Proceedings of the Federated Conference on
Computer Science and Information Systems pp. 763–772

DOI: 10.15439/2015F69
ACSIS, Vol. 5

Qualitative and Quantitative Evaluation of Stochastic Time Petri Nets

Franco Cicirelli, Christian Nigro, Libero Nigro
Laboratorio di Ingegneria del Software
Dipartimento di Ingegneria Informatica Modellistica Elettronica e Sistemistica
Università della Calabria, Italy
{f.cicirelli@dimes.unical.it, christian.nigro@tiscali.it, l.nigro@unical.it}.

Abstract— **Time Petri Nets (TPN) are a well-known formalism for modelling time-dependent systems with timing constraints. This paper proposes an approach based on a stochastic extension of TPN (sTPN), which enables both qualitative assessment of feasible temporal behaviors through model checking, and quantitative evaluation of a probability measure of a given behavior, by statistical model checking. The experimental work rests on the use of the latest version of the UPPAAL toolbox which supports both exhaustive non deterministic analysis and statistical model checking of system properties. The approach is demonstrated through an example.**

I. INTRODUCTION

The development of safety-critical software systems is challenged by the needs of addressing both functional and temporal correctness issues. Violation of timing constraints can have important consequences in the practical domain (e.g., economy, medicine, cyber physical control systems, etc.). Therefore, it is highly recommended the use of formal tools for modelling and analysis of concurrent and timed software.

From the point of view of analysis, both qualitative verification and quantitative evaluation of system properties are nowadays advocated by engineers and developers. Whereas qualitative verification of a system model tries to identify feasible behaviors, quantitative evaluation aims to associate them a measure of occurrence probability.

Qualitative verification is often based on the exhaustive enumeration of all the possible execution states of a model, organized in the so called model *state graph*, and on checking desired properties through efficient traversal algorithms on the state graph. To avoid an infinite growth of the state graph, each state node is implemented as a couple: a *discrete data part*, and *a dense time part*. The time component typically stores in a compact way (*zone*) all the clock (timer) inequalities which hold in the state. As a consequence, a state is in reality a *state class* which subsumes an infinite number of states which could be reached by only changing the clock values (firing times of transitions).

Despite the use of state classes, depending on the system model, the state graph can suffer of state explosion problems. In addition, the construction and navigation of the state graph can become undecidable when a complex combination of modelling factors (non-deterministic time of actions, sporadicity of process arrival, message passing, stochastic aspects etc.) occurs. In these cases, the study of system properties can only be approximated, e.g., through simulation.

Qualitative non deterministic verification based on model checking [1] has been demonstrated by Time Petri Nets [2]-[5] and Timed Automata [6] based tools, e.g. [7]-[12]. Quantitative evaluation of system behavior is supported by recent extensions to the ORIS tool [8] and by latest versions of UPPAAL [9]-[11] which include a statistical model checker (SMC) [13].

This paper proposes an original approach to qualitative and quantitative evaluation of systems with timing constraints which is based on the Time Petri Net (TPN) formalism which is very often used for modelling real-time and embedded systems, communication protocols etc. To permit both non-deterministic analysis and stochastic analysis of system properties, a stochastic extension of TPN (sTPN) [14] is also considered. The contribution of the paper consists in a mapping of TPN/sTPN onto UPPAAL so as to exploit, on a same model, both the exhaustive model checker and the stochastic model checker.

Whereas the support of sTPN in the ORIS tool is based on the concept of *stochastic state class* and *stochastic state graph*, i.e., a density probability distribution function is attached to each state class which characterizes the possible stochastic evolutions from it, i.e., estimating the probability of the outgoing state transitions, the use of sTPN in UPPAAL rests on batches of simulation runs and statistics inference of desired results from these runs. As a consequence, ORIS can provide a greater resolution on the probability measures. However, this paper argues that the proposed approach based on UPPAAL has the following strengths: (1) it is based on a popular and efficient toolbox, (2) it does not incur in an infinite stochastic state graph nor suffer of stochastic state explosion problems as discussed in [14] (3) it in any case can provide quantitative measures of probability which are valuable from the engineering point of view.

The paper is structured as follows. Section II provides basic definitions of TPN and sTPN formalisms. Section III

This work was not supported by any organization

describes a modelling example. Section IV gives an overview to non-deterministic analysis and stochastic analysis enabled by UPPAAL. Section V discusses the developed structural translation from TPN/sTPN to UPPAAL. Section VI illustrates the application of the proposed approach to a thorough property assessment of the model described in Section III, by detailing qualitative and quantitative analysis. Section VII concludes the paper by indicating directions of on-going and future work.

II. TIME PETRI NETS DEFINITIONS

A basic TPN is a tuple $(P, T, B, F, M_0, I_{nh}, EFT^s, LFT^s)$ where:

- P is a finite nonempty set of places;
- T is a finite nonempty set of transitions;
- $P \cap T = \emptyset$;
- B is the backward incidence function, $B: P \times T \to \mathbb{N}$, where $\mathbb{N}$ denotes the set of natural numbers;
- F is the forward incidence function, $F: P \times T \to \mathbb{N}$;
- I_{nh} is the set of inhibitor arcs, $I_{nh} \subset P \times T$ where $(p,t) \in I_{nh} \Rightarrow B(p,t) = 0$;
- M_0 is the initial marking function, $M_0: P \to \mathbb{N}$, which associates with each place a number of tokens;
- $EFT^s: T \to R^+$ is a function which associates each transition with a (finite) earliest static firing time. R^+ denotes the set of non-negative real numbers;
- $LFT^s: T \to R^+ \cup \{\infty\}$ is a function which associates each transition with a (possibly infinite) latest static firing time. In any case it must be $LFT^s \geq EFT^s$.

Differently from [13], in this work TPNs admit both inhibitor arcs and non-unitary arc weights.

The state of a TPN is a pair $s =< m, \tau >$ where $m: P \to N$ is the net marking, and $\tau: T \to R^+$ associates each transition with a (dynamic) *time-to-fire* (clock or timer). The state evolves according to the *firability* and *firing* clauses.

A transition t is *enabled*, as in classic Petri nets, if each of its input places contains sufficient tokens, i.e., iff

$$\forall\, p \in P, (p,t) \in I_{nh} \Rightarrow M(p) = 0 \ \wedge \\ B(p,t) > 0 \Rightarrow M(p) \geq B(p,t)$$

Transition t is *firable* if it is enabled and its time-to-fire $\tau(t)$ is not higher than that of any other enabled transition.

When t fires, the state $s =< m, \tau >$ is replaced by a new states $s' =< m', \tau' >$ where marking m' is derived from m by the withdrawal of tokens from the input places and the deposit of tokens in the output places. More precisely, the firing process consists of the two (atomic) phases:

$$m_{int}(p) = m(p) - B(p,t) \text{ (withdraw phase)}$$
$$m'(p) = m_{int}(p) + F(p,t) \text{ (deposit phase)}$$

Transitions which are enabled in the intermediate marking m_{int} (and then also in m) *and* in the final marking m' are said *persistent* to the t firing. Transitions which are enabled in m' but not in m_{int} are said *newly enabled*.

A transition which is multiple enabled in a state s is supposed to consume its enablings one at a time (*single server* semantics). Therefore, after its own firing, would t be still enabled, it is regarded as a newly enabled one.

For any transition t_p which is persistent to the firing of t, its time-to-fire is reduced, in the new state s', as follows:

$$\tau'(t_p) = \tau(t_p) - \tau(t)$$

For any newly enabled transition t_{ne} its time-to-fire is constrained to occur not deterministically in its static time interval:

$$EFT^s(t_{ne}) \leq \tau'(t_{ne}) \leq LFT^s(t_{ne})$$

A. Stochastic extensions

An sTPN [14] specializes a basic TPN as follows. The set of transitions is partitioned into two subsets: $T = T_1 \cup T_2$, where $T_1 \subseteq T$ is the subset of *timed transitions*, $T_2 \subset T$ is the set of *immediate transitions*. Besides its static time interval, a timed transition is attached a probability density distribution function $PDF: T_1 \to R^+$, which is constrained in the static time interval $[EFT^s, LFT^s]$ of the transition. It is also said the static time interval is the *support* of the pdf. An immediate transition is attached a real positive weight $\pi: T_2 \to R^+$.

The semantics interpretation of an sTPN is as follows. Immediate transitions (as in Generalized Stochastic Petri Nets –GSPN- [15]) always fire *before* any timed transition, and consumes no time. The set of simultaneously enabled immediate transitions in the current state constitutes a *random switch*, i.e., each immediate transition t_i is firable with probability

$$Prob(t_i) = \frac{\pi(t_i)}{\sum_{t_j \in T_2 \text{ and } t_j \text{ is enabled}} \pi(t_j)}$$

The time-to-fire $\tau(t)$ of a timed transition t is stochastically defined, at its enabling instant, by sampling the $PDF(t)$ with the constraint:

$$EFT^s(t) \leq PDF(t) \leq LFT^s(t)$$

As an example, the PDF of a timed transition can be (default) a uniform distribution function which picks up a value in the static time interval of the transition, or a negative exponential distribution function. However, in the general case, a timed transition can follow a generally distributed function constrained in the support time interval.

Firing of a timed transition follows the same rules as in basic TPNs.

III. A Modelling Example

Fig. 1 depicts a TPN model [14] made up of two (almost identical) production cells, identified by suffixes 1 and 2, which operate sequentially and cyclically.

Each cell admits two parallel activities named JobA (t1, t7) and JobB (t2, t8). JobA requires the use of a resource res (places p2 and p9) which may fail during the usage. In the case of failure, JobA is not completed, and a recovery action recA (t5 and t11) is instead executed which replaces the normal behavior provided by JobA. A failed resource is repaired by a repair transition (see t3 and t9). The execution of a production cell is started by the start transition (t0, t6). A cell logically terminates its current production phase when a token is generated in the couple of termination places (p5, p6) or (p12, p13).

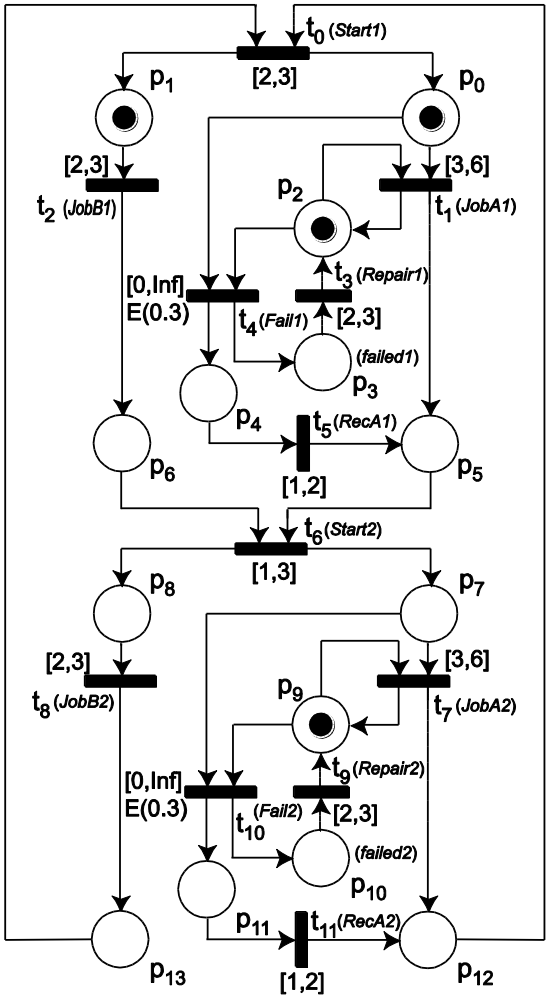

Fig. 1 A TPN/sTPN model with two production cells

All the transitions in Fig. 1 are supposed, in the stochastic interpretation, to be served by a uniform distribution in the associated static time interval, with the exception of the failure transitions (t4, t10) which have a support interval of $[0,\infty]$ and an exponential distribution function whose rate is $\lambda = 0.3$, as witnessed by the notation E(0.3) attached to the transitions.

The use of $[0,\infty]$ as the time interval of a failure transition is noteworthy. In both non-deterministic and stochastic interpretations the failure cannot occur later than 6 time units measured from the time instant in which the failure transition gets enabled. This happens because of the [3,6] time interval of the JobA transition, which forbids the failure to occur later than 6. In the stochastic interpretation, however, something subtle occurs. Due to the exponential distribution E(0.3), the failure is expected to happen with a very low probability, as the sample chosen from the E(0.3) pdf can be much greater than 6 and thus later with respect to the completion time of the JobA activity. All of this reflects the different concerns of non-deterministic analysis and stochastic analysis (see later in this paper).

IV. UPPAAL Concepts

The popular and efficient UPPAAL toolbox [9]-[10] allows modelling and verification of time-dependent systems. An UPPAAL model consists of a network of timed automata (TA) [6]. TA are designed as *template processes*, which can have parameters, can be instantiated, and consist of *atomic actions*.

TA are extended with local or global integer (and boolean) variables and arrays of integers, clocks and channels. In latest versions of the toolbox, C-like functions and structures are also permitted. Time is dense and can be controlled by means of *clock* variables. Clocks can only be reset and compared against nonnegative integer constants.

All the clocks of a model increase automatically with the same rate of advancement of the hidden system time. TA synchronize to one another by CSP-like channels (*rendezvous*) which carry no data values.

Asynchronous communication is provided by broadcast channels where a single sender can synchronize with a (possibly empty) group of receivers. The sender of a broadcast signal in no case is blocked. Locations (states) of an automaton are linked by *edges* (transitions).

Every edge can be annotated by a *command* with three (optional) elements: (i) a *guard*, (ii) a *synchronization* (? for input and ! for output) on a channel, and (iii) an *update* consisting of a set of clock resets and a list of variable assignments. Channel synchronization implies the commands of the sender and of the receiver(s) are jointly executed. However, the update of an output command is executed *before* that of a matching input command.

An atomic action consumes no time and refers either to the execution of an internal command of one automaton or to a joint execution of the multiple commands during a synchronization among two or more TA.

A clock *invariant* can be attached to a location as a *progress* condition. The timed automaton can remain into the location as long as its invariant holds. UPPAAL supports also *committed* and *urgent* locations which must be exited immediately (i.e., without passage of time), and *urgent channels* whose synchronizations must be fired without passage of time. Committed locations, among them, can be

interleaved. Similarly, urgent locations, among them, can be interleaved. However, committed locations have priority with respect to urgent locations.

The symbolic model checker of UPPAAL handles the parallel composition of the TA of a model. Parallel composition means generating all the possible action interleavings of the component concurrent processes.

UPPAAL consists of a graphical editor, a simulator and a verifier (the symbolic model checker verifyta). For exhaustive property assessment, the verifier tries to build the reachability graph of the model, where each state node holds a *data part* (variable values and location of each automaton) and a *firing domain* (time zone or clock inequalities system). The time zone implies each state graph node actually represents a *class* of equivalent states which fulfill the clock inequalities. The simulator executes a system model and visually documents the reached execution state by following a particular path in the model state graph. The simulator is useful for model debugging and to examine a diagnostic trace (counterexample) created by the verifier, e.g., when a property is not satisfied. Safety, absence of deadlocks, and bounded liveness (e.g., an end-to-end time constraint) properties can be verified by reachability analysis upon the state graph, using a subset of TCTL temporal logic formulas [10] as shown in the following:

- $E <> \varphi$ (Possibly ψ, i.e., a state exists where ψ holds)
- $A[]\ \psi$ (Invariantly ψ, equivalent to: $not\ E <> not\ \psi$)
- $E[]\ \psi$ (Potentially Always ψ, i.e. a state path exists over which ψ always holds)
- $A <> \psi$ (Always eventually ψ, equivalent to: $not\ E[]\ not\ \psi$)
- $\psi -\!-> \xi$ (ψ always *leads-to* ξ, equivalent to: $A[]\ (\psi\ imply\ A <> \xi)$)

where ψ and ξ are state properties (formulas), e.g., clock constraints or boolean expressions over predicates on locations.

Although min/max determinations, e.g., of a clock, can be achieved by using the above described logic queries, a shorthand notation is available as in the following:

sup{ state-predicate } : list-of-expressions
inf{ state-predicate } : list-of-expressions

These queries evaluate the superior/inferior value of the list of expressions, only in the states of the state-graph which satisfy the state predicate specified within { and }.

A. UPPAAL Statistical Model Checker

The problem with symbolic model checking is that it could not be practically applied to realistic complex systems which generate an enormous (possible infinite) state graph, or it becomes undecidable for systems which combine in a complex way continuous time with stochastic behavior.

Property checking in these cases can only be approximated or estimated. In recent years the UPPAAL toolbox was extended to support stochastic model checking (SMC) [9]. UPPAAL SMC [11] avoids the construction of the state graph and checks properties by performing a certain number of simulation runs, e.g., in parallel on a modern multi-core machine. After that some statistics techniques are used to infer results from the simulation runs.

SMC refines and extends basic UPPAAL. Only broadcast synchronizations are allowed among stochastic TA (STA). In addition, either an invariant or the rate of an exponential distribution can be attached to a location. Stopwatches, i.e. clocks whose automatic advancement can be temporarily stopped (their first derivative is put equal to zero as an invariant of a location) can be exploited. A stopwatch resumes its advancement as soon as the automaton exits the location in which it was stopped.

UPPAAL SMC also provides floating point (double) variables which, e.g., can be assigned the value of a clock. Virtually, the symbolic model checker can be applied to a stochastic model too, in which case all doubles, exponential distribution rates etc. are simply ignored. However, on a stochastic TA model can be issued the following specific query types. Bold symbols are meta-symbols used to describe the SMC query language.

1. simulate N [**(**clock|#|void**)** <=bound] { Expression1, …, Expressionk }
2. Pr[**(**clock|#|void**)** <=bound] (**(**<>|[]**)** Expression)
3. Pr[**(**clock|#|void**)** <=bound] (**(**<>|[]**)** Expression) **(**<=|>=**)** PROB
4. Pr[**(**clock|#|void**)** <=bound] (**(**<>|[]**)** Expression) **(**<=|>=**)** Pr[**(**clock|#|void**)** <=bound] (**(**<>|[]**)** Expression)
5. E[**(**clock|#|void**)** <=bound; N] (**(**min:|max:**)** Expression)

Expressions are state predicates without side-effects. They can specify an automaton to be in a certain location, or some constraints on data variables or clocks etc. All the queries are evaluated according to a bound which can be related to (implicit) global time or to a clock or to a number of simulation steps (#).

Query 1 makes N simulation experiments and collects information about the listed expressions. Query 2 evaluates the probability the given expression holds within the assigned bound (<>) or always holds within the bound ([]) with a confidence interval (the default 95% confidence degree can be customized by the user). Query 3 checks if the estimated probability is less/greater than a given probability value. Query 4 compares two probabilities. Query 5 estimates the minimum or the maximum value of an expression.

Responding to queries implies a certain number of simulation runs are carried out, either explicitly requested (see the parameter N in the queries 1 and 5) or implicitly defined by the query. Quantitative estimation of a query of type 2 rests on Monte Carlo-like simulations. Qualitative queries of the type 3 and 4 use sequential hypothesis testing. An important feature provided by UPPAAL SMC is

visualization of simulation results. Following a satisfied query, the modeler can right click on the executed query and choose an available diagram (histogram, probability distribution etc.) to be plotted. At the time of this writing, UPPAAL SMC is supported by the development version 4.1.19.

V. MAPPING TPN/STPN ONTO UPPAAL

A TPN/sTPN model is translated into UPPAAL by associating each transition with a suitable template process and by introducing some global data and helper functions. A similar approach was adopted by authors in [16] which provides a formal correctness approach exploitable also in current work. For brevity, though, in the following only an informal semantics will be given.

A. TPN issues

The structural translation adapts itself to the needs of both non deterministic analysis (TPN model checking) and the stochastic analysis (sTPN SMC). During the exhaustive verification of a TPN model, all the transitions are homogeneously modelled as timed transitions (see the tTransition automaton in Fig. 2). Immediate transitions, in particular, are expressed as timed transitions with a [0,0] static time interval. Random switches are thus replaced by non-deterministic selection (*race*) among transitions firable at the same time.

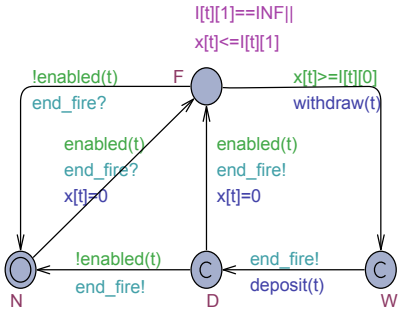

Fig. 2 The tTransition template automaton

The tTransition automaton in Fig. 2, with the sole parameter const tid t, models the generic timed transition of a TPN model. The t transition id is used to select the clock x[t] which captures the time-to-fire of the transition. The global functions enabled(t), withdraw(t) and deposit(t) assist the automaton evolution.

A timed transition starts into the N (Not enabled) location. As soon as it finds itself enabled, it resets its clock x[t] and moves to the F (Firable) location. The invariant attached to the F location states that the transition can remain in F until the latest static firing time which can be infinite. On the other hand, transition t can fire as soon as it happens its clock x[t] reaches the earliest static firing time. Bounds of the static time interval of the transition t, constrained to be positive integers (an upper bound infinite is denoted by the constant INF), are held into the global array $I: T \times 2 \rightarrow int$. $I[t][0]$ is the $EFT^s(t)$, $I[t][1]$ is the $LFT^s(t)$.

The firing process is accomplished with the help of the end_fire broadcast channel. Two broadcast synchronizations are actually used, separately triggering the withdraw-phase and the deposit-phase. Each end_fire synchronization is capable of influencing all the other transitions which are forced to re-examine their enabling status, and thus commanding a return from F to N when a firable transition detects it is no longer enabled, or moving from N to F when a not enabled transition finds itself enabled. It is worth noting that the two location W (Withdraw) and D (Deposit) are committed locations, thus ensuring the firing process is instantaneous and atomic. In the D location, a just fired transition which is still enabled, comes again into the F location and resets its clock x[t] (single server semantics). If the transition is not enabled, it reaches the default N location.

To bootstrap a TPN model, a Starter automaton (see Fig. 3) is exploited which launches an initial (fictitious) end_fire synchronization to force transitions which are enabled in the initial marking of the model, to reach the F location.

Fig. 3 The Starter automaton

Behind the design of the tTransition automaton, the following global declarations are introduced. Model topology is captured by the constants P (number of places), PRE (maximum number of input places of a transition), POST (maximum number of output places of a transition), T (number of transitions), B and F (input/output constant incidence matrices), M (marking vector), I (time intervals). Each element of the matrices B and F, purposely implemented as TxPRE and TxPOST respectively, holds the index of a place, and the weight of an input and, respectively, an output arc. Transitions are numbered from 0 to T-1. The subtypes tid and pid describe respectively the sub range types of possible transition or place ids.

B. sTPN issues

The support of sTPN shares the same basic global declarations described in the previous sub-section plus some specific declarations required for statistical model checking.

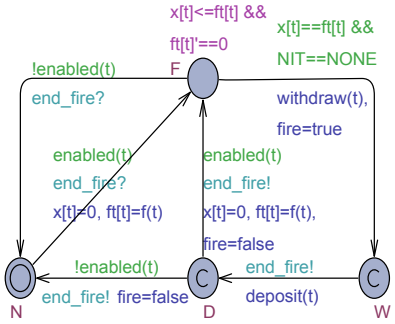

Fig. 4 The sTransition automaton

An sTPN model rests on two new timed automata shown in Fig. 4 and Fig. 5, respectively associated to a stochastic timed transition (sTransition) and to an immediate transition (iTransition).

As a matter of convention, first are numbered the stochastic transitions, from 0 to the TT-1 (TT is a model constant), then are numbered the immediate transitions (from TT to T–1). Of course, T=TT+IT. Correspondingly are defined the subtypes ttid and itid describing respectively the subranges of the stochastic and immediate transitions. An sTransition has the only the t (id) parameter of type ttid. Similarly, iTransition has the sole t parameter of type itid. All of this ensures the transition automata of a TPN/sTPN model can be implicitly instantiated at system configuration time.

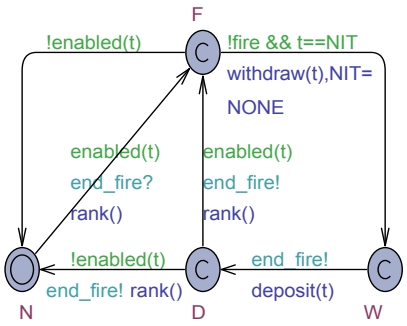

Fig. 5 The iTransition automaton

The sTransition uses two clocks: x[t] and ft[t]. The first one serves the same purposes discussed for the tTransition automaton (see Fig. 2). The second one actually stores the sampled fire time achieved (through integration of) the density distribution function (denoted by f(t)) associated to the transition. As one can see from Fig. 4, an sTransition actually fires (exiting the F location) when its time-to-fire clock x[t] equals the fire time held in the ft[t] clock. As part of the invariant attached to F, the first derivative of the clock ft[t] is put to 0 to avoid its advancement. Other details in Fig. 4 are as for the tTransition automaton.

The iTransition automaton in Fig. 5 does not use any clock. When it finds itself enabled, it moves to the committed location F. The rank() function implements the random switch and assigns to the global NIT variable (Next Immediate Transition) the id of the immediate transition probabilistically selected through the transition weights. The immediate transition that detects it is the selected NIT, instantly fires and resets NIT to NONE in preparation of a next execution of the random switch. As soon as a firable immediate transition discovers it is no longer enabled (for a conflict) it comes back to N.

Due to the committed location F in Fig. 5, and the (sub) guard NIT==NONE in Fig. 4, it is guaranteed that immediate transitions get fired before timed transitions. Moreover, the global bool fire variable in Fig. 4 and Fig. 5 ensures the fire of an immediate transition cannot actually start if the firing in progress of a timed transition is not completed.

VI. Analysis of a TPN/sTPN model

The structural translation from TPN/sTPN to UPPAAL described in the previous section, was applied in a case to the model presented in Section III.

A. Non deterministic analysis

Here the concern was to evaluate qualitative properties of the model. Towards this timed transitions and immediate transitions are both modelled with the tTransition automaton in Fig. 2, and the exhaustive model checker of UPPAAL was used along with the construction of the (hopefully finite) state graph. The following clarifies the system configuration of the TPN model:

```
system Starter, tTransition;
```

where implicit instantiation is exploited. Due to the design of the template processes, only one instance of Starter is created, whereas T instances of tTransition (as demanded by the tid sub range type) are generated.

As a preliminary assessment of system behavior, the model was checked for the absence of deadlocks. The query was:

```
A[] !deadlock                                    satisfied
```

where deadlock is a reserved keyword of UPPAAL. As a part of this first query, it was also implicitly assessed that the model is *safe*, i.e., 1-bounded. This was simply achieved by having declared the marking vector thus:

```
int[0,1] M[P]={ 1,1,1,0,0,0,0,0,0,1,0,0,0,0 };
```

and never observing an assignment out of bounds to M.

After that the model was checked for the existence of *regenerative states* [14]. A regenerative state is one where previous history of the system can be disregarded because it does not influence the future behavior.

First it was checked the property stated in [14], page 715, that repair2 cannot be persistent at each firing of start1, thus (also activating the generation of a diagnostic trace):

```
A[] (tTransition(0).W||tTransition(0).D) imply !tTransition(9).F
```

which asks if invariantly in both the intermediate marking (determined by the withdraw phase) and the final marking (determined by the deposit phase) of the firing of t0 (start1), the transition t9 (repair2) cannot be in its firable location F.

The property was found to be false as witnessed by the counterexample proposed by UPPAAL that shows effectively repair2 can be persistent to start1.

In reality, the two production cells could admit regenerative states when a *complete* processing of the second cell certainly finds the first cell in its home marking, and vice versa: when the first cell finishes its processing, the second cell *is* in its home marking. The following queries

were issued to the model checker (recall places are uniquely identified by their subscripts, i.e., the numbers from 0 to P−1):

```
A[] M[5]==1 && M[6]==1 imply (M[9]==1 &&
   forall(i:int[7,13]) i!=9 imply M[i]==0)                satisfied

A[] M[12]==1 && M[13]==1 imply (M[2]==1 &&
   forall(i:int[0,6]) i!=2 imply M[i]==0)                 satisfied
```

which verify if, invariantly, each time one cell finishes its activities, i.e., a token is generated in the terminal places p5 and p6 or p12 and p13 in Fig. 1, the marking of the partner cell is surely its initial marking, meaning that no previous behavior is still in progress in the partner cell. As it was confirmed previously, e.g., at the processing end of the second cell the only possibly still pending sub activity in the second cell is the repairing of the failed res2 resource. However, satisfaction of latter queries guarantee that at each termination of the first cell, the second one always founds itself in its home marking, thus no previous behavior exists in the second cell which could interfere with new behavior.

The existence of regenerative states was a key for assessing specific behaviors of the model. For example, it was first checked about the possibility for the two resources to be failed simultaneously:

```
E<> M[3]==1 && M[10]==1                                   satisfied
```

Having found a double failed state exists, the next step was to identify *all* the possible sequences of transition firings which bring the system into the double failed state. Towards this an array s of transition ids (type tid) was introduced, together with a top variable, to memorize in the occurrence order a transition firing sequence.

The problem, obviously, was the proper dimensioning of the array s. Here it was fundamental the previous study on the regenerative states. In the light of the previous results, it was possible to create the array s with T elements, one per transition. In fact, when e.g. the second cell finishes its processing, there is no need to store again transition firings occurring in the first cell. In particular, it were also detected the instants when the array s can be safely reset in order to help model checking by forcing in s a default value. For instance, when one studies the sequence *first-cell-to-second-cell*, at the time of firing of the start1 (i.e., t0) the array s can be reset. Similarly, when the behavior *second-cell-to-first-cell* is observed, at the subsequent firing of start2 (i.e., t6) the array s can be reset.

Together with the array s, a fired(t) boolean function was introduced to check specifically for the firing of a certain transition. As an example, the existence of the double failed state was also assessed by the query:

```
E<> M[3]==1 && M[10]==1 && !fired(0)          satisfied
```

launched with the initial marking shown in Fig. 1. Now, a state with the two resources failed was checked without an intervening firing of start1, i.e., of the t0 transition, which would begin a new system execution.

Moreover, before launching the previous query, it was also asked to the model checker to generate a diagnostic trace. This way, with the property verified, it was inspected in the simulator both the sequence of events which brings the model in the double failed state, and the values of top and the first top values of s to identify the firing sequence. Therefore, as a further benefit of the previous existential query, it emerged that the sequence: t2-t4-t5-t6-t10 is capable of installing the double failed state.

The analysis was also enriched by the introduction of a global *decoration clock* y which is reset initially and at each firing of start1 (t0), and its value checked as soon as the double failed state is reached. Such a clock permits to measure the time needed, in the best and worst case, to reach the goal state. The conditional reset operation is simply added to the deposit(tid) function. An additional global decoration clock z was introduced to measure the *sojourn time* in the double failed state. The clock z is reset when the goal state is reached, and reset again as soon as the goal state is abandoned. Finding the minimal/maximal values of z provided the sojourn time. As a side benefit, watching the data and constraints (clock values) in the simulator, at the conclusion of the previous query, suggested the interval [4,9] for the possible duration of the sequence t2-t4-t5-t6-t10.

The following query was issued to find, if there are any, a new sequence:

```
E<> M[3]==1 && M[10]==1 && !fired(0) &&
!(s[0]==2 && s[1]==4 && s[2]==5 && s[3]==6 &&s[4]==10)
                                                          satisfied
```

It emerged the new sequence: t4-t2-t5-t6-t10 with a duration of [4,6] time units. The query:

```
E<> M[failed1]==1 && M[failed2]==1 && !fired(t0) &&
!(s[0]==2 && s[1]==4 && s[2]==5 && s[3]==6 &&s[4]==10) &&
!(s[0]==4 && s[1]==2 && s[2]==5 && s[3]==6 && s[4]==10)
                                                          satisfied
```

allowed to discover the third sequence t4-t5-t2-t6-t10 with duration [3,5]. Continuing with updating the query, it emerged that no more sequences exist. The sojourn time for the three firing sequences was found to be in interval [0,1].

By changing the initial marking in Fig. 1 so as to start the execution from the second cell towards the first, and repeating the work of firing sequence detection, it emerged that only other three sequences exist for this new scenario (although in [14] the existence of 7 firing paths in this second scenario was cited but not detailed). Results of the two scenarios are summarized in the Table 1 and Table 2.

As one can see from Fig. 1, a slight temporal difference exists between the first and the second cell, and is concerned with the lower bound of the time intervals of start1 and start2 which are respectively [2,3] and [1,3]. In the case both

cells have the same [1,3] start interval, as expected, the paths in the two scenarios will have an identical temporal behavior. In [14] it is shown that the use of identical time intervals for start1 and start2 implies an infinite stochastic state graph which prevents the analysis. Such problems do not arise in the use of UPPAAL.

TABLE 1 FIRING SEQUENCES TO DOUBLE FAILED STATE FROM FIRST-TO-SECOND CELL

	Firing sequence	Duration	Sojourn time
ρ0	t2-t4-t5-t6-t10	[4,9]	[0,1]
ρ1	t4-t2-t5-t6-t10	[3,6]	
ρ2	t4-t5-t2-t6-t10	[3,5]	

TABLE 2 FIRING SEQUENCES TO DOUBLE FAILED STATE FROM SECOND-TO-FIRST CELL

	Firing sequence	Duration	Sojourn time
ρ3	t8-t10-t11-t0-t4	[5,9]	[0,0]
ρ4	t10-t8-t11-t0-t4	[4,6]	
ρ5	t10-t11-t8-t0-t4	[4,5]	

A further investigation was finally devoted to finding the *end-to-end* delay of a whole operation of the two production cells, i.e., measuring the min/max time required to generate the tokens in the places p12 and p13. The clock y was specialized to this new purpose: it is now reset initially and at each firing of start1 (t0) and checked when p12 and p13 have a token. The two queries:

```
inf{ M[12]==1 && M[13]==1 } : y
sup{ M[12]==1 && M[13]==1 } : y
```

furnished the time window [5,22] for the end-to-end delay.

The model checking work confirmed results achieved in [14] using the ORIS tool, but also suggested some new details.

Qualitative analysis of TPN models is rather efficient. Its scalability ultimately depends on the model topology (number of places and number of timed transitions and associated clocks) and the degree of the necessary model boundedness. As discussed in [16], what is really critical is the number of *active clocks* (i.e., the number of simultaneously fireable timed transitions) mirroring the parallelism degree of the model. An active clock is one which is growing and whose value is checked in a subsequent invariant or edge guard.

A. Stochastic analysis

Whereas the non-deterministic analysis based on model checking says something can happen in the system, i.e., it predicates about qualitative concerns (properties) of the system, the stochastic analysis aims to determine a quantitative measure of the probability with which a given property can actually occur in the system.

As a concrete example, UPPAAL SMC was applied to the model in Fig. 1. To this end the system model was configured by using the sTransition automaton for the timed transitions, and the iTransition automaton for the immediate transitions (which are not used in Fig. 1).

As a preliminary step, it was checked the time limit to use for simulations. As an example, transitions start1 (t0) and start2 (t6) were monitored by cumulating their service time samples and by counting their number of firings. Then the following query was issued:

```
simulate 1 [<=100000] { t0mst, t6mst }
```

where t0mst and t6mst are decoration variables (of type double) added to the model for SMC, reflecting the monitored service time means of the two selected uniform distributions (expected values 2.5 and 2). Fig. 6 confirms that after about $3*10^4$ the mean values are reached indicating an end of transient behavior.

The following query was used to check the probability of occurrence of a single failure in the resources (time is a decoration clock mirroring system time advancement):

```
Pr[<=100000] (<>time>=30000 && (M[3]==1 || M[10]==1) )
```

UPPAAL SMC determined for the event, using 36 simulation runs, a confidence interval (CI) of [0.902606,1] with a confidence degree of 95%. Moreover, it proposed a time span like that depicted in Fig. 7 to highlight a probability distribution for the event:

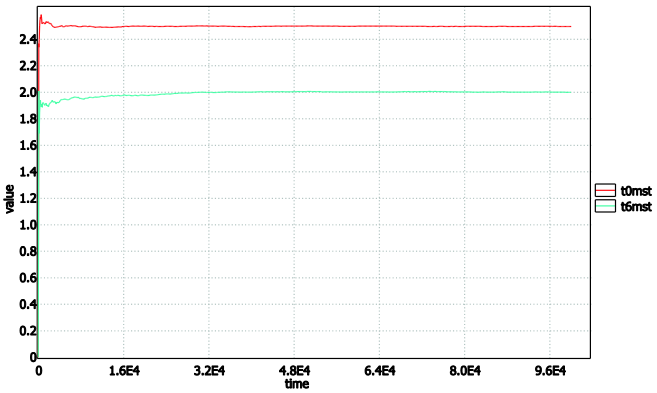

Fig. 6 Monitored values of t0 and t4 mean service time

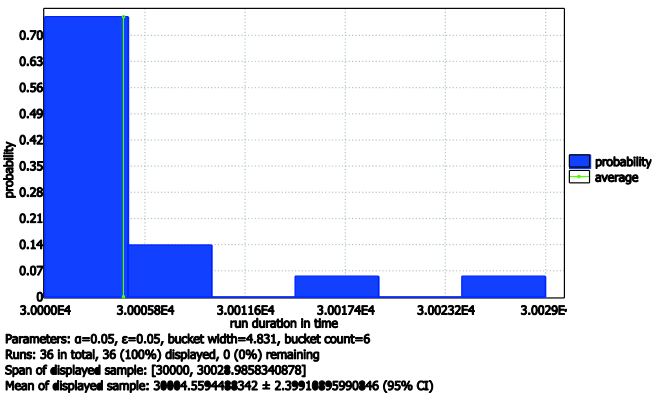

Fig. 7 A probability distribution for the single failure

In a similar way it was checked the probability of occurrence of a simultaneous double failure of resources:

```
Pr[<=100000] (<>time>=30000 && (M[3]==1 && M[10]==1))
```

The event has still a CI of [0.902606,1] 95%, with a suggested probability distribution depicted in Fig. 8.

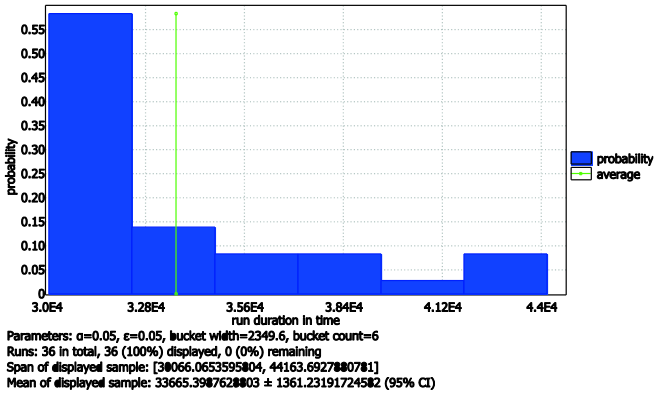

Fig. 8 A probability distribution for the double failure

All the firing sequences in Table 1 and Table 2 have a very low probability of occurrence. Each event was checked as exemplified by the following query:

```
Pr[<=100000] (<>time>=30000 && s[0]==2 && s[1]==4 && s[2]==5 && s[3]==6 && s[4]==10)
```

For all the sequence paths a CI [0,0.0973938] 95% was indicated by UPPAAL SMC.

Some experiments were specifically devoted to quantify the probability of achieving a given end-to-end delay (EED), i.e., the elapsed time from a firing of start1 (t0) to the generation of a token in both places p12 and p13 of Fig. 1. The next query was used to assess the probability of having an EED<=10:

```
Pr[<=100000] (<>time>=30000 && (M[12]==1 && M[13]==1) && y<=10)
```

Clock y is reset at each firing of t0 and measured at the end of the execution of the second cell. UPPAAL SMC proposes a CI of [0.902606,1] 95% and a cumulative probability distribution as portrayed in Fig. 9.

The probability of having an EED<=10 was checked against 80% as follows:

```
Pr[<=100000] (<>time>=30000 && (M[12]==1 && M[13]==1) && y<=10) >= 0.80
```

UPPAAL responds that such a probability is >=0.81 with 95% of confidence.

Monitored values during simulation of the end-to-end delay, collected in Fig. 10, were achieved by the query:

```
simulate 1 [<=100000] { EED }
```

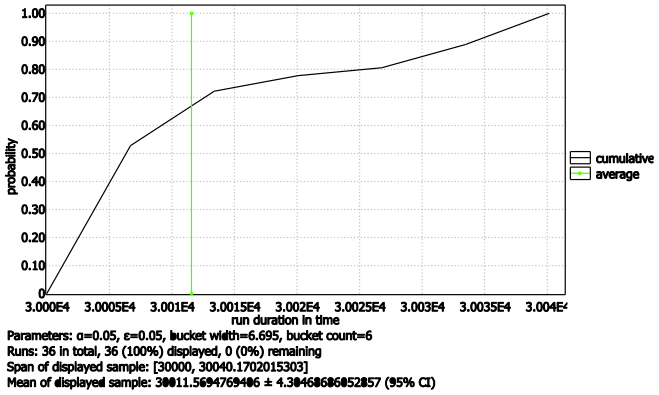

Fig. 9 A cumulative probability distribution for an EED<=10

EED is a decoration variable which receives the value of the clock y at each end of the second cell execution. From Fig. 10 it emerged an average value for the EED of about 10 tu.

Further property estimation was based on MITL formulas [11] which can provide more tight answers. The following query checks the probability that starting from the marking M[0]==1 && M[2]==1 it can follow within 3 time units a failure of res1:

```
Pr( M[0]==1 && M[2]==1 U[0,3] M[3]==1 )
```

Using 738 runs, [0,0.05] was returned as a 95% CI, indicating that the event happens with a very low probability.

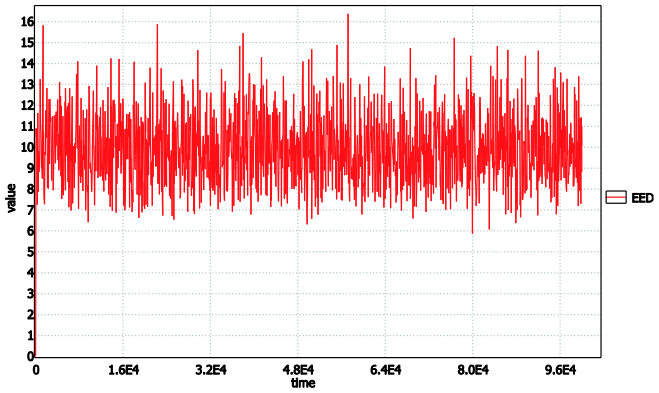

Fig. 10 Monitored EED value

The probability that within [0,1000] tu the double failure state can occur was re-checked with the query:

```
Pr( <>[0,1000] M[3]==1 && M[10]==1 )
```

which returned a [0.142412,0.242412] 95% CI. The query:

```
Pr( <>[0,10] M[12]==1 && M[13]==1 )
```

was used to evaluate the probability that within 10 tu the double cell terminates its execution. The [0.605827,0.705827] 95% CI was observed.

To check specifically the termination event within [10,22] tu it was issued the query:

```
Pr( <>[10,22] M[12]==1 && M[13]==1 )
```

which returned a [0.849729,0.949729] 95% CI to witness a very high probability of occurrence.

The occurrence probability of the firing sequences leading to double failure, was re-checked by the query:

```
Pr( <>[0,1000] top==5 && s[0]==2 && s[1]==4 && s[2]==5 &&
s[3]==6 && s[4]==10 )
```

for which (using 738 runs) a [0,0.054065] 95% CI was proposed. Similarly, for the second and third path of Table 1, it was estimated respectively a CI of [0,0.05] and [0,0.05271] 95%.

Since stochastic analysis depends on batches of simulation runs, that is UPPAAL SMC renounces to build the model state graph, there are no scalability problems when the model admits more clocks, variables etc. All of this was exploited in the case study by introducing a tailored decoration which is simply ignored when the same model is interpreted for non-deterministic analysis.

VII. CONCLUSIONS

This paper proposes an approach to modelling and analysis of Time Petri Nets (TPN) [2]-[5] also in the presence of stochastic features (sTPN) [14].

The approach is centered on a structural translation of TPN/sTPN onto the latest version of UPPAAL which enables both qualitative non-deterministic analysis based on exhaustive model checking, and quantitative evaluation of system properties by exploiting the statistical model checker.

The UPPAAL translation makes it possible, during analysis, to reason directly in the terms of TPN/sTPN vocabulary (e.g., marking reachability and transition firings), thus simplifying its practical usage.

Prosecution of the research is directed at:

- Extending the sTPN modelling through the support of generally distributed probability functions attached to transitions.
- Improving the automatic translation of TPN/sTPN models in the context of the TPN Designer toolbox [20].
- Applying the approach, e.g., to complex maintenance procedures, phased-mission systems [17], time-constrained workflow systems.
- Experimenting the methodology with the PRISM Probabilistic Model Checker [12].
- Specializing the approach to Preemptive Time Petri Nets [18], i.e., towards the schedulability analysis of real-time tasking sets under general conditions, when the schedulability problem becomes undecidable. A preliminary framework was prototyped in [19] using UPPAAL stopwatches and over-approximation. The goal is to allow non-deterministic qualitative analysis (when possible) and quantitative analysis of task response times using the statistical model checker.

REFERENCES

[1] E.M. Clarke, O. Grumberg, D.A. Peled, *Model checking*, MIT Press, 2000.

[2] P.M. Merlin, D.J. Farber, "Recoverability of communication protocols: implications of a theoretical study", *IEEE Trans. Commun.*, **24**(9):1036-1043, 1976.

[3] B. Berthomieu, M. Diaz, "Modeling and verification of time dependent systems using Time Petri Nets," *IEEE Trans. Soft. Eng.*, Vol. 17, No. 3, pp. 259-273, Mar. 1991.

[4] B. Berthomieu, M. Menasche, "An enumerative approach for analyzing Time Petri Nets," Information Processing: Proc. IFIP Congress 1983, R.E.A. Mason, ed., vol. 9, pp. 41-46, 1983.

[5] E. Vicario, "Static analysis and dynamic steering of time dependent systems using Time Petri Nets," *IEEE Trans. Soft. Eng.*, vol. 27, no. 8, pp. 728-748, Aug. 2001.

[6] R. Alur, D.L. Dill, "A theory of timed automata", *Theoretical Computer Science*, Vol. 126, pp. 183-235, 1994.

[7] B. Berthomieu, P.-O. Ribet, and F. Vernadat, "The Tool TINA—Construction of abstract state spaces for Petri Nets and Time Petri Nets," *Int. J. Production Research*, Vol. 42, No. 14, 2004.

[8] G. Bucci, L. Carnevali, L. Ridi, E. Vicario, "ORIS: a tool for modeling, verification and evaluation of real-time systems", *Int. J. on Software Tools for Technology Transfer*, Springer (2010) 12:391–403, DOI 10.1007/s10009-010-0156-8.

[9] UPPAAL on-line, www.UPPAAL.org.

[10] G. Behrmann, A. David, K.G. Larsen, "A tutorial on UPPAAL", In: *Formal Methods for the Design of Real-Time Systems*, M. Bernardo and F. Corradini Eds., Lecture Notes in Computer Science, Vol. 3185, Springer-Verlag, pp. 200-236, 2004.

[11] A. David, K.G. Larsen, A. Legay, M. Mikucionis, D.B. Poulsen, UPPAAL SMS Tutorial, *Int. J. on Software Tools for Technology Transfer*, Springer, **17**:1-19, 06.01.2015, DOI 10.1007/s10009-014-0361-y, 2015.

[12] M.Z. Kwiatkowska, G. Norman, D. Parker, "PRISM 4.0: Verification of Probabilistic Real-Time Systems". In *Proc. of CAV 2011*, pp. 585-591, 2011.

[13] H.L.S. Younes, "*Verification and planning for stochastic processes with asynchronous events"*, PhD Thesis, Carneige Mellon, 2005.

[14] E. Vicario, L. Sassoli, L. Carnevali, "Using stochastic state classes in quantitative evaluation of dense-time reactive systems", *IEEE Trans. on Soft. Eng.*, Vol 35, No. 5, pp. 703-719, 2009.

[15] M.A. Marsan, G. Balbo, G. Conte, S. Donatelli, G. Franceschinis, *Modelling with Generalized Stochastic Petri Nets*, John Wiley and Sons, 2004.

[16] F. Cicirelli, A. Furfaro, L. Nigro, "Model checking time-dependent system specifications using time stream Petri nets and UPPAAL", *Appl. Math. Comp.*, Vol. 218, pp. 8160-8186, 2012.

[17] L. Carnevali, M. Paolieri, K. Tadano, E. Vicario, "Towards the quantitative evaluation of phased maintenance procedures using non-Markovian regenerative analysis, *Lecture Notes in Computer Science*, Springer, Vol. 8168, pp 176-190, 2013.

[18] G. Bucci, A. Fedeli, L. Sassoli, and E. Vicario, "Timed state space analysis of real time preemptive systems," *IEEE Trans. Soft. Eng.*, vol. 30, no. 2, pp. 97-111, Feb. 2004.

[19] F. Cicirelli, A. Furfaro, L. Nigro, F. Pupo, "Development of a schedulability analysis framework based on pTPN and UPPAAL with stopwatches". In *Proc. of the 16th IEEE/ACM Int. Symp. on Distributed Simulation and Real Time Applications (DS-RT)*, pp. 57-64, 2012.

[20] L. Carullo, A. Furfaro, L. Nigro, F. Pupo. "Modelling and Simulation of Complex Systems using TPN Designer". *Simulation Modelling Practice and Theory*. **11**/7-8, pp. 503-532, 2003.