

An algorithm for deriving critical scenarios in mechatronic systems

Sarhane Khalfaoui^{1,2}, Hamid Demmou¹, Edwige Guilhem², Robert Valette¹

¹Laboratoire d'Analyse et d'Architecture des Systèmes LAAS CNRS
7, avenue du Colonel Roche, 31077 Toulouse cedex, France
{hamid, robert}@laas.fr

²PSA Peugeot Citroën, Direction des Systèmes d'Information
18, rue des Fauvelles, 92256 La Garenne Colombes cedex, France
{sarhane.khalfaoui, edwige.guilhem}@mpsa.com

Abstract-- To evaluate the reliability of mechatronic systems, the feared scenarios should be known in order to choose the safe architecture of the system, during the development phase. The aim of this work is to propose an algorithm for deriving the critical scenarios from a Petri net model. These scenarios characterise how the system leaves the normal operation to go to the feared state by determining the sequences of actions and state changes leading to a dangerous situation.

Keywords: Petri net, mechatronic systems, safety, reliability in design.

I. INTRODUCTION

Mechatronic systems combine electrical, mechanical, hydraulic and electronic technologies associated with a computer control. This computer particularly controls the operation modes and allows the development of new safety functions. Mechatronic systems are hybrid. In fact, continuous dynamic is related to energetic part and the discrete dynamic is introduced by the computer control.

This paper deals with safety in design of mechatronic systems. For this purpose, it is important to characterize feared behaviors (which are critical) in the early design stage. As they have to be rare (security constraints), simulation is typically insufficient because only nominal behaviors are explored [1].

In order to help designers taking into account safety constraints, the feared behaviors have to be directly derived from a system model. Qualitative and quantitative analyses of these behaviors are necessary to select good architectures. The qualitative analysis points out all the behaviors leading to states in which the motorist and the passengers safety is no longer guaranteed. It is this point that is addressed in the paper.

As the system is hybrid, we have chosen a modeling technique that associates Petri Nets and Differential Algebraic Equations [2]. The Petri Net model contains operation modes, fault occurrences and safety functions. That is why the qualitative analysis (feared scenario derivation) can be based on this model.

The qualitative analysis based on the reachability graph comes up with the combinatorial explosion of the search space [1]. This explosion is partly due to the fact that complex systems are inherently concurrent. The components involved in the feared scenario evolve in parallel with the remaining part of the system. A way to avoid this problem is to reason only upon these components. For this reason, we

use directly the Petri net model without deriving the reachability graph.

In [3], we presented a method for deriving critical scenarios. In this method, based on two steps (backward and forward reasonings), we use Linear logic [4] to analyse the Petri model. The principal of this method is to progressively enrich the knowledge of the context in which the interaction (a set of events occurring in a partial order) between some components leads to the feared state.

In this paper, we present the algorithm that is the hard core of the method. It allows the derivation of partial orders and the enrichment of the current marking. This enrichment means that new components are involved in the feared interaction (critical scenario).

The method and the algorithm are illustrated by means of a simple example.

II. A METHOD FOR DERIVING CRITICAL SCENARIOS

We call scenario a set of events (here transition firings) leading from one partial state (here partial marking) to another one and verifying a partial order. As we have stated in the introduction, we assume that the system is made up of a set of components. A partial state is the conjunction of the states of a subset of these components.

Definition: A partial order is defined by a directed graph (E, A) where the nodes E are a set of transition firings and the arcs A are pairs (ti, tj) such that ti precedes tj .

There is a partial order between tk and tl if and only if there is an arc between tk and tl .

The aim of this method is to derive all the scenarios, composed of actions and states, leading to a feared state and to analyse precisely why the system leaves the normal operation to go to the feared state.

Starting from a partial knowledge of the scenario that leads to the feared partial state, we progressively enrich this knowledge by analysing the scenario and either introducing components states necessary to its occurrence or considering other components states that forbid it. In the first case, this is formalised by means of adding new tokens necessary to fire transitions within the scenario. In the second case, we add new tokens in order to fire transitions in conflict with transitions in the scenario.

This method is made up of two steps: a backward and a forward reasoning. The backward reasoning starts from the partial feared state in order to derive the events that are necessary to reach it, and gives the last nominal states

preceding the abnormal behavior. The forward reasoning starts from these nominal states, enriches the scenario and points out the bifurcations between it and the normal operation.

An important point to underline is that the approach is based on Linear logic and sequent calculus. In the context of the paper, a sequent is composed of a partial initial marking, a final one and a list of transition firings likely to lead from the initial marking to the final one. Proving the sequent is equivalent to deriving one partial order among these transition firings, ensuring the effective reachability of the final marking [5]. This approach differs from the classical reachability analysis because it directly handles partial orders and not firing sequences that are totally ordered lists of transition firings. That is why the approach takes into account true concurrency and avoids the combinatorial explosion due to interleaving. A unique partial order encapsulates the set of firing sequences that are all the projections of the partial order.

The presented algorithm is not simply a sequent prover. In fact, for both backward and forward reasonings, the starting point is a partial knowledge of the initial and final markings and the list of transition firings is unknown. The supplementary issues to be addressed are the enrichment process (adding new tokens) and the end of reasoning criterion (fixing the transition firing list).

We have chosen to implement the backward reasoning as a reachability problem between two markings. This is done by reversing the arcs of the Petri net model. Doing so, backward and forward reasonings are similar and it is why the algorithm is the same for both of them.

Before presenting the scenario derivation algorithm, we will first introduce the Petri net example on which we will illustrate the different steps of the algorithm later.

III. EXAMPLE

The Petri net shown in Fig. 1 represents an equipment that can have three states: stopped (place OFF), in nominal operation (place N) or in a dangerous state (place D). An actuator participates to the nominal behavior (place Ac) when it is in a normal state (place Ac). Transition t_1 initializes the equipment. Transition t_2 corresponds to the evolution to the feared state. Transition t_3 corresponds to the nominal behavior and t_4 represents a fault occurrence of the actuator. This example will be used in the sequel to illustrate the method and the algorithm. Let us start by an illustration of a partial order in the presented approach.

Consider an initial marking with a token in place OFF and a final marking with a token in place D. To reach the final marking, one should first fire transition t_1 and then transition t_2 . Therefore, there is a precedence relation between t_1 and t_2 and $(t_1, t_2) \in A$ (an arc is connecting t_1 to t_2). This precedence relation is the direct consequence of the fact that the token in place N is produced by the firing of t_1 and consumed by t_2 . Hence, (t_1, t_2) corresponds to the presence of a token in place N.

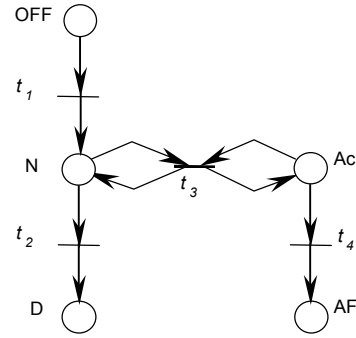


Fig. 1. Petri net example

In general, each element (t_i, t_j) of set A can be assimilated to the token produced by the firing of t_i and consumed by the firing of t_j .

In order to treat all the tokens similarly, the tokens of the initial markings are considered as produced by an initial event i , and those of the final marking are considered as consumed by a final event f .

The obtained partial order is illustrated in Fig. 2 with $E = (i, f, t_1, t_2)$ and $A = \{(i, t_1), (t_1, t_2), (t_2, f)\}$:

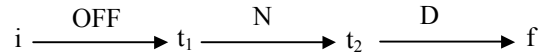


Fig. 2. An example of partial order

IV. THE ALGORITHM

In this section, we will describe the data structures and the different steps of the algorithm. The explanation of these steps will be given in the next section by means of the illustrative example in section 5.

A. Specification of Data Structures

1) Input data

The input data used by the algorithm is composed of the list of the initial tokens (L_i , the partial state that initiates the reasoning) and the list of nominal partial states (L_n) that will be used as the stop criterion.

2) Output data

The algorithm generates a set of partial orders. Each partial order i is defined by a couple $(E(i), A(i))$ and we have to add to it the list of the tokens ($Le(i)$) used to enrich the marking.

3) Internal data structure

- L_c is the current list of tokens, which is updated by the algorithm after each transition firing: the consumed tokens are removed and the produced ones added. From a logical point of view, it is a list of independent logic atoms. It corresponds to a current step in Linear logic [5]. In order to derive the set A , each token is denoted by a pair (e, p)

where e is the event producing the token and p its location. For instance, in the example in Fig. 2, the initial value of L_c is (i, OFF) and after the firing of t_1 , L_c is (t_1, N) .

Let us now define the collection of sets partitioning the transitions that have to be considered for a given L_c :

- T_{ewc} is the list of enabled transitions that are not in conflict with another transition. For example, for the initial marking in the net in Fig.1, transition $t_1 \in T_{ewc}$.
- T_{pewc} is the list of *potentially enabled* transitions that are not in conflict with another transition.

Definition: A transition is potentially enabled if and only if one at least of its input places contains a token and one at least of its input places is empty.

Let us consider the net in Fig.1 with the marking such that there is a token in place N and no token in place Ac . Transition t_3 is *potentially enabled*. It doesn't belong to T_{pewc} , however, because it is in conflict with transition t_2 .

- T_{ece} is the list of enabled transitions that are in conflict with enabled transitions only. In the situation just depicted above, t_2 doesn't belong to T_{ece} because it is in conflict with t_3 , which is potentially enabled. On the contrary, if the marking consists of one token in N and one token in Ac , then t_2 belongs to T_{ece} because t_3 is enabled.
- T_{ece} is the list of enabled transitions that are in conflict with at least one potentially enabled transition. It is the case of transition t_2 when the marking is composed of one token in N and no token in Ac .
- T_{pec} is the list of potentially enabled transitions in conflict with transitions that are either enabled or potentially enabled. It is the case of transition t_3 in the situation depicted just above.

Note that all of these sets of transitions are disjoint.

To avoid deriving the same partial order several times, we introduce the following list:

- L_{ft} is the list of forbidden transitions (which can't be fired from a given current list because they were fired in an other partial order).

Each time a conflict is encountered during a partial order derivation, the partial order has to be broken down into two partial orders. One is the continuation and the other will be derived later.

- C contains the necessary data to derive the new partial order. It is composed of the current list of tokens just before firing the transition involved in this conflict, the list of forbidden transitions, the current state of the partial order and the current list of the enriched tokens. C is a list of elements of the form: (L_c, L_{ft}, E, A, L_e) .

B. Procedures:

Let us first define a set of routines that are useful for the algorithm:

1) Fire_Transition(tk)

From a logical point of view, this procedure is executed when we establish a causal relation between some atoms (tokens from a Petri net point of view) in L_c , those that are necessary hypotheses for an event (enabling transition tk), and other ones, which are the consequences of this event (produced by the transition tk).

Step FT.1: (storing the fired transition tk)

Add tk in E

Step FT.2: (unmarking of input places and storing arc (ti,tk) in A)

For each atom (ti,p) necessary to fire tk , remove (ti,p) from L_c and add (ti,tk) in A

Step FT.3: (marking the output places of tk)

For each output place pi of tk add an atom (tk,pi) in L_c

2) Enrich_Marking_1(tk) (enrichment of tj which is in conflict with tk)

When, during a reasoning about a set of components, we meet a potentially enabled transition in conflict with other transitions, this means that the conflict involves at least a component which was not till now considered. In order to include it, we have to add new tokens in L_c , so that the potentially enabled transition became enabled.

L is a private list of tokens in the routine. It is initially empty.

Step EM.11: (marking empty input places of tj)

For each transition tj in conflict with tk and for each empty input place pl of tj , add a token (ek, pl) in L (ek is the event corresponding to the enrichment of marking of tk)

Step EM.12:

Check the consistency of the introduction of the tokens in L (we have to check that the component is not already considered in another state, this can be done by means of p -invariant in the Petri net). Delete the inconsistent tokens from L .

Step EM.13: (storing new tokens)

Add the tokens in L to the lists L_c and L_e

3) Enrich_Marking_2(tk) (enrichment of tk)

When, during a reasoning about a set of components, we meet a potentially enabled transition, this means that the corresponding event involves at least a component which was not till now considered. In order to include it, we have to add new tokens in L_c .

L is a private list of tokens in the routine. It is initially empty.

Step EM.21: (marking empty input places of tk)

For each empty input place pl of tk add a token (ek, pl) in L (ek is the event corresponding to the enrichment of marking of tk)

Step EM.22:

Check the consistency of the introduction of the tokens in L (we have to check that the component is not already considered in another state, this can be done by means of p -invariant in the Petri net). Delete the inconsistent tokens from L .

Step EM.23: (storing new tokens)

Add the tokens in L to the lists L_c and L_e

4) *Store_Context(tk)*

As it has been mentioned above, each time a conflict is encountered during a partial order derivation, the partial order has to be broken down into two partial orders. One is the continuation and the other is derived later. This routine stores the necessary context for a conflict involving transition tk in order to derive other partial orders later.

Step SC.1: Add tk to Lft

Step SC.2: Add a new tuple (Lc, Lft, E, A, Le) in C

Step SC.3: Delete the content of Lft

C. *The algorithm:*

Initial step

This step initializes C with one tuple (Lc, Lft, E, A, Le) in order to derive the first partial order where:

Lc = Li

Lft and Le are empty.

E = {i}

A = Lc

inc = 1.

Step 1

// Derive new partial orders //

If C is empty then go to Final step.

Else store the first tuple of C in (Lc, Lft, A, E, Le) and delete it from C.

Go to step 2

Step 2

Generate from (Lc, Lft, A, E, Le) all the enabled or potentially enabled transitions.

Generate the following lists: Tewc, Tpewc, Tece, Tecpe and Tpec.

Delete from all these lists:

- The transitions in E (to avoid infinite loops)
- The transitions in Lft and all those which are parallel to them (to avoid deriving the same partial order several times).

Go to step 3

Step 3

// The stop criterion //

If Lc contains only tokens from Ln that are not initial ones, or if Tewc, Tpewc, Tece, Tecpe and Tpec are empty then go to step 9

Else go to Step 4

Step 4

// The enabled transitions that are not involved in a conflict are fired first because no decision-making is required //

If Tewc is empty then go to Step 5

Else

Let tk be the first transition of Tewc

Fire_Transition (tk)

Go to Step 2

Step 5

// It resolves the conflicts by firing one transition and stores the necessary information to derive other partial orders //

If Tece is empty then go to step 6

Else

Let tk be the first transition of Tece

Store_Context (tk)

Fire_Transition (tk)

Go to step 2

Step 6

// We enrich the context only when all the decisions that do not require enrichment have been made //

If Tecpe is empty then go to step 7

Else

Let tk be the first transition of Tecpe

Enrich_Marking_1 (tk)

If tk is now in conflict with at least one enabled transition then Store_Context (tk)

Fire_Transition (tk)

Go to step 2

Step 7

If Tpec is empty then go to step 8

Else

Let tk be the first transition of Tpec

Enrich_Marking_2 (tk)

Store_Context (tk)

Fire_Transition (tk)

Go to step 2

Step 8

// In this step, we enrich the partial marking in order to make the potentially enabled transitions without conflict effectively enabled and we fire them next //

If Tpewc is empty then go to next step

Else

Let tk be the first transition of Tpewc

Enrich_Marking_2 (tk)

Fire_Transition (tk)

Go to step 2

Step 9

For each atom (ti,p) of Lc add (ti,f) in A (f is the final event).

Store the derived partial order, noted inc, with: E(inc) = E, A(inc) = A, Le(inc) = Le.

inc = inc + 1

Go to step 1

Final step

// The end of the algorithm //

All the partial orders are derived.

V. ILLUSTRATION OF THE ALGORITHM ON THE EXAMPLE

Now, let us illustrate the algorithm of scenario derivation on the Petri net of Fig.1. Our method aims at pointing out the critical scenarios and the bifurcations between the normal states and the feared one. We are now going to describe how the algorithm is applied in order to identify the scenarios leading to the marking of place D that is considered as a feared state. The nominal states are the partial markings one token in place OFF, one token in N and one token in Ac: Ln = {OFF, N, Ac}.

This approach is made up of two steps: a backward and then a forward reasoning.

A. Backward reasoning

The algorithm is applied on the example of Fig.1 in which all the arcs are reversed:

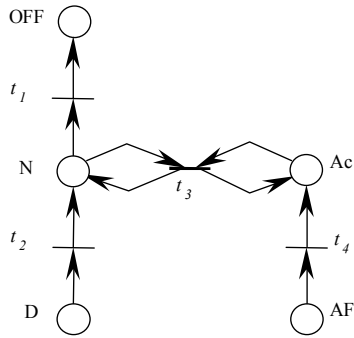


Fig. 3. Petri net in which the arcs are reversed

Initial step

We have: $L_c = L_i = \{(i, D)\}$, $L_{ft} = \{\}$, $L_e = \{\}$, $E = \{i\}$, $A = \{(i, D)\}$, $inc = 1$ and $C = (L_c, L_{ft}, E, A, L_e)$.

Step 1

- C is not empty then $(L_c, L_{ft}, E, A, L_e) = (\{(i, D)\}, \{\}, \{i\}, \{(i, D)\}, \{\})$.
- C becomes empty.

Step 2

$T_{ewc} = \{t2\}$, $T_{ece} = \{\}$, $T_{ecpe} = \{\}$, $T_{pec} = \{\}$.

Step 3

L_c doesn't contain only tokens from L_n (token $D \notin L_n$). We go then to step 4.

Step 4

$T_{ewc} (= \{t2\})$ is not empty.

Let $t_k = t2$

Fire_Transition($t2$):

Step FT.1: $E = \{i, t2\}$

Step FT.2: $L_c = \{\}$ and $A = \{(i, t2)\}$

Step FT.3: $L_c = \{(t2, N)\}$

Go to step 2

Step 2

$T_{ewc} = \{\}$, $T_{ece} = \{\}$, $T_{ecpe} = \{t1\}$, $T_{pec} = \{t3\}$.

Step 3

$L_c (= \{t2, N\})$ contains only tokens from L_n then go to step 9

Step 9

The derived partial order is defined by: $E1 = \{i, t2\}$, $A1 = \{(i, t2), (t2, f)\}$ and $L_e = \{\}$.

Go to step 1

Step 1

C is empty then end of the algorithm.

B. Forward reasoning

The algorithm is applied on the example of Fig.1. This step starts from the nominal state N identified in the backward reasoning. So we have $L_i = \{(i, N)\}$. $L_n = \{OFF, N, Ac\}$.

We just give some illustrative points of the algorithm execution.

We have $L_c = \{(i, N)\}$.

Step 2

$T_{ewc} = \{\}$, $T_{ece} = \{\}$, $T_{ecpe} = \{t2\}$, $T_{pec} = \{t3\}$.

Step 6

$T_{ecpe} (= \{t2\})$ is not empty.

Let $t_k = t2$ (the only one transition in the set)

Enrich_Marking_1($t2$):

Initially $L = \{\}$

Step EM.11:

The only transition in conflict with $t2$ is $t3$. Ac is the input place of $t3$ that is not marked. Let's add a token $(e2, Ac)$ in L : $L = \{(e2, Ac)\}$.

Step EM.12:

We have introduced a token in place Ac . This marking enrichment is consistent because we respect the p-invariant: $Ac + AF = 1$ (place AF is not marked).

Step EM.13:

$L_c = \{(i, N), (e2, Ac)\}$

$L_e = \{(e2, Ac)\}$

Store_Context($t2$):

$L_{ft} = \{\}$

Step SC.1: Add $t2$ to L_{ft}

Step SC.2:

$C = (L_c = \{(i, N), (e2, Ac)\}, L_{ft} = \{t2\}, E = \{i, e2\}, A = \{\}, L_e = \{(e2, Ac)\})$.

Step SC.3: Delete the content of L_{ft} : $L_{ft} = \{\}$.

Fire_Transition($t2$):

Step FT.1: $E = \{i, e2, t2\}$

Step FT.2: $L_c = \{(e2, Ac)\}$. $A = \{(i, t2)\}$

Step FT.3: $L_c = \{(t2, D), (e2, Ac)\}$

Go to step 2

Step 2

We have: $L_c = \{(t2, D), (e2, Ac)\}$, $L_{ft} = \{\}$, $E = \{i, e2, t2\}$, $A = \{(i, t2)\}$ and $L_e = \{(t2, Ac)\}$. We deduce, then, that: $T_{ewc} = \{\}$, $T_{ece} = \{\}$, $T_{ecpe} = \{t4\}$, $T_{pec} = \{t3\}$.

Go to step 1

Step 6

$T_{ecpe} (= \{t4\})$ is not empty then:

Let $t_k = t4$

Enrich_Marking_1($t4$):

Initially $L = \{\}$

Step EM.11:

The only one transition in conflict with $t4$ is $t3$. N is the input place of $t3$ that is not marked. Let's add a token $(e3, N)$ in L : $L = \{(e3, N)\}$.

Step EM.12:

We have introduced a token in place N . This marking enrichment is not consistent because we don't respect the p-invariant: $OFF + N + D = 1$ (there is a token in place D yet). We delete the inconsistent token $(e3, N)$ from L . Then, L becomes empty.

Step EM.13:

L_c is still equal to $\{(t2, D), (e2, Ac)\}$

L_e is still equal to $\{(e2, Ac)\}$

As $t3$ is not enabled, we don't store context.

Fire_Transition($t4$):

Step FT.1: $E = \{i, e2, t2, t4\}$

Step FT.2: $L_c = \{(t2, D)\}$. $A = \{(i, t2), (e2, t4)\}$

Step FT.3: $L_c = \{(t2, D), (t4, AF)\}$

Go to step 2

Step 2

All the transition sets are empty.

Step 3

The stop criterion is satisfied because all the transition sets are empty.

Step 9

The derived partial order is defined by: $E(1) = \{i, e2, t2, t4\}$, $A(1) = \{(i, t2), (t2, f), (e2, t4), (t4, f)\}$ and $Le(1) = \{(e2, Ac)\}$.

inc = 2

Go to step 1

Step 1

We start deriving the second partial order ($E2, A2$) from the first element of C stored above.

- $C = (Lc = \{(i, N), (e2, Ac)\}, Lft = \{t2\}, E = \{i, e2\}, A = \{(i, t2), (t2, f), (e2, t4), (t4, f)\}, Le = \{(e2, Ac)\})$.

C becomes empty.

- The enabled transitions for Lc are $t2, t3$ and $t4$.
- Transition $t2$ is deleted because it is in Lft
- Transition $t4$ is deleted because it is parallel to $t2$.

Therefore we obtain: $Tewc = \{t3\}$, $Tece = \{(e2, t3)\}$, $Tecpe = \{(t3, f)\}$, $Tpec = \{(t3, f)\}$.

Transition $t3$ is fired.

The derived partial order is defined by: $E(2) = \{i, e2, t3, f\}$, $A(2) = \{(i, t3), (e2, t3), (t3, f), (t3, f)\}$ and $Le(2) = \{(e2, Ac)\}$.

The algorithm is finished because the stop criterion is satisfied (C is empty (Step 1)).

VI. DISCUSSION AND CONCLUSION

We have presented and illustrated an algorithm for deriving scenarios in a Petri net. In the case of the example in the Fig.1, two scenarios have been derived. A feared one leading from marking $\{N, Ac\}$ to marking $\{D, AF\}$ by firing transitions $t2$ and $t4$ (first partial order in Fig.4) and a normal one leading from marking $\{N, Ac\}$ to $\{N, Ac\}$ by firing $t3$ (second partial order in Fig.5).

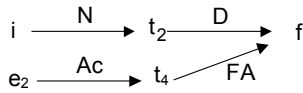


Fig. 4. First partial order ($E(1), A(1)$)

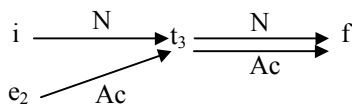


Fig. 5. Second partial order ($E(2), A(2)$)

Some observations may be derived:

- The initialization of the equipment denoted by transition $t1$ is not involved in the feared behavior. It does not belong, indeed, to any of the two derived scenarios. It is an illustration of the fact that our approach doesn't imply the enumeration of all the states.
- The algorithm execution has enriched the marking (a token in place Ac). This means that starting from a study of the equipment, we have derived that the feared scenario involves the actuator. In the first partial order, transition $t4$ is fired denoting an actuator failure.

The bifurcation between the scenario leading to the feared state and the nominal operation corresponds to the execution of the routine *Store_context(t2)*. This means that it is generated by the conflict between $t2$ and $t3$. A more detailed study shows that the conflict is between transition $t3$ and the pair of parallel transitions $t2$ and $t4$. To better understand what occurs, it is necessary to involve the continuous aspect of the system and particularly the thresholds attached to transitions $t2$ and $t3$ ($t4$ denotes a failure and its firing is not deterministic). A possible interpretation of the example is that transition $t2$ denotes a temperature overflow of the equipment, the actuator could be a ventilator and $t3$ denotes a cooling action. In this context, it is clear that the temperature threshold attached to $t3$ has to be lower than that attached to $t2$. Therefore, if $t3$ is enabled it is fired before $t2$. When it is not the case, it is necessarily because $t4$ has been fired before. This means that the actual scenario leading to the feared state consists in firing $t4$ before $t2$.

This intuitive reasoning can be done here because the example is simple. For more complex systems, it will be necessary to modify the algorithm to take into account some aspects of the continuous behavior. This work will be developed in next future.

REFERENCES

- [1] G. Moncelet, « Application des réseaux de Petri à l'évaluation de la sûreté de fonctionnement des systèmes mécatroniques du monde automobile », Thèse de Doctorat, N°3076, Université Paul Sabatier, Toulouse, France.
- [2] R. Champagnat, P. Esteban, H. Pingaud, R. Valette, « Modeling and simulation of a hybrid system through Pr/Tr PN DAE model », *ADPM'98 3rd, International Conference on Automation of Mixed Processes*, March 19-20, 1998, Reims, France, p. 131-137.
- [3] S. Khalfaoui, E. Guilhem, H. Demmou, R. Valette, « A method for deriving critical scenarios in mechatronic systems », $\lambda\mu 13$, European Conference on System Dependability and Safety, March 18-20, 2002, Lyon, France.
- [4] J.Y. Girard, « Linear Logic », *Theoretical Computer Science*, 50, 1987, p.1-102.
- [5] B. Pradin-Chézalviel, R. Valette, L.A. Künzle, « Scenario duration characterization of t-timed Petri nets using linear logic », *IEEE PNPM'99, 8th International Workshop on Petri Nets and Performance Models*, Zaragoza, Spain, September 6-10, 1999, p.208-217.