services and encryption. The approach proposed was successfully used to port the Sharpe tool (Symbolic Hierarchical Automated Reliability/Performance Evaluator) onto the Web [17, 16]. WebSPN is available at the following site: *http://sun195.iit.unict.it/∼webspn/webspn2/*

4.1 Communication Mechanisms

The approach we followed can be seen as an extension of the client/server programming paradigm. The client, in fact, (1) processes locally the request to be sent to the server, who (2) executes it at a different time, at the end of which (3) it notifies the client of the results of the calculation. Unlike the classical approach, however, the client does not need to possess any specific software; through a simple Web interface it loads the software using the mechanisms provided by Java. In fact, the WebSPN user interface is implemented as a Java applet which can be executed using a common Java-enabled Web browser. Besides the graphical interface, the applet contains all the modules necessary for future communication sessions. The immediate advantage is the simplicity of access to the application and the total absence of a preliminary phase to distribute and install the interface software. The application provider can update the application, modifying the interface as he likes, without having to provide potential clients with updated versions of the software. The user, in turn, has the certainty of always using the latest version of the software, and can also count on optimal installation and an execution speed that is not always possible with his own computing resources. The immediate retrievability on the Web ensures the complete availability of applications that otherwise would probably remain known only to a limited number of potential users. In this sense, an immediate use for the design choice we propose is in teaching, to allow students easy, economic access to the modeling tool.

The only requirement for the client is a Java-enabled Web browser, while the server needs the following software modules:

- Web server;
- Java Virtual Machine;
- Solution engine of WebSPN;
- Java applet of the user interface;
- Software module to run the communication session with the client.

The last module, entirely developed in Java, comprises two submodules. One of these, in particular, is transferred to the client when the latter forwards a request for access to the server and provides the client with the mechanisms needed to run the communication sessions just started. The second submodule, on the other hand, is always in execution on the server and deals with accepting requests from various clients, robustly managing the various connections with clients, and sending clients the results put out by the server. It also keeps a memory of the correspondence between clients and the applications they use.

4.2 Security and Access Control

Network sharing of WebSPN requires the design of suitable security mechanisms. Two kind of problems have to be solved on the client and the server side.

On the client side, the user wants a reliable way to check where a piece of software is really coming from, who has created it and whether it has been changed through the network by some intruder. The user wants to know the kind of actions that the code he/she donwloaded from the network will execute on his/her machine, in order to evaluate the risk level associated with its execution. On the server side we want to control users' accesses in such a way to deny access to unknown or not authorized users and to monitor users which are using the tool (for statistical or accounting purposes).

Security techniques based on public and private keys can be applied to solve these kind of problems [18]. First version of Java did not include such authentication mechanisms. Security model in JDK 1.0.2 version follows the so called *sandbox model* [11]: untrusted code received from the network (such as applets) runs in a restricted environment and it has not full access to local system resources (e.g. the file system). JDK 1.1 provides the *applet signing* mechanism which allows software developers to digitally sign an applet using their private key [11]. A user wishing to execute an applet can verify its signature using the public key of the applet author in order to decide to accept the applet as trusted. Once an applet is trusted it can run with full rights of execution, as it was local code. Using this model users can selectively give trust to some applet and deny to some others. Besides, users can be sure of the applet integrity: in fact, they can detect if it has been modified or damaged through the network.

The JDK contains the necessary tools and API for code signing, including support for digital signatures, message digest and certificate manipulation. This enabled us to satisfy all security needs on the client side. Unfortunately, the security API so far described are available only using the *appletviewer* provided with the Sun JDK : common Web browsers such as Netscape Communicator or MS Internet Explorer support different kinds of security API.

Due to this incompatibility among different security model, we had to integrate some different technologies to satisfy the requirements on the server side. In particular, we set up a Web server with SSL functionalities [8] in order to keep confidential some information flowing on the net. To use the solution engine of our tool (which is executed on the server), a user is asked to register himself through a registration form on our Web site. The server associates to the user a *digital ID* which will send back to him. This registration phase is managed through a SSL connection to the Web server; data flowing in the network are crypted in such a way that nobody else can read them.

Successively, when the user launch the applet execution, he has to present this digital ID to our server in order to access to full capabilities of the application. In this way, on the server we are able to give rights of execution to users regularly registered, monitoring at the same time users which are currently running the application. As educational institution, we don't ask the user to pay for use of these services; however a private organization could use these mechanisms for making users pay according to the use of the service.

A user who has not registered, can download the applet and execute it, but he cannot use the computational part of the application. In fact he has no rights to execute anything on the server without explicit authorization.

4.3 The Main Features of WebSPN

To load the graphical user interface of WebSPN it is necessary to link up with the Web page containing the link for the applet and click on the relative icon: the subsequent loading and execution of the interface onto the local machine is quite transparent to the user. Using the new internationalization support of Java, the user is allowed to select the language he/she prefers. Italian, Spanish, English and French are currently available.

The main WebSPN display, shown in Fig. 7, has the following four zones: a) *Menu panel*; b) *Control panel*; c) *Design area*; d) *Status panel.*

The Menu Panel offers the usual choices. Besides the submenus *File* and *Edit* there is also the *Draw* submenu with items which allow the user to select the graphic

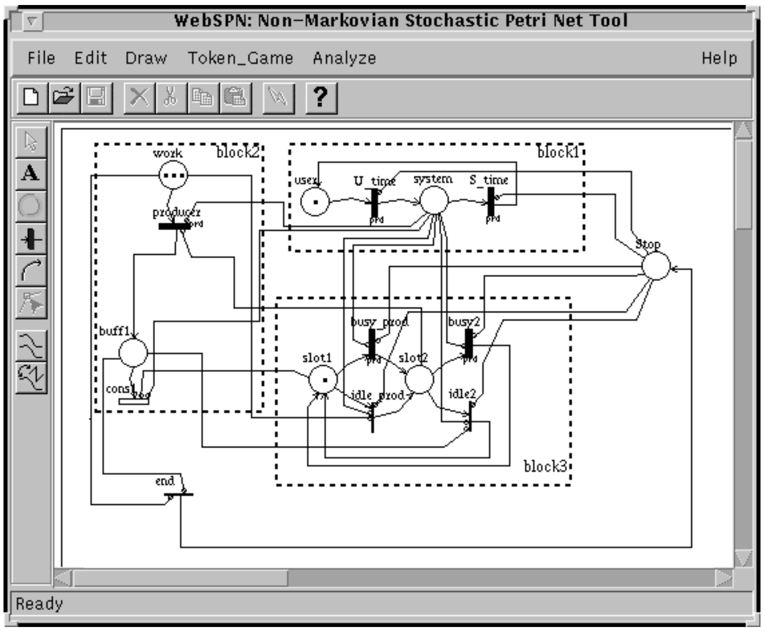

Figure 7: Main display of WebSPN

primitives to be used in the specification phase. More immediate use of the graphic functions is provided by a series of push buttons on the left hand side of the display. The *Token Game* submenu allows to start the token game execution to verify the logical behavior of the model. The transitions enabled in a given marking are highlighted in green color and the user is allowed to click on the one to be fired to check the logical evolution of the system.

The Control Panel can be used to activate a series of functions to create, load and save a model (there are also the classical cut, copy and paste functions) and others to activate the solution of the model and to add some text. An *Help* button is also provided which activates a fully-operative *html* browser to scroll the help or to surf in Internet.

The Design Area is where the user is allowed to draw his model. Significant graphic symbols and the associated dialogue boxes are enabled.

The Status Panel gives run-time indications regarding the status of the interface, signalling the occurrence of any event that may be of interest to the user in the language previously selected.

With a double click on a generic primitive, the user gains access to a dialogue box with which it is possible to specify the properties of the primitive.

In the case the primitive is a place, a name can be assigned as well as the initial number of tokens. If a transition is selected, then a dialog box will appear which offers the possibility to change its default name and orientation and to specify its timing (immediate, exponential or general). If the transition is defined as generally distributed, then the following choices are available: exponential *pri*, deterministic, uniform and

Weibull. The user is also allowed to give as input an *X-Y* file with the sampled values of the distribution function. Moreover, for any general transition a memory policy (among *prd, prs* and *pri*) must be assigned.

Once the specification stage is completed, the user passes to the analysis stage simply by pressing the *Analyze* key which opens a dialogue box in which the user specifies the evaluation indices he wants.

The specification of the measures is self-guided and the possibility of errors is reduced as the user is mainly required to use the mouse to compose his queries. Reward can be assigned to the markings and average and accumulated measures can be asked both at steady state and in transient conditions.

Pressing the *Ok* button the graphic representation is converted into text format using an internal specification syntax. The ASCII file thus created is then transferred to the server where the management module begins execution of a new WebSPN instance. Once the processing is completed the results are transmitted back to the client. A fully integrated environment is also provided to plot the results which provides all the features commonly available in similar plotting packages.

Another possibility offered by WebSPN is saving the graphic description of a model as well as of a plot with results in the *Xfig* format. *Xfig* is a public domain tool for vector graphics on Unix, frequently used in academic environments. The picture in this format can be easily manipulated on different platforms and converted into one of the many available graphic formats (e.g., PostScript).

5 Application example

In this section we describe and solve a model of Petri net with several concurrently enabled general transitions and different memory policies. This model does not represent a system in particular, and is mainly used for proving the potentialities of the tool WebSPN. However, as we pointed out below, this model can be used as a basis for analyzing different types of systems such as transactional databases, manufacturing systems and client/server communication systems.

5.1 Model description

The system moves between an operative phase, when useful work is produced, and a phase of maintenance when the processing capacity is temporarily interrupted, and that therefore does not contribute to increase the produced work. The task of the system in the operative state is to process a certain number of jobs. The execution of each job consists of two sequential phases: the first one executes a pre-processing of the job, while the second one completes the processing. The system can either pre-process a job or process another one, by alternately executing the two types of execution for certain time slots. Our target is to analyze such system, in order to obtain the probability distribution of the time required for completing a fixed number of jobs, and to obtain a measure of the productivity of the system according to its operative phases.

The *Petri net* shown in figure 7 represents the model of the system that, according to the description, consists of three functional blocks generically referred to as *Block1, Block2* and *Block3*. Block1 models the alternation of the system between the operative phase and the maintenance one. Block2 models the two sequential phases of processing of jobs. Finally, Block3 models the alternation of the system during the operative phase between the phase of pre-processing and the one of processing of jobs.

Within Block1, the two states of operation where the system can be are represented by places *user* and *system* and by transitions *U_time* and *S_time*. A token in place *user* denotes the operative state, while a token in place *system* denotes the maintenance one. The duration of the operative phase is denoted by transition *U_time*, while the maintenance one is denoted by transition *S_time*. The inhibitor arcs outgoing from place *system* and leading to the timed and immediate transitions contained in Block2 and Block3 *producer, cons1, busy_prod, idle_prod, busy2, idle2* are used for interrupting the activity of the system during the phase of maintenance.

Block2 models the processing of jobs. In particular, the amount of jobs to be processed is denoted by the number of tokens contained in place *work*, while the time of pre-processing of each job is represented by transition *producer*. Pre-processed jobs are therefore queued in a buffer (place *buff1*) waiting for the second phase of processing (transition *cons1*).

In Block3, the alternation between the phases of pre-processing and processing of jobs is represented through places *slot1* and *slot2* and transitions *busy_brod, busy2, idle_prod, idle2*. A token in place *slot1* denotes that the system is executing the pre-processing of a job, while a token in place *slot2* denotes the execution of a phase of processing. An inhibitor arc between *slot1* and *cons1* deactivates the phase of processing when the pre-processing one is active. The same way, the inhibitor arc between *slot2* and *producer* deactivates the phase of pre-processing when the processing one is active. The time that the system alternately spends for these two activities is represented by transitions *busy_prod* and *busy2*. The immediate transition *idle_prod* (*idle2*) prevents the system to remain in phase 1 (2), even if no job has to be processed. The function of the inhibitor arcs from place *work* to transition *idle_prod* and from place *buff* to transition *idle2* is to enable such transitions when no job has to be processed in the corresponding phase of processing. For example, let us consider the inhibitor arc outgoing from place *work* and leading to transition *idle_prod*. If place *work* contains at least one token, transition *idle_prod* is inhibited, so when a token is present in place *slot1*, the system pre-processes jobs for a time fixed by transition *busy_prod*. The inhibitor arc from place *slot1* to transition *cons1* is used for interrupting the processing activity. When transition *busy_prod* fires, the token in place *slot1* is moved to place *slot2*. If still no job has to be processed (no token in place *buff*), transition *idle2* is enabled and fires immediately, and the system pre-processes a new job.

Immediate transition *end* and place *Stop* are used for modeling the processing of all the jobs assigned to the system at the beginning. In fact, transition *end* is inhibited until at least one token is present in places *work* and *buff*. When all the jobs have been processed, transition *end* fires, and immediately moves a token to place *Stop*. All the activities of the system are thus interrupted through the inhibitor arcs outgoing from place *Stop*.

The measures that we are going to evaluate from the analysis of the model are:

- the probability distribution of the time required for completing the set of jobs assigned to the system at the beginning;
- the productivity of the system, referred to as time trend of the average level of work produced.

The first measure corresponds to the probability distribution of having a token in place *Stop.* Conversely, the productivity of the system can be obtained by appropriately assigning some rewards to the different states of operation of the system. In fact, the system executes useful work if it is in one of the two following conditions:

- **case 1:** mark[user]==1 and mark [work]>0 and mark[slot1]==1
- **case 2:** mark[user]==1 and mark [buff1]>0 and mark[slot2]==1

while it does not produce any work if:

- **case 3:** mark[system]==1 and (mark[work]>0 or mark[buff]>0)
- **case 4:** mark[stop]==1

If by $p_i(t)$ and r_i we respectively mean the probability that the system is in the state i, and the reward associated with such state, by studying the Petri net we can obtain the expected istantaneous reward rate as:

$$E[Z(t)] = \sum_{i \in S} r_i p_i(t)$$

and the expected value of the accumulated reward $Y(t)$ as:

$$E[Y(t)] = \sum_{i \in S} r_i L_i(t)$$

where

$$L_i(t) = \int_0^t p_i(\tau) d\tau.$$

With regard to the distributions of the firing times to be assigned to timed transitions, let us assume that transitions *U_time, S_time, busy_brod, busy2* are deterministic. We assume that transitions *producer* and *cons1* are respectively distributed uniformly and exponentially. The measures considered can therefore be evaluated by changing the memory policy associated to transitions *producer* and *cons1*, and consequently changing the type of real system corresponding to the model.

In the case of *prd* policy, the temporary interruption of the processing of a job (either because the whole system enters the phase of maintenance, or because, even if the system is in the production phase, it interrupts the pre-processing phase for changing to the processing one or vice versa) causes the interrupted job to be discarded. A new processing is executed when the system is available again. The correspondence with a real system is perhaps hard to find; however, we must notice that *prd* policy is the most commonly used in literature.

Conversely, by adopting *prs* policy, we keep a memory of the work that we were executing. In this case, when transition *producer* is disabled, we keep a memory of the work that has already been executed on the job considered. When the system enters the operative state again, the pre-processing of the job continues from the point we had reached. In this case, the model can represent a system of *manufacturing*, where a machine used for production alternates cycles of production and cycles of maintenance, and production takes place on two sequential phases. We must notice that *prd* and *prs* policies are equivalent for transition *cons1*, since this one is exponential.

With *pri* policy, when transition *producer* is disabled, the work that had already been produced is lost, but we keep a memory of the job that we were processing. When the transition is enabled again we start from zero, but the amount of work to be produced on the job remains the same, because the job has not been changed. Such a behavior can be easily noticed when accessing transactional databases, where each transaction is atomic (i.e. has to be processed with no interruption). If an interruption occurs, the transaction is entirely processed again. If we assume a memory policy like *prs* for transition *cons1*, the model could represent a client/server system where the accesses to database (transition *producer*) take place atomically, and the phase of processing of the query (transition *cons1*) requires a variable time, distributed exponentially.

5.2 Numerical Results

The following assumptions have been done for the solution of the model:

- transition *producer* is distributed uniformly between 0.5 and 1.5;
- transitions *U_time* and *S_time* are deterministic, with a firing time of 1;
- transitions *busy_prod* and *busy2* are deterministic, with a firing time of 0.1;
- transition *cons1* is distributed exponentially, with a firing rate of 0.1;
- transition *end* is immediate and has a priority of 2;
- transitions *idle_prod* and *idle2* are immediate and have a priority of 1;
- the total number of jobs to be processed is 3.

In figure 8 we show the progress of the distribution of completing time according to the memory policies assigned to transitions *producer* and *cons1*. As we can notice, the behavior of the system changes significantly according to the memory policy adopted. *prs* policy assures the highest probability of completion with the same time. Both *prd* and *prs* assure the completion of jobs. In fact, curves reach the value 1, even if for high values of t. Conversely, a different behavior can be observed if we assume a policy like *pri*. In fact, in that case, the resulting distribution is defective, since the unitary value is never reached for $t \to \infty$. This is closely connected with the choice of the parameters associated to transitions *producer* and *U_time*. As we can notice in figure 9, when the firing time of transition *U_time* is lower than 1.5, transition *producer* has a finite probability of not completing its work. In this specific case, such value is 50% (i.e.

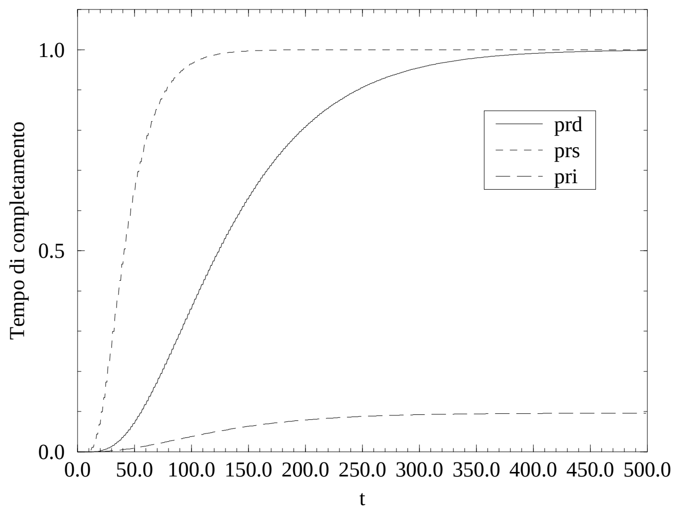

Figure 8: Distribution of completing time

in 50% of cases the sampled value for pre-processing a job results to be higher than the quantum of time assigned by the system to the processing of jobs). Since in the case of *pri* policy the job is proposed with the same work requirement, this causes a situation of impasse, which prevents the work assigned to the system to be completed.

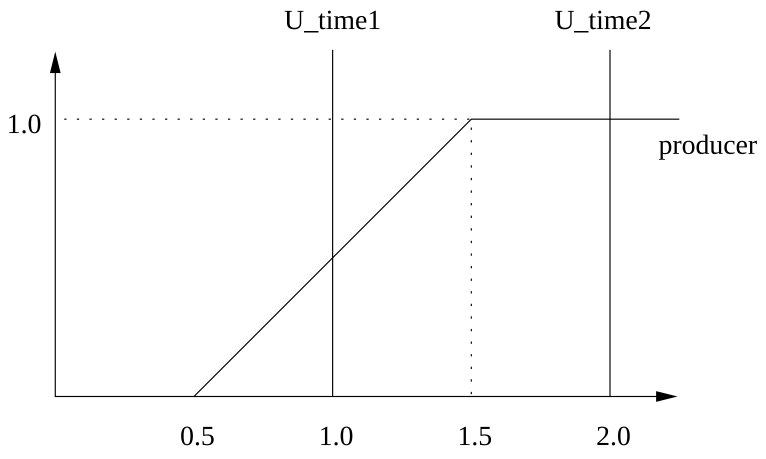

Figure 9: Distributions of transitions *producer* and *U_time*

Figure 10 shows how the behavior of the whole system changes if transition *U_time* is assigned a firing time higher than 1.5 (for example 2.0). In such case, transition *producer* has a finite probability of firing before the system enters the phase of maintenance, and therefore the distribution of completing time with *pri* policy reaches the value 1.

A specific study has been done for analyzing the productivity of the system, i.e. the average gain and the average cost according to time and respective accumulated values. Such measures have been obtained in terms of expected reward rate and accumulated

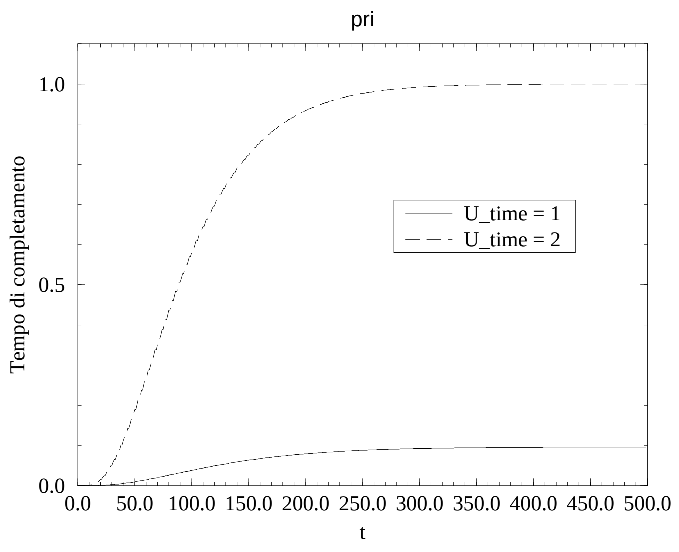

Figure 10: Distribution of completing time

reward, by assigning the following reward indices to the four sets of markings identified in the previous paragraph:

- **case 1:** reward 1.0
- **case 2:** reward 2.0
- **case 3:** reward 1.5
- **case 4:** reward 0.5

Such quantities (expressed in terms of gain/cost per unit of time) denote the gain per unit of time produced when the system is in the case 1 or 2, and the cost per unit of time when the system is in condition 3 or 4.

Figure 11 shows the progress of the expected reward rate associated to the case 1, where (according to the deteministic transition U_time) we assumed a firing time of 2.0 and *pri* policy for transition *producer*. As we can notice, the progress starts from 1 and decreases till 0. This shows that, when time increases, the average gain obtained during the pre-processing phase tends to decrease, as a consequence of the exhaustion of the jobs to be processed. The curve remains at 1 up to $t = 0.5$ (lower extreme of the uniform distribution associated to transition *producer*), and then decreases with variable trend. In particular, we can observe that the curve reaches *zero* for $t = 2.0$ (firing time of *U_time*), remains in such position for an interval of 1.0 (firing time of *S_time*), then goes up again and repeats this periodic trend. As we can see from the zoom in the upper right angle of the figure, a variable trend is also present in the intervals with a positive expected reward, due to the alternation of the system between the two phases of sequential processing (deterministic transitions *busy_prod* and *busy2* that have a firing time of 0.1).

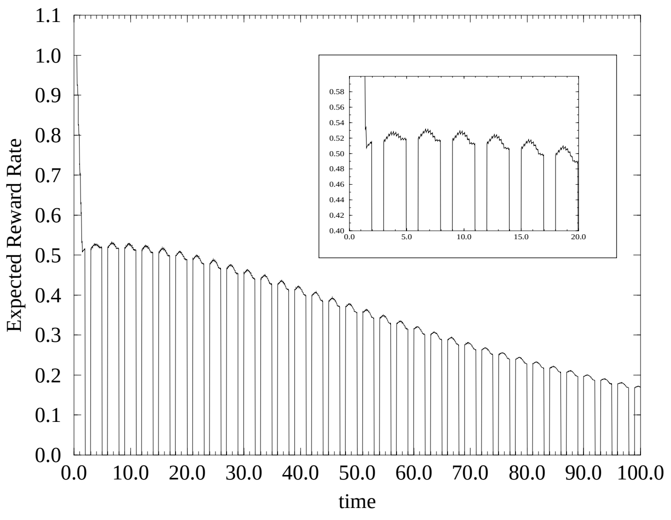

Figure 11: Expected reward rate

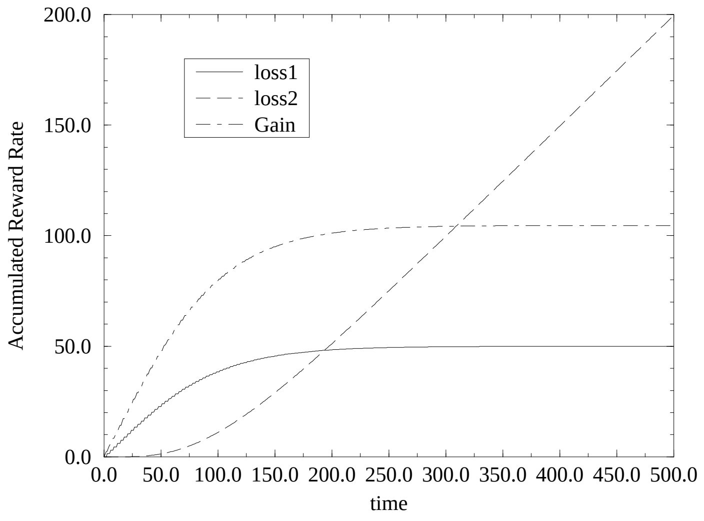

Figure 12: Accumulated reward rate

The progress of the accumulated reward is shown in figure 12. The curves called $loss1$ and $loss2$ respectively refer to the conditions $case3$ and $case4$, while the curve $Gain$ carries the sum of the accumulated reward in conditions of operation $case1$ and $case2$. $Gain$ and $loss1$ have an increasing trend that tends to become stable for $t \to \infty$, since the system tends to reach the absorbing state corresponding to the completion of all the jobs. Conversely, $loss2$ has an increasing trend tending to the infinity for $t \to \infty$. This denotes that the cost due to the inactivity of the system continues to increase for all the time of inactivity of the system. $loss2$ meets $loss1$ and $Gain$ respectively for $t = XX$ and $t = YY$. At $t = XX$ the cost due to the inactivity of the system equals the maintenance one, while at $t = YY$ the cost of inactivity is higher than the gain produced by the system in the processing of the jobs it has been assigned.

According to the results obtained, appropriate strategies for increasing the productivity of the system can be adopted, by operating on its typical parameters such as time, maintenance frequency and the scheduling between the phases of pre-processing and of processing.

6 Conclusion

A new modeling tool, called WebSPN, for specification and automatic solution of not-Markovian *SPN* has been described. The tool implements a time-discretization approach and allows concurrent enabling of generally distributed transitions with *prd, prs* and *pri* preemption policies. Due to the use of the Java programming language, WebSPN is easily accessible from any node connected with the Internet as long as it possesses a Java-enabled Web browser. We described the main idea behind the different memory policies we propose and the way in which the discretization approach works. A quite complex example has been solved with WebSPN to highlight the main features of the proposed approach.

References

[1] M. Ajmone Marsan, G. Balbo, A. Bobbio, G. Chiola, G. Conte, and A. Cumani. The effect of execution policies on the semantics and analysis of stochastic Petri nets. *IEEE Transactions on Software Engineering*, SE-15:832–846, 1989.

[2] A. Bobbio, V.G. Kulkarni, A. Puliafito, M. Telek, and K. Trivedi. Preemptive repeat identical transitions in Markov Regenerative Stochastic Petri Nets. In *6-th International Conference on Petri Nets and Performance Models - PNPM95*, pages 113–122. IEEE Computer Society, 1995.

[3] A. Bobbio and M. Telek. Markov regenerative SPN with non-overlapping activity cycles. In *International Computer Performance and Dependability Symposium - IPDS95*, pages 124–133. IEEE CS Press, 1995.

[4] G. Chiola. *GreatSPN* 1.5 Software architecture. In G. Balbo and G. Serazzi, editors, *Computer Performance Evaluation*, pages 121–136. Elsevier Science Publishers, 1992.

[5] G. Ciardo, J. Muppala, and K.S. Trivedi. SPNP: stochastic Petri net package. In *Proceedings International Workshop on Petri Nets and Performance Models - PNPM89*, pages 142–151. IEEE Computer Society, 1989.

[6] J.A. Couvillon, R. Freire, R. Johnson, W.D. Obal, M.A. Qureshi, M. Rai, W. Sanders, and J.E. Tvedt. Performability modeling with UltrasSAN. *IEEE Software*, 8:69–80, September 1991.

[7] A. Cumani. Esp - A package for the evaluation of stochastic Petri nets with phase-type distributed transition times. In *Proceedings International Workshop Timed Petri Nets*, pages 144–151, Torino (Italy), 1985. IEEE Computer Society Press no. 674.

[8] A. O. Freier, P. Karlton, and P. C. Kocher. SSL Version 3.0. Technical report, Internet draft, Netscape Communications Corp., December 1995.

[9] R. German. New results for the analysis of deterministic and stochastic Petri nets. In *International Computer Performance and Dependability Symposium - IPDS95*, pages 114–123. IEEE CS Press, 1995.

[10] R. German, C. Kelling, A. Zimmermann, and G. Hommel. *TimeNET - A toolkit for evaluating non-markovian stochastic Petri nets.* Report No. 19 - Technische Universität Berlin, 1994.

[11] Li Gong. Java Security: Present and Near Future. *IEEE Micro*, 17(3):14–19, May 1997.

[12] J. Gosling. The Java Language Environment: a White Paper. Technical report, Sun Microsystems, May 1995.

[13] A. Horvath, A. Puliafito, M. Scarpa, and M. Telek. A discrete-time approach to the analysis of non-markovian petri nets. *Technical Report of the University of Catania*, CS-12, 1998.

[14] C. Lindemann. DSPNexpress: a software package for the efficient solution of deterministic and stochastic Petri nets. *Performance Evaluation*, 22:3–21, 1995.

[15] A. Puliafito, M. Scarpa, and K.S. Trivedi. Petri nets with k simultaneously enabled generally distributed timed transitions. *Performance Evaluation*, 32 n.1, February 1998.

[16] A. Puliafito, O. Tomarchio, and L. Vita. Porting Sharpe on the Web: Design and Implementation of a network computing platform using JAVA. In *Proceedings of TOOL'97*, Saint Malo, France, June 1997.

[17] R. A. Sahner, K. S. Trivedi, and A. Puliafito. *Performance and Reliability Analysis of Computer Systems.* Kluwer Academic Publishers, November 1995.

[18] W. Stalling. *Network and Internetwork Security Principles and Practice.* Prentice Hall, 1995.

[19] M. Telek, A. Bobbio, and A. Puliafito. Steady state solution of MRSPN with mixed preemption policies. In *International Computer Performance and Dependability Symposium - IPDS96.* IEEE CS Press, 1996.

[20] E. Yourdon. Java, the Web, and Software Development. *IEEE Computer*, 29(8):25–30, August 1996.