COLOURED PETRI NETS: TIMED STATE SPACE EXPLORATION EXAMPLES AND SOME SIMULATION SHORTAGES

M. A. Piera[1], G. Mušič[2]
[1]Autonomous University of Barcelona, Dept. of Telecommunication and Systems Engineering, Spain,
[2]University of Ljubljana, Faculty of Electrical Engineering, Slovenia

Corresponding author: M. A. Piera, Autonomous University of Barcelona, Dept. of Telecommunication and Systems Engineering, Barcelona, Spain,

Abstract. The paper deals with the problem of timed state space generation and exploration in the frame of simulation-optimization approach for discrete-event systems. Coloured Petri net representation of a system is considered and corresponding techniques of timed state space generation and timed simulation are addressed. It is shown that the established simulation techniques do not perform adequately in some application relevant examples since in general, only a subset of a timed state space of a simulated system is represented. Two examples are provided to illustrate the effect of timed state space reduction. While the optimal solution is preserved within the reduced state space in one example, in the second example this is not the case and the optimum is missed. This indicates that the timed simulation technique has to be carefully designed in order to be suitable for the simulation-optimization approach.

1 Introduction

Modern man-made technical systems are designated with high responsiveness that involves flexibility of operation. The increased flexibility leads to a number of possible operation scenarios. A corresponding analysis is required in order to support the decision-making procedures involved in planning the systems operation. This motivates a development in discrete-event modelling and analysis techniques that can provide an adequate analysis support.

Coloured Petri nets (CPNs) are a powerful framework for discrete-event modelling, simulation and analysis. In contrast to most system description languages, CPNs are state and action oriented at the same time - providing an explicit description of both the states and the actions [4]. Advantages of ordinary Petri nets, such as simple modelling of concurrency and synchronization [9], are enhanced by a compact representation of the model and the underlying state space by the use of coloured tokens and hierarchy. Furthermore, a functional programming language CPN ML is added, which is used for the net inscriptions, i.e. the text strings attached to the places, transitions and arcs of a CPN. This way various declarations are made and the language also facilitates modelling of data manipulation that is triggered by event occurrences. CPNs have been applied in a wide range of application areas, and many projects have been carried out in industry.

When CPN models are used for performance analysis, the time has to be included in the model. CPNs include a time concept that makes it possible to capture the time taken to execute activities in the system [5]. In this way CPNs can be applied for simulation-based performance analysis, i.e., timed simulation can be performed for testing the performance of the system under certain operating conditions.

The ability of performance measures investigation shows the possibility to use CPN models within the simulation-optimization approach. Within this approach, an optimization algorithm is used to arrange the simulation of a sequence of system configurations so that an optimal or near optimal system configuration could eventually be obtained [8, 7].

In order to be able to reach the optimum, it is important to be able to generate any possible trace of the system behaviour. It can be observed that in most of the discrete-event simulation tools this is not always the case. They are generally able to represent only a subset of timed state space of a simulated system. The paper deals with the critical evaluation of the timed state space generation strategies found in DES simulators. In particular CPN framework and the related time mechanisms are studied.

2 Timed state space exploration

For simplicity, a Place/Transition Timed Petri net will be considered in the following. It can be described as a bipartite graph consisting of two types of nodes, places and transitions. The nodes are interconnected by directed arcs. The state of the system is denoted by the distribution of tokens (called marking) over the places. Compared to CPN, no token colours nor arc inscriptions other than weight will be used, and transition inscriptions will be limited to specification of time delay. This will allow more focused presentation on time aspects while the mechanism of time inclusion will be kept as close as possible to the one of CPNs. This enables a straightforward extension of presented concepts to CPNs.

The concept of time is not explicitly given in the original definition of Petri nets. As described in [2], there are

three basic ways of representing time in Petri nets: firing durations (FD), holding durations (HD) and enabling durations (ED). The FD principle says that when a transition becomes enabled it removes the tokens from input places immediately but does not create output tokens until the firing duration has elapsed. In [10] a well-defined description of this principle is given. When using HD principle, a created token is considered unavailable for the time assigned to transition that created the token. The unavailable token can not enable a transition and therefore causes a delay in the subsequent transition firings. This principle is graphically represented in Figure 1, where the available tokens are schematized with the corresponding number of undistinguishable (black) tokens and the unavailable tokens are indicated by empty circles. The time assigned to a transition is written beside the transition, e.g., t_d is assigned to transition t_1. When the time is 0 this denotation is omitted. In Figure 1, t denotes a model time represented by a global clock and t_f denotes the firing time of a transition.

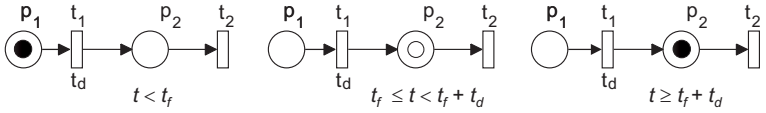

Figure 1: Timed Petri net with holding durations

HD and FD are in fact the same way of representing time while ED principle leads to a different timed behaviour of a model. With ED, the firing of the transitions happens immediately and the time delays are represented by forcing transitions that are enabled to stay so for a specified period of time before they can fire. The main difference between using HD and ED can be seen in a Petri net where a conflict appears. In this case more transitions are enabled by one marking. When ED policy is used, the firing of one transition can interrupt the enabling of other transitions, as the marking, which has enabled previous situation, has changed [2].

This indicates that ED concept is more general than HD. Furthermore, in [6] an even more general concept is used, which assigns delays to individual arcs, either inputs or outputs of a transition. This way both ED and HD concepts are covered, and the enabling delay may even depend on the source of transition triggering while holding delay may differ among different activities started by the same transition. This paper builds on some ideas presented in [6] although they are used in a different context. While the primary focus in [6] is on a development of a decentralized timed state space generation approach, here the formalization is simplified and the focus is on the practical implications of using various types of timed state spaces.

When modelling several performance optimization problems, e.g. scheduling problems, such a general framework as presented in [6] is not needed. It is natural to use HD when modelling most scheduling processes as transitions represent starting of operations, and generally once an operation starts it does not stop to allow another operation to start in between. HD principle is also used in timed version of CPNs. While CPNs allow the assignment of delays both to transition and to output arcs, we further simplify this by allowing time delay inscriptions to transitions only. This is sufficient for the type of examples investigated here, and can be generalized if necessary.

2.1 P/T Timed Petri net with holding durations

To include a time attribute of the marking tokens, which implicitly defines their availability and unavailability, the notation of [4] will be adopted. Tokens are accompanied with a timestamp, which is written next to the token number and separated from the number by @. E.g., two tokens with time stamp 10 are denoted 2@10. A collection of tokens with different time stamps is defined as a multiset, and written as a sum (union) of sets of timestamped tokens. E.g., two tokens with time stamp 10 and three tokens with timestamp 12 are written as 2@10+3@12. The timestamp of a token defines the time from which the token is available.

Time stamps are elements of a time set TS, which is defined as a set of numeric values. In many software implementations the time values are integer, i.e. $TS = \mathbb{N}$, but will be here admitted to take any positive real value including 0, i.e. $TS = \mathbb{R}_0^+$. Timed markings are represented as collections of time stamps and are multisets over TS: TS_{MS}. By using HD principle the formal representation of a P/T Timed Petri net is defined as follows.

$TPN = (P, T, I, O, M_0, f)$, where:

- $P = \{p_1, p_2, \ldots, p_k\}, k > 0$ is a finite set of places,
- $T = \{t_1, t_2, \ldots, t_l\}, l > 0$ is a finite set of transitions (with $P \cup T \neq \emptyset$ and $P \cap T = \emptyset$),
- $I : (P \times T) \to \mathbb{N}$ is the input arc function. If there exists an arc with weight k connecting p to t, then $I(p,t) = k$, otherwise $I(p,t) = 0$.
- $O : (P \times T) \to \mathbb{N}$ is the output arc function. If there exists an arc with weight k connecting t to p, then $O(p,t) = k$, otherwise $O(p,t) = 0$.
- $M : P \to TS_{MS}$ is the marking, M_0 is the initial marking of a timed Petri net.
- $f : T \to TS$ is the function that assigns a non-negative deterministic time-delay to every $t_j \in T$.

Functions I and O define the weights of directed arcs, which are represented by arc inscriptions. In the case when the weight is 1, this annotation is omitted, and in the case when the weight is 0, the arc is omitted. Let ${}^\bullet t \subseteq P$ denote the set of places which are inputs to transition $t \in T$, i.e., there exists an arc from every $p \in {}^\bullet t$ to t.

To determine the availability and unavailability of tokens, two functions on the set of markings are defined. The set of markings is denoted by $\mathbb{M}$. Given a marking and model time, $m : P \times \mathbb{M} \times TS \to \mathbb{N}$ defines the number of available tokens, and $n : P \times \mathbb{M} \times TS \to \mathbb{N}$ the number of unavailable tokens for each place of a TPN at a given time. Note that model time also belongs to time set TS.

Two timed markings can be added (denoted $+_T$) in a similar way as multisets, i.e. by making a union of the corresponding multisets. The definition of subtraction is somewhat more problematic. To start with, a comparison operator is defined. Let M_1 and M_2 be markings of a place $p \in P$. By definition, $M_1 \geqslant_T M_2$ iff $m(p,M_1,T_k) \geq m(p,M_2,T_k), \forall T_k \in TS$.

Similarly, the subtraction will be defined by the number of available tokens, and the subtrahend should not contain any unavailable tokens. Let M_1, M_2 and M_3 be markings of a place $p \in P$, $M_1 \geqslant_T M_2$, and $m(p,M_1,T_k)$, $m(p,M_2,T_k)$, and $m(p,M_3,T_k)$, be the corresponding numbers of available tokens at time T_k, and $n(p,M_2,T_k) = 0$. The difference $M_3 = M_1 -_T M_2$ is then defined as any $M_3 \in \mathbb{M}$ having $m(p,M_3,T_k) = m(p,M_1,T_k) - m(p,M_2,T_k)$.

Clearly, the subtraction is not uniquely defined this way, since it is not clear, which of the available tokens, eventually having different timestamps, will be removed from the minuend. However, the subtraction will only be used to define removal of tokens from input places when a transition is fired. If the HD principle is used and there are several available tokens in an input place with different timestamps, it is not important which of them will be removed. All the subsequent transition firings will only deal with present or future points in time, and all currently available tokens will also remain available for the eventual enablement of future firings. The situation would be changed, of course, if the ED timing principle was used, since there also the past is important for the enablement of transitions. For such a case, the subtraction operator is defined more generally in [6].

Even when applying HD principle, it can however be practical to uniquely define the substraction operation on timed markings. E.g. in this paper the comparison of each newly generated marking to all previously reached markings is applied when generating timed state space. This is simplified if the new markings are always generated in the same way, e.g., by always removing the token with the most recent timestamp first.

Using the above definitions, the firing rule of a TPN can be defined. Given a marked $TPN = (\mathcal{N}, M)$, a transition t is time-enabled at time T_k, denoted $M[t\rangle_{T_k}$ iff $m(p,M,T_k) \geq I(p,t)$, $\forall p \in {}^\bullet t$. An enabled transition can fire, and as a result removes tokens from input places and creates tokens in output places. If transition t fires, then the new marking is given by $M'(p) = M(p) -_T I(p,t)@T_k +_T O(p,t)@(T_k + f(t)), \forall p \in P$. If marking M_2 is reached from M_1 by firing t at time T_k, this is denoted by $M_1[t\rangle_{T_k} M_2$. The set of markings of TPN $\mathcal{N}$ reachable from M is denoted by $R(\mathcal{N}, M)$.

2.2 Classes of timed state spaces

In the definition of the firing rule of a TPN, the enabled transition is described as a transition that can fire. This is a common definition of the firing rule in the Petri net literature. Nothing is said about the exact moment of firing. In the timed state space generation, time of the firing is a key attribute. Depending on the more detailed definition of the time of transition firing one can distinguish among several classes of timed state spaces.

Most general timed state space [6] is $TSS = (N,A)$, where $N = R(\mathcal{N}, M_0)$ is a node set and A is the set of arcs given as $A = \bigcup_{M \in N} \{(M,t,M')_k | M[t\rangle_{T_k} M'\}$. In TSS a firing $M[t\rangle_{T_k} M'$ is not tied to any specific firing time T_k. This can be any time greater than T_{k0} when transition is enabled: $T_k \geq T_{k0}, M[t\rangle_{T_{k0}}, \nexists T_{k1} < T_{k0}, : M[t\rangle_{T_{k1}}$.

To better define the firing time it may be required that a transition fires at the earliest possible time. This way an earliest time state space is defined: $ESS = (N,A)$, where $N = R(\mathcal{N}, M_0)$ and A is given as $A = \bigcup_{M \in N} \{(M,t,M')_k |$ $M[t\rangle_{T_k} M'$, $\nexists T_{k1} < T_k, : M[t\rangle_{T_{k1}}\}$.

In ESS a firing $M[t\rangle_k M'$ is tied to the earliest firing time at a given marking. But this does not prevent the inclusion of other transitions from the same marking into the ESS. E.g. if two transitions t_1 and t_2 are in conflict and t_1 is enabled before t_2, also t_2 participates in the ESS.

This possibility is eliminated by reduced earliest time state space RSS, which only allows the inclusion of transitions in conflict that can fire at the same time. It is defined as $RSS = (N,A)$, where $N = R(\mathcal{N}, M_0)$ and A is given as $A = \bigcup_{M \in N} \{(M,t,M')_k | M[t\rangle_{T_k} M', \nexists t_1, T_{k1} < T_k, : M[t_1\rangle_{T_{k1}}\}$.

Most of the timed state space generating algorithms found in various software tools actually generate RSS. An example is CPN Tools software [3, 5].

For a P/T Timed Petri net the three classes of time state spaces are related as follows: $RSS \subseteq ESS \subset TSS$.

Next, algorithms for ESS and RSS generation are briefly sketched. For simplicity, the algorithms will be presented for Place/Transition Timed nets only. The extension to CPNs is straightforward.

Algorithm 1 ESS generation

set(max_time);
ESS.N := $\{M_0\}$;
ESS.A := $\emptyset$;
U := $\{M_0\}$;
$T_k := 0$;
while $U \neq \emptyset$ **do**
  for all $M \in U$ **do**
    T_k := occurence_time(M);
    repeat
      for all $t \in T : M[t\rangle_{T_k} M' \wedge \nexists T'_k < T_k : M[t\rangle_{T'_k}$ **do**
        if $M' \notin ESS.N$ **then**
          ESS.N := ESS.N $\cup \{M'\}$;
          U := U $\cup \{M'\}$;
        end if
        ESS.A := ESS.A $\cup \{(M,t,M')_{T_k}\}$;
      end for
      T_k := next_time_hit(T_k, M);
    until $n(p,M,T_k) = 0, \forall p \in P \vee T_k > max\_time$
    U := U - $\{M\}$;
  end for
end while

The algorithm for generating ESS is shown in Algorithm 1. It is assumed that reached markings are stored with an additional attribute, i.e. the time of their occurrence. When comparing a new marking to previously generated nodes of the state space, this attribute is also taken into account, i.e. two identical marking that occurred at different times are considered different. Such an implementation is used in order to be able to distinguish behaviours that produce the same event (sub)sequence at different times. The function "occurrence_time(M)" returns a value of this attribute associated with marking M. The function "next_time_hit(T_k, M)" returns the value of the lowest timestamp value in M greater than T_k. Parameter max_time is used to define a time limit beyond which the transition firings are not explored.

The algorithm for generating RSS is shown in Algorithm 2. It differs in the way unavailable tokens are treated at the currently processed marking. The time is not incremented to wait for the tokens to become available at this point, but only after all the markings reachable in one step are determined. This way only the earliest transition(s) fireable from a given marking are kept. To enable this, the domain of function "next_time_hit(T_k, M)" is extended from a single marking to the whole set of timestamps included in the set of unexplored markings U.

Algorithm 2 RSS generation

set(max_time);
RSS.N := $\{M_0\}$;
RSS.A := $\emptyset$;
U := $\{M_0\}$;
$T_k := 0$;
while $U \neq \emptyset \wedge T_k \leq max\_time$ **do**
  repeat
    U' := U;
    for all $M \in U$ **do**
      for all $t \in T : M[t\rangle_{T_k} M'$ **do**
        if $M' \notin RSS.N$ **then**
          RSS.N := RSS.N $\cup \{M'\}$;
          U := U $\cup \{M'\}$;
        end if
        RSS.A := RSS.A $\cup \{(M,t,M')_{T_k}\}$;
        U := U - $\{M\}$;
      end for
    end for
  until $U = U'$
  T_k := next_time_hit(T_k, U);
end while

Simulation can be regarded as an exploration of a single path in the timed state space of the model. Most of the available simulation packages for TPNs or CPNs are based on RSS generation principle. Of course only one transition is chosen to fire at every simulation step. Clearly, this rules out some of the possible paths in the timed state space. When using simulation for performance optimization, this may cause the obtained solutions being not optimal. This will be demonstrated by examples in the following sections.

3 RSS policy applied to the Job Shop Scheduling Problem

The well known Job Shop Scheduling Problem (JSSP), consists to schedule a group of jobs in a set of machines, subject to the restriction that each machine can process only one job at a time and each job has an order of processing specified in each machine, and leads to a typology of NP-hard problems.

There is a high number of instances for different numbers of jobs and machines [1], which usually are considered by the optimization community as a benchmarking problem to test algorithms and combinatorial optimization methods in which the main goal is to minimize the time of completion of all jobs (makespan).

Recently, the JSSP has also been used by the simulation community to test simulation-optimization in which random variables are used to specify the different operation times. In [8] a state space exploration algorithm to tackle the state space explosion by means of heuristics and deal with the optimal solution is presented.

To illustrate the RSS policy shortages, a Job shop production system with two machines and two jobs will be considered. Figure 2 illustrates graphically the system in which Job 1 is represented by a ring and Job 2 is represented by a cylinder (transport subsystems are not considered relevant to the problem).

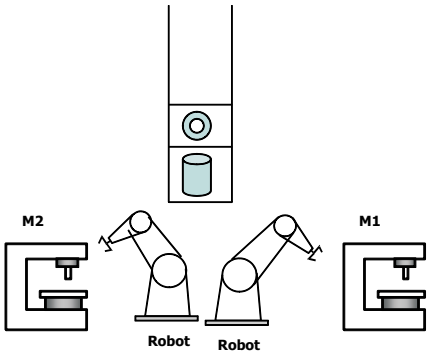

Figure 2: Job Shop 2x2

The machine sequence of each job is known (see Table 1), and each machine can serve only one job simultaneously.

Job 1		Job 2	
Machine	Operation Time	Machine	Operation Time
1	3	2	3
2	6	1	3

Table 1: Machine sequence

Figure 3 illustrates the Petri Net model, in which:

- Transition $\mathbf{T_1}$ is used to describe machine M1 processing Job 1.
- Transition $\mathbf{T_2}$ is used to describe machine M1 processing Job 2.
- Transition $\mathbf{T_3}$ is used to describe machine M2 processing Job 1.
- Transition $\mathbf{T_4}$ is used to describe machine M2 processing Job 2.
- Place **M1** is used to describe machine M1 free.
- Place **M2** is used to describe machine M2 free.
- Place **J1O1** is used to describe Job 1 waiting for the first operation in machine M1.
- Place **J1O2** is used to describe Job 1 waiting for the second operation in machine M2.
- Place **J2O1** is used to describe Job 2 waiting for the first operation in machine M2.
- Place **J2O2** is used to describe Job 2 waiting for the second operation in machine M1.
- Place **Prod** is used to represent the processed jobs.

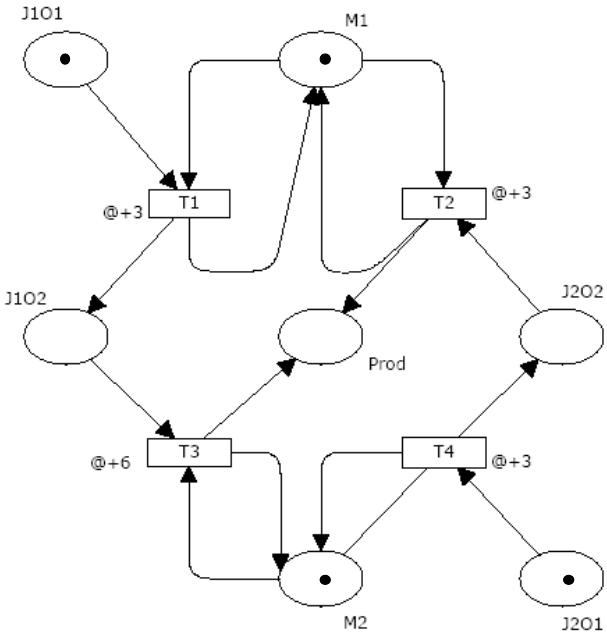

Figure 3: Petri Net model for the JS2x2

According to the discrete event system representation proposed for the JS2x2, the different plans to process the 2 jobs can be found in the state space (ESS) represented in Figure 4 (Node $\mathbf{M_5}$ is not expanded because it is equivalent to node $\mathbf{M_4}$). Node $\mathbf{M_0}$ represents the initial marking, and nodes $\mathbf{M_{11}}, \mathbf{M_{12}}, \mathbf{M_{13}}$ and $\mathbf{M_{14}}$ represent the final markings in which both jobs have been processed. As it can be easily seen, both jobs could be processed in 9 time units (nodes $\mathbf{M_{12}}, \mathbf{M_{13}}$) by taking the advantage that machine M1 and M2 can work in parallel (consider for example sequence $\mathbf{T_1 - T_4 - T_3 - T_2}$). However, a bad planning policy could lead a make-span of 15 time units (nodes $\mathbf{M_{11}}, \mathbf{M_{14}}$) in which all operations are performed sequentially (consider for example sequence $\mathbf{T_1 - T_3 - T_4 - T_2}$). In case a RSS policy would be used to analyze the state space of the JS2x2 system, marking $\mathbf{M_1}$ would not be able to reach marking $\mathbf{M_3}$. In a similar way, marking $\mathbf{M_2}$ would not be able to reach marking $\mathbf{M_6}$. Figure 5 illustrates the RSS obtained by means of CPN tools software.

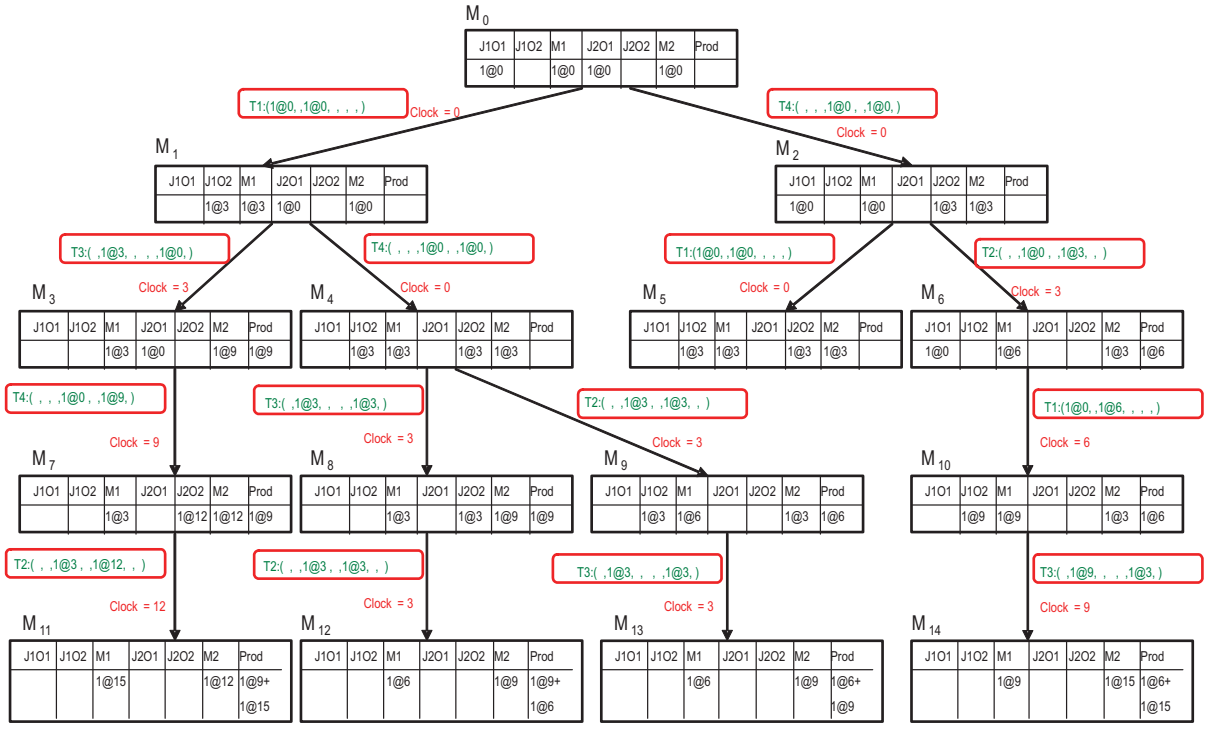

Figure 4: State space (ESS) for the JS2x2

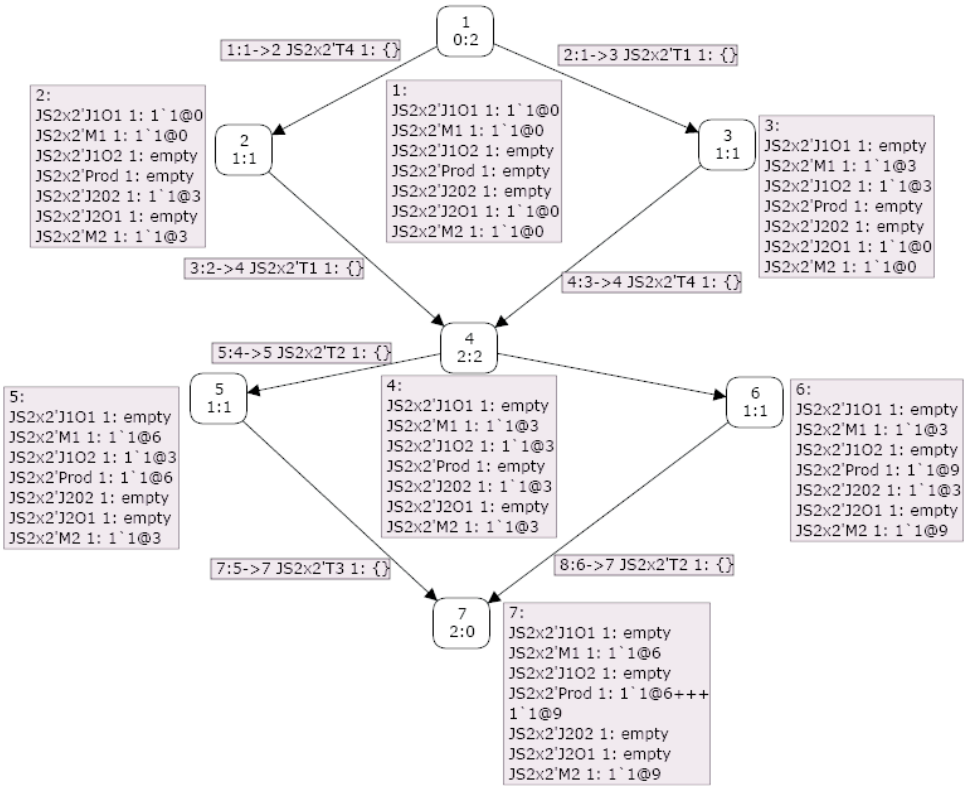

Figure 5: RSS for the JS2x2

4 Some shortages of RSS policy in the frame of simulation-optimization approach

The basic idea underlying the use of CPN models for performance analysis by means of dynamic generation of the states space, is to translate a problem of planning, scheduling or routing into a search problem in the states space of the system. An academical scheduling problem is presented in this section to show up that the RSS policy can not be used to guarantee optimality because some states are missed.

Figure 6 illustrates a production system with a stock of raw material and two machines (transport subsystems dynamics are neglected) with the same characteristics but with different performances: Machine M2 is a old generation machine that still works but with a rate $1/10^{th}$ of a new CNC machine (ie M1). The problem consist to deal with the optimal schedule to process 3 details using 2 machines (M1 and M2) in which the processing time of a detail in machine M1 is 3 time units while the processing time in machine M2 is 30 time units.

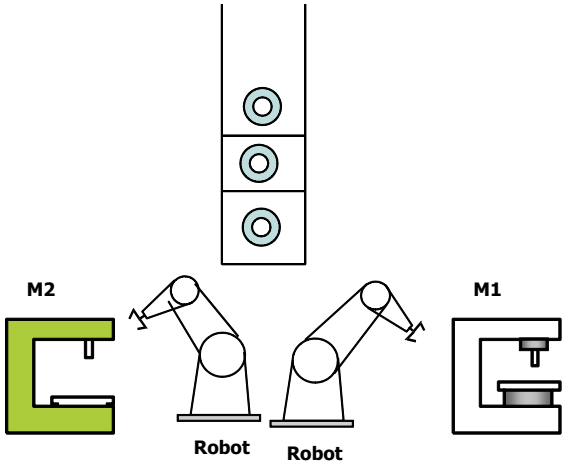

Figure 6: Different performance production subsystems

Figure 7 illustrates the petri net model of the production system in which transition T1 represents machine M1 processing a detail while transition T2 represents machine M2 processing a detail.

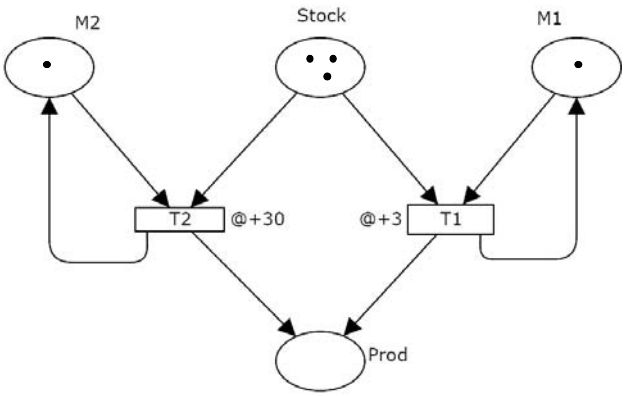

Figure 7: Petri net of the production system

Figure 8 illustrates the state space (ESS) of the system in which it is easy to note that the optimal policy would assign the whole workload to machine M1 (sequence $\mathbf{T_1 - T_1 - T_1}$) obtaining a make-span of 9 time units (marking $\mathbf{M_7}$), while the worst policy requires 90 time units (marking $\mathbf{M_{14}}$) which is obtained by assigning the whole workload to machine M2 (sequence $\mathbf{T_2 - T_2 - T_2}$).

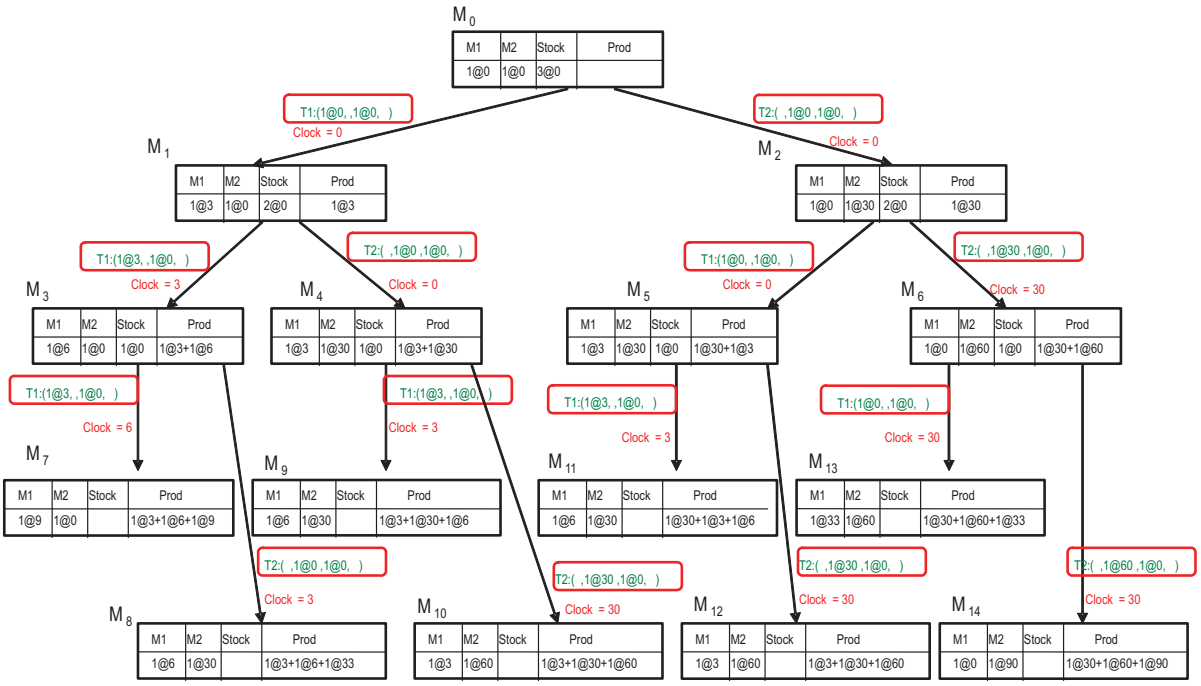

Figure 8: State Space (ESS) for the production system

As it can be checked in Figure 9, the use of reduced earliest time state space algorithms to analyse the state space of a CPN model, only can deal with a policy that requires 30 time units to process the 3 details.

5 Conclusions

When using state space analysis for optimization of discrete-event systems it is important to be able to generate any possible trace of the system behaviour. Corresponding software tools are often based on the reduced earliest time state space generation principle, which rules out some of the possible paths in the timed state space. Similarly, discrete event simulation is based on a similar principle, i.e., the scheduled event list is ordered on a smallest-scheduled-time-first basis and only the first event in the list occurs in the next simulation step. This way some of the possible behaviour scenarios are not explored and optimal solution can be missed. A simple illustrating example is presented in this paper. This indicates that the underlying simulation technique has to be carefully designed in order to be suitable for the simulation-optimization approach.

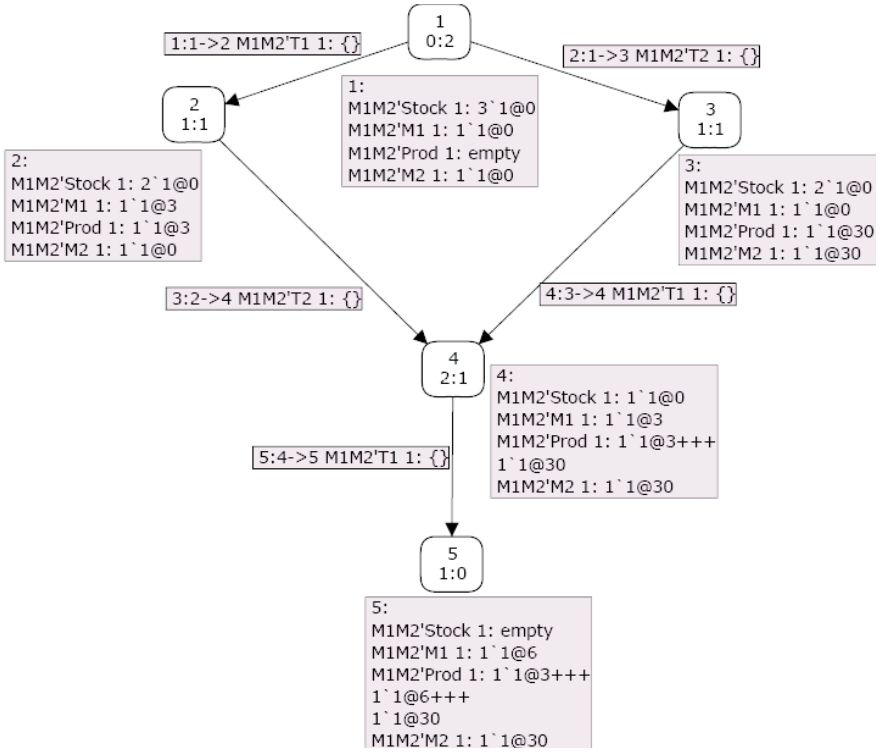

Figure 9: RSS for the production system

6 References

[1] Beasley, J.E.: *Or-Library*. 2005. http://people.brunel.ac.uk/˜mastjjb/jeb/info.html, http://people.brunel.ac.uk/˜mastjjb/jeb/orlib/files/jobshop1.txt

[2] Bowden, F.D.J.: *A Brief Survey and Synthesis of the Roles of Time in Petri nets*. Mathematical and Computer Modelling 31 (2000), 55–68.

[3] CPN Tools home page: http://www.daimi.au.dk/CPNTools/.

[4] Jensen, K.: *Coloured Petri Nets: Basic Concepts,Analysis Methods and Practical Use, vol. 1, 2nd ed.* Springer-Verlag, Berlin, 1997.

[5] Jensen, K., Kristensen, L.M., and Wells L.: *Coloured Petri Nets and CPN Tools for Modelling and Validation of Concurrent Systems*. International Journal on Software Tools for Technology Transfer, 9 (2007), 213–254.

[6] Lakos, C., and Petrucci, L.: *Modular state space exploration for timed petri nets*. International Journal on Software Tools for Technology Transfer, 9 (2007), 393–411.

[7] Mujica, M., Piera, M.A., and Narciso, M.: *Optimizing Time Performance in Reachability Tree-based Simulation*. The 20th European Modeling & Simulation Symposium, Campora S. Giovanni (Amantea, CS), Italy, 2008, 800–805.

[8] Piera, M.A., Narciso, M., Guasch, A., and Riera, D.: *Optimization of Logistic and Manufacturing Systems through Simulation: A Colored Petri Net-Based Methodology*. SIMULATION, Transactions of The Society for Modeling and Simulation International, 80 (2004), 121–129.

[9] Proth, J., Xie, X.: *Petri nets. A Tool for Design and Management of Manufacturing Systems*. John Wiley & Sons Ltd., 1996.

[10] Zuberek, W.M.: *Timed Petri nets. Definitions, Properties and Applications*. Microelectronics and Reliability, 31 (1991), 627–644.