A Temporised Conflict-Free Routing Policy for AGVs

Dimitri Antakly, Jean-Jacques Loiseau, Rosa Abbou

A Temporised Conflict-Free Routing Policy for AGVs

Dimitri Antakly Jean Jacques Loiseau Rosa Abbou

Laboratoire des Sciences du Numérique de Nantes,
Université Bretagne Loire, Nantes, France
(e-mail: Dimitri.Antakly@hotmail.com;
{Jean-Jacques.Loiseau,Rosa.Abbou}@irccyn.ec-nantes.fr)

Abstract: This paper deals with the conflict avoidance problem of an Automated Guided Vehicles (AGV) system, in a Flexible Manufacturing System (FMS). Regarding the complexity of this kind of problems, it has generated many works to find an optimal strategy for scheduling and routing AGVs. In this paper, we propose a new strategy based on a temporal logic, modelled using time Petri nets. It consists in imposing suitable delays on an AGV in order to avoid a critical situation with another AGV. We built an algorithm including three main stages. In the first one, we test the evolution of our system. In the second stage, we calculate our suitable delay, if it exists, by testing the different critical scenario cases. In the last stage, we assign the delays and update the system. The validity of the algorithm is proven mathematically, and its feasibility is shown using a simulation.

Keywords: Automated Guided Vehicles (AGV), conflict avoidance, temporal logic, time Petri nets, discrete event simulation (DES).

1. INTRODUCTION

Increasing the degree of autonomy in a Flexible Manufacturing System (FMS) is a main objective in today's workshops. Doing so, requires the addition of automated systems, that can execute alone without any human interference. All the advantages of automating with industrial robots, such as decreased production costs, shorter cycle times, improved quality and reliability, increased safety and better floor space utilization, are countered with a big impasse: the increase in complexity. More one seeks autonomy in a system, the more control in its operation becomes complex. An AGVS (Automated Guided Vehicles System), is a set of a driver-less vehicles usually used to transport products between work stations in a workshop. All the AGVs of the AGVS, must be coordinated and work in a cooperative manner. The scheduling of each AGV's missions and the routing of its respective path, are accomplished by a computer-based control system. The objective of this paper is to establish robust conflict-free routes for each AGV, and to make sure the entire system is well synchronised and there is no possible time deadlocks. A deadlock is when the system is entirely blocked and requires human interference to be unblocked.

Many approaches have been generated in the literature dealing with this kind of problems. We can mention the heuristic approach solving hard problems but with *approximate results* (Noorul *et al.* (2003)); approaches based on simulations over UPPAAL and ARENA in order to get optimal routes using discrete event simulation (DES) (Maza and Castagna (2005); Arnaud *et al.* (2009)); approaches based on logical instincts or *Genetical Algorithms*(GA) consisting in keeping only the *fittest* solutions after generating a number of possible solutions (Subbaiah *et al.* (2009)). Another work based on *Coloured Timed Petri Nets* (Dotoli and Fanti (2004)) has a similar approach as the one presented in this paper, and we will be discussing its positionning regarding this paper later on, in the conclusion section. We can also quote most recent timed automata approach developped by (Girault *et al.* (2016)), in which the authors propose a new *compositional* method to calculate on the fly the new supervisor, each time the system evolves in a certain direction. In this work, the proposed approach combines in a way the advantageous parts of each cited approach, as explained in the following.

We proposed a policy that ensures conflict avoidance between the AGVs of the system, based on a temporal logic but handles both the logic and the time aspects of the system. The policy consists in adding extra delays, if needed, on the execution time of an AGV on a working station, in order for it to avoid a possible accidental situation, as our system evolves. What makes this strategy different than the Ramadge and Wonham theory of supervision (Ramadge and Wonham (1987)), is the fact that the added *temporised* aspect allows us to handle the *unwanted* states or situations instead of throwing them away. In order to solve the problem and proceed with the strategy, the system was modelled using TPN (Time Petri nets) and an algorithm was built, detailing the strategy step by step.

This article is organised as follows. The second section is dedicated to the description of the problem. In the third section, we introduce the algorithm proposed and we prove it mathematically along with a simulation. Finally, section four is dedicated for the conclusion and future works.

2. PROBLEM STATEMENT AND OVERVIEW

In this section, we will introduce the time Petri nets, as well as the different hypotheses used to define the area of our work.

2.1 Time Petri Nets

A time Petri net is defined as a tuple $(P, T, Pre(., P), Post(P, .), \alpha, \beta, M_0)$ (Berthomieu and Diaz (1991), where:

- $P = \{P_1, P_2, ..., P_m\}$, is a finite non empty set of places (or states),
- $T = \{T_1, T_2, ..., T_n\}$, is a finite non empty set of transitions $(T \cap P = \emptyset)$,
- $Pre(., P) \in (\mathbb{N}^P)^T$ is the function defining the places at the start of a transition,
- $Post(P, .) \in (\mathbb{N}^P)^T$ is the function defining the places at the end of a transition,
- $M_0 \in \mathbb{N}^P$ is the initial marking of the net,
- $\alpha \in (\mathbb{Q}^+)^T$ and $\beta \in (\mathbb{Q}^+ \cup \{\infty\})^T$ are the functions giving for each transition, respectively, its earliest and latest firing moments $(\alpha \leq \beta)$.

A time Petri net (TPN) has two conditions for transition occurrence, the logical and the timed condition. The logical condition of a state (or configuration) is a marking $M_i(t)$ defining the number of tokens in the place P_i at time t. The vector $M(t)$ defines the number of tokens in all the places at time t. The logical condition of a transition is enabled if the tokens required by the preconditions are present in the marking. The timed condition for transition occurrence is defined by the interval $[\alpha, \beta]$, a transition is enabled if its clock value lies in that interval. An example of a TPN is shown in Fig. 1 (modelled using ROMEO (Lime *et al.* (2009))). To explain more the logical condition, we can see in Fig. 1 (representing a small parallel manufacturing process) that the transition T1 is enabled when there's a token in place P1. As for transition T2, it's only enabled when there are tokens in places P2 and P3.

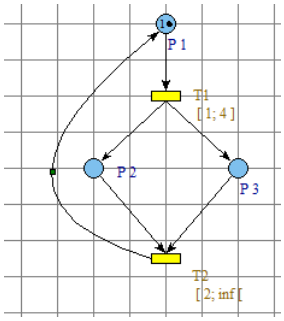

Fig. 1. Example of a TPN

We cite two types of arcs, that are useful for the work to follow:

- The *logical inhibitor arc* which its role is to inhibit the transition occurrence, as long as the place, at the first end of the arc, has tokens in.
- The *reset arc* which role is to reset the place, at the first end of the arc, when the corresponding transition, at the other end of the arc, is fired.

2.2 Problem Description

The problem we are handling in this paper is a very common problem, which has generated many works dealing with it. This work was inspired by the real AGV platform of the IUT-University of Nantes (*Institut Universitaire de Technologie de Nantes, FRANCE*). In our case study, the paths of the circuit are unidirectional and the capacity of the resources (or the working stations), as well as the paths, are unitary which means that only one AGV at a time is allowed on a path or on a node (the working stations will be named nodes for simplification reasons). We also add that an AGV cannot take any order while it is on its way (on a path) between two working stations, it is only *controllable* on the nodes (it is blind on the paths). This leads us to induce the accidental situations, where two AGVs can collide either on a path or on a node. The most important thing to define here in our discrete events system is the *temporised* aspect. The travel time on each path and the execution time on each node are already known and they are inevitable. For example, the rest period of an AGV on a node P_1 lies necessarily in the interval $[T_1; T_1 + \Delta_1]$, where Δ_1 is a delay imposed on the node, $\Delta_1 \geq 0$.

Example. As an illustration of the problem described before, a presentation of the real workplace of the IUT-University of Nantes, is shown in Fig. 2. In this figure, the coloured points are not only the working stations but also the *action points*, on which the AGVs can take orders such as "go straight", "take a turn", "load", etc.

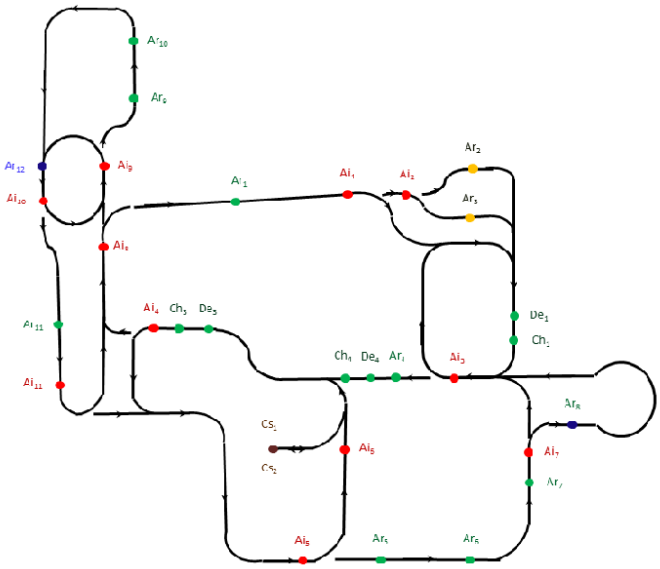

Fig. 2. Schematic of the circuit at the IUT-University of Nantes

To execute any order, the AGV needs a response time (or execution time). The execution times on the different *nodes* are different and may be brief, but they are inevitable. The fact that leads us to be, in this example, in the context of the hypotheses we described before.

3. PROPOSED METHODOLOGY

Now that we have defined the problem, we seek to manage the possible conflicts on the resources (modelled by the places in the TPN). The algorithm we propose resolves this problem for any size or shape of the circuit. The algorithm leads to a valid solution, if it exists. It handles the conflict avoidance for two AGVs, and can be certainly developed to handle N AGVs since the conflicts occur between two AGVs at a time. Hence, the problem is always reduced to two AGVs at the end ($N = 2$), but its complexity will increase at high rate as we increase the number of AGVs; the order of complexity will be discussed later, in section three. The objective is to calculate, on-line (after the start-up of the AGVs) and during a predefined run time, the effective dates of liberation of each node after adding the delays, if needed. We note i and j the AGV numbers. We associate for the AGVs the delays Δ_i and Δ_j which will be calculated. The function $ChoixDe\Delta(\Delta_i, \Delta_j)$ used in the algorithm is implemented in a way to choose one Δ between the two possible values, impose it on the corresponding node and resetting the other to zero. This way we delay one AGV not both. The choice of a Δ strongly depends on imposed specifications, for example:

- Passing priority of an AGV on the other, considering energetic or capacity constraints, or simply to respect a certain ratio of production between different cycles;
- Total delays imposed on the node;
- Total delays imposed on the AGV;
- Minimising the waiting time.

We note that if we find $\Delta < 0$, we assign the value zero to it, knowing that our system is causal and the execution times are inevitable (we can not accelerate the procedures on the nodes). We now introduce the notation used to formulate the algorithm:

- T_i: execution time on node i.
- Θ_{ij}: travel duration between the nodes i and j.
- $U_i(t)$: command allowing the liberation of a node i (or working station).
- z_i: the date of a job completion on node i.
- Δ_i: the delay imposed on node i.
- k_i: the date of liberation of node i.
- Timer: global clock.
- t: date of arrival on the last node occupied by AGV1.
- t': date of arrival on the last node occupied by AGV2.
- D: total run time.
- m: security margin, imposed by the specification of the system, between the date of liberation of a node and the date of a new occupation.
- $InitialisationDuCircuit()$: function that initialises the circuit, the travel durations and the execution times.
- $InitialisationDesGammes(V_1, V_2)$: function that initialises the sequences of each AGV ($k = 1$ or 2).
- $PositionsInitiales(V_1, V_2)$: function determining the initial position of each AGV.
- $M_i(V_k)$: marking function determining if the node i is occupied by the AGV k and that by examining the sequences initialised by $InitialisationDesGammes(V_1, V_2)$, in order to detect if an AGV exists its cycle and in this case be prepared for a deadlock.
- $Succ(M_i, V_k)$: function determining the successor of the AGV k exiting the node i, and that also is done regarding the sequences (it returns the number of the succeeding node).
- $Position(V_k)$: function returning the actual position of the AGV k.

Algorithm 1 Calculation of the delays

$InitialisationDuCircuit()$;
$InitialisationDesGammes(V_1, V_2)$;
$PositionsInitiales(V_1, V_2)$;
while $Timer \leq D$ **do**
  if $(M_i(V_1)) == 1 || M_j(V_2) == 1)$ **then**
    $z_i = t + T_i$;
    $z_j = t' + T_j$;
    $\Delta_i = 0$;
    $\Delta_j = 0$;
    if $(M_i(V_1)) == 1 || M_j(V_2) == 0)$ **then**
      $S = Succ(M_j, V_2)$;
      $Position(V_2) = Succ(M_j, V_2)$;
      $z_S = (k_j + \Theta_{jS}) + T_S$;
      $z_j = z_S$;
    end if
    if $(M_i(V_1)) == 0 || M_j(V_2) == 1)$ **then**
      $S = Succ(M_i, V_1)$;
      $Position(V_1) = Succ(M_i, V_1)$;
      $z_S = (k_i + \Theta_{iS}) + T_S$;
      $z_i = z_S$;
    end if
    if $(Succ(M_i, V_1) == Position(V_2)$
    $\&\& Succ(M_j, V_2) == Position(V_1))$ **then**
      $\Delta_i = z_j - (z_i + \Theta_{ij}) + m$;
      $\Delta_j = z_i - (z_j + \Theta_{ij}) + m$;
      $ChoixDe\Delta(\Delta_i, \Delta_j)$;
    else if $Succ(M_i, V_1) == Succ(M_j, V_2)$ **then**
      $l = Succ(M_i, V_1) = Succ(M_j, V_2)$;
      $\Delta_i = (z_j + \Theta_{jl}) + T_l - (z_i + \Theta_{il}) + m$;
      $\Delta_j = (z_i + \Theta_{il}) + T_l - (z_j + \Theta_{jl}) + m$;
      $ChoixDe\Delta(\Delta_i, \Delta_j)$;
    else if $Succ(M_i, V_1) == Position(V_2)$ **then**
      $\Delta_i = z_j - (z_i + \Theta_{ij}) + m$;
      if $\Delta_i < 0$ **then**
        $\Delta_i = 0$;
      end if
    else if $Succ(M_j, V_2) == Position(V_1)$ **then**
      $\Delta_j = z_i - (z_j + \Theta_{ji}) + m$;
      if $\Delta_j < 0$ **then**
        $\Delta_j = 0$;
      end if
    end if
    $k_i = z_i + \Delta_i$;
    $k_j = z_j + \Delta_j$;
    $U_i(k_i) = 1$;
    $U_j(k_j) = 1$;
  end if
end while

We first comment this Algorithm 1 and give some explanation before formally proving its validity. In the first part, we start by initialising the circuit, the execution times, the sequence of each AGV, the travel durations, and the initial position of each AGV which should be a node. During a specified run time, we observe the evolution of the system

(the 1^{st} **if** test). In other words, each time an AGV occupy a new node, we anticipate the presence of the other AGV (supposedly on the road, in the worst case) on its next node by calculating the date of job completion on the node (2^{nd} and 3^{rd} **if** tests). The use of this *anticipation* will be elaborated and clearer as we go on. We note that the first time we enter the big "While" loop the values of k do not exist, but anyway we will not be risking to use them since the two AGVs are on nodes and we will go straight to the following tests. The first test corresponds to the most critical case where the AGVs are permuting places, that is why we calculate two values of Δ: one corresponding to the waiting time of the AGV on the node i and the other corresponding to the waiting time of the AGV on the node j. This waiting time is the difference between the date of job completion on the succeeding node (adding the security marge m to it) and the arriving date of the AGV occupying the first node on the succeeding one.

After calculating the two Δ, the function $ChoixDe\Delta(\Delta_i, \Delta_j)$ (already explained) will choose the fittest one and reset the other to zero. It is the same for the last two tests, except that in these cases we know already on which node we should impose the delay. Finally, the second test corresponds to the case where the two AGVs have the same next destination. Thus, for this case to calculate a Δ corresponding to a given node, we should do the difference between on one hand, the execution time on the destination node named T_l, adding to it the arrival date of the AGV occupying the other node (and the marge m), and on the other hand the necessary time for the AGV occupying the current node to arrive to the destination.

Each time the system evolve there is a single true test or none, and due to that we will obtain a suitable value of Δ for a determined node. The last stage (or the iteration) consists in defining the dates k in where the commands of liberation of the working stations are not equal to zero. So we have our set of commands $U_c(t)$.

Proof.
We demonstrate mathematically the validity of the proposed algorithm, using a reduction to the absurd. We prove the absence of conflict proving that, for the system supervised and delayed by the values calculated with Algorithm 1, any conflict leads to a contradiction.

First remark that the possible conflicts can be sorted into two families: conflicts on the places (or nodes) and conflicts on the trajectories (or transitions). A conflict on a place means that there is a possibility that two AGVs collide while trying to occupy the same working station, and a conflict on a transition means that there is a possibility that an AGV blocks the path of another AGV. Hence, these scenarios are the only "accidental" ones. We use the notation, "places", "arcs" and "transitions" in the following, since in the simulation that follows the modelling is done using a TPN.

We successively exam these two categories. We should prove the absence of conflicts (or accidents), considering both the logical and the temporal states of the system.

On the places: We consider the nodes P_l, P_i et P_j such as, the following relations are verified: $Post(., P_i) \cap Pre(P_l, .) \neq \varnothing$ and $Post(., P_j) \cap Pre(P_l, .) \neq \varnothing$, or in other words, there exists one (or many) route(s) leading from P_i and P_j towards P_l. We suppose that $M_{P_l}(t) > 1$ is verified, then there is two possible cases:

- If $M_{P_l}(t_0) = 1$ (with t_0 the initial moment and $t_0 < t$) thus, $M_{P_i}(t_0) = 1$ and a transition t_{il} is firable such as: $z_i + \Theta_{il} + \Delta_i \leq z_l$ (with $\Delta_i \geq 0$).

- If $M_{P_l}(t_0) = 0$ thus, $M_{P_i}(t_0) = 1$ and $M_{P_j}(t_0) = 1$ and two transitions t_{il} and t_{jl} are firable such as: $(z_i + \Theta_{il} + \Delta_i \leq z_j + \Theta_{jl} + T_l)$ or $(z_j + \Theta_{jl} + \Delta_j \leq z_i + \Theta_{il} + T_l)$ (with $\Delta_i \geq 0$ and $\Delta_j \geq 0$).

In each case, we verify by reduction to the absurd, that the initial hypothesis ($M_{P_l}(t) > 1$) is not possible.
- In the first case, we have $\Delta_i = z_l - (z_i + \Theta_{il}) + m$ (with $m > 0$), or $\Delta_i = 0$ (according to the algorithm). By replacing the two possible values of Δ_i in the inequality, we obtain: $z_l - z_i - \Theta_{il} + m \leq z_l - z_i - \Theta_{il}$ giving us $m \leq 0$ which is absurd; with the second value, if $\Delta_i = 0$ we obtain: $0 \leq z_l - z_i - \Theta_{il} \Rightarrow z_l \geq z_i + \Theta_{il}$ which defies the initial proposition.
- In the second case, we have $\Delta_i = (z_j + \Theta_{jl}) + T_l - (z_i + \Theta_{il}) + m$ (and $\Delta_j = (z_i + \Theta_{il}) + T_l - (z_j + \Theta_{jl}) + m$) (according to the algorithm and with $m > 0$). By replacing these values in the inequality, we obtain: $m \leq 0$, which is absurd.

On the transitions: There are two possibilities to have an accidental situation:
- The same transition is fired two times during an interval smaller to the travel time, for example t_{ij} is firable at t and at $[t; t + \Theta_{ij}[$.
- Two transitions t_{il} and t_{jl}, both leading to P_l, are fired during an interval smaller to the travel time Θ_{il} (or Θ_{jl}). For example, t_{il} is firable in $[t; t + \Theta_{jl}[$ (with t_{jl} fired at t), or t_{jl} is firable in $[t; t + \Theta_{il}[$ (with t_{il} fired at t).

We verify that both possibilities are absurd.

- The first proposition is written mathematically using the delay times: $z_i + \Delta_i < \Theta_{ij}$, but according to the algorithm in this case we have $\bar{z_i} = z_i + \Theta_{ij} + T_j$, with $\bar{z_i}$ representing the new value of z_i calculated in the algorithm using the intermediate value z_S. Having $\Theta_{ij} + T_j > 0$ thus we have $z_i > \Theta_{ij}$ ($\Delta_i > 0$), then to say that $z_i + \Delta_i < \Theta_{ij}$ is absurd.
- The second proposition is written mathematically using the delay times: $z_i + \Delta_i < \Theta_{jl}$, but according to the algorithm in this case we have $\Delta_i + z_i = (z_j + \Theta_{jl} + T_l + m) - \Theta_{il}$. Leading to two possible cases, the first where $z_j + T_l + m > \Theta_{il}$, giving $z_i + \Delta_i > \Theta_{jl}$, contradicting the second proposition; the second where $\bar{z_j} + T_l + m < \Theta_{il}$ (with $\bar{z_j} = z_j + \Theta_{jl} + T_l$) and thus representing a non accidental situation because $\Theta_{il} > z_j + \Theta_{jl} + T_l$, then the path leading from i to l is of a bigger duration than the sum of the liberation time on j, the travel time from j to l and execution on l. In this case the AGVs will not be in an accidental situation. All of this is absurd, which ends the proof. ■

We took into consideration the logical and the temporal states (having all the information on the durations and the arrival dates of the AGVs in their current positions) in our proof, so we can say that the proposed algorithm gives robust solutions if the solutions actually exist (and

since we did not take any "due dates" into account, so the solutions will always exist). We add that a *Deadlock* is impossible in this algorithm, since that always we have one AGV waiting for the other to proceed and never two AGVs waiting at the same time.

4. SIMULATION AND VERIFICATION

4.1 Simulation using ROMEO

We already illustrated a schematic of the actual circuit that inspired this work (Fig. 2), and given that the circuit is very large to run a simulation on (and also redundant), we managed to resume all the critical scenarios that can happen and represent them in a simpler circuit (Fig. 3), we also note that the *simpler* circuit also represents all the critical scenarios that are possible between two AGVs in any circuit. We consider that the sequences (cyclic) of the AGVs are: $AGV_1 : (P_1, P_2, P_4, P_5)^*$ and $AGV_2 : (P_2, P_3, P_4, P_5)^*$.

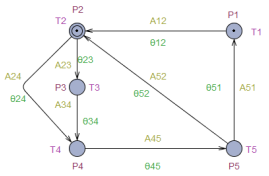

Fig. 3. Simulation Circuit

We model our problem using a TPN (Fig. 4, on the next page due to its size), the software Romeo allows us to model, simulate and verify our DES (Lime *et al.* (2009)).

The execution times on the working stations are shown in the Fig. 4, as well as the travel time between nodes. There are two horizontal sequences representing the two cycles of each AGV. These two are connected by places *P2Occ*, *P4Occ* and *P5Occ*, representing the occupation (or the vacancy) of the working stations P_2, P_4 and P_5, forming the only conflict zones for the considered sequences (because they are the only nodes visited by the two AGVs, the others are only visited by either AGV_1 or AGV_2). These places are responsible of inhibiting the commands "U" from the preceding nodes towards the nodes 2, 3 and 5 (for example the command of the nodes 1 and 5 for the node 2); and thus inhibiting every transition towards the node if it is occupied (by using the logical inhibitor arc). The execution times on the places 2, 4 and 5 is divided into two parts (the places "debut"), when the token (corresponding to the AGV) passes to the second part, the places *P2Occ*, *P4Occ* and *P5Occ* are emptied (using the Reset arc or Flush). This division into two parts takes into consideration the travel time from the preceding node to the current node and the security margin (m).

The transitions D represent the Δ or the waiting time on the place after the execution has finished. Simulating this net step by step we managed to regroup without any conflict, the necessary delays to be added. For example Fig. 5, 6 and 7 show respectively the calculation of Δ_1, Δ_3 and Δ_2 corresponding to AGV_1 that is why it is named $D21$ (the two AGVs can use this place). We obtain $\Delta_1 = 2$, $\Delta_3 = 0$, $\Delta_2 = 4$.

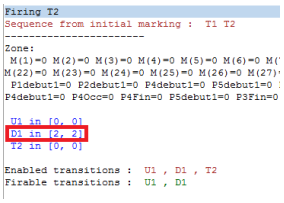

Fig. 5. Calculation of Δ_1 for the first cycle of AGV1

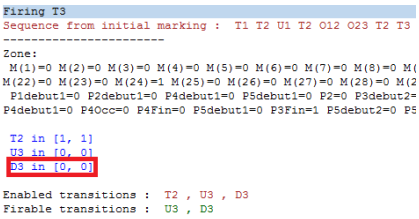

Fig. 6. Calculation of Δ_3 for the first cycle of AGV2

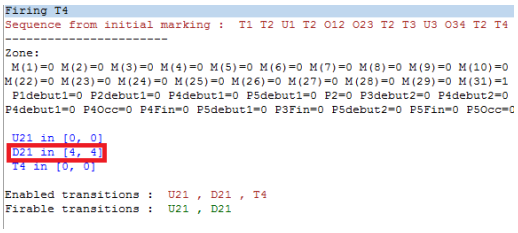

Fig. 7. Calculation of Δ_2 for the first cycle of AGV1

With the help of the model-checker of ROMEO, we can verify that the model is conflict free using the CTL logic. For example in Fig. 8, we verify the absence of conflicts on the places "P2début1" and "P2début2", by verifying that the proposition "does it exists globally (which means any path), between 0 and infinity, a situation where the marking of the place is superior to 1"; the model-checker replies that this proposition is false and a marking like that does not exist (the indexes of the places tested are respectively 2 and 18, the index on ROMEO depends on the order of placement of the places).

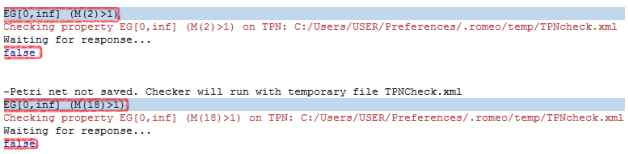

Fig. 8. Verification of the absence of conflicts on node P2

So, with no hazards occurrences, we can prove (this time by simulation) that the previous algorithm works.

4.2 Complexity Study

We analyse the complexity of the proposed algorithm, and study the possibility of extending it to a higher number of AGVs (three or more).

The proposed algorithm is of the order of $O(n^2)$, with n the number of times the tests are launched or in other

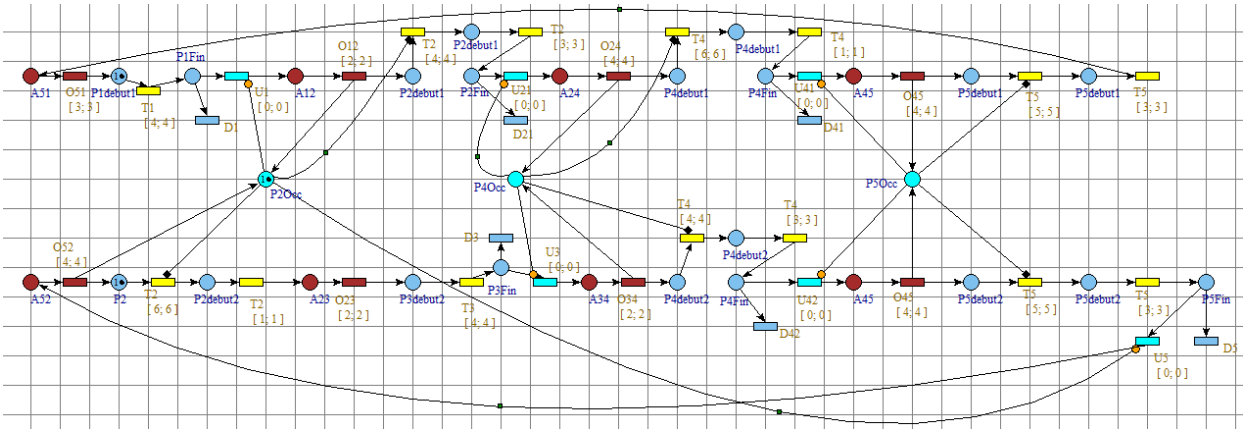

Fig. 4. TPN model on ROMEO tool

words the system evolves ($n = max$*(total number of places occupied by AGV1; total number of places occupied by AGV2)*). Because each time a new node is occupied we launch a series of tests requiring the computation of a two dimensional array (one dimension for each AGV), containing the sequences and the delay times; in order to update them in case we add delays. The computation time estimated for this algorithm will be between $10ms$ and $1s$ for n between 1000 and 10000.

Getting to three AGVs, we already risk having the tests launched two times on parallel threads at the same time which is likened to a matrix multiplication (of cubic complexity $O(n^3)$). The computation time is estimated for this algorithm to be between $10s$ and $2.7h$ for a n between 1000 and 10000.

We conclude that already passing to three AGVs increases the risk of an exploding computational time.

5. CONCLUSION

In this paper, we proposed a new policy to build conflict free routes for two AGVs working in any circuit, independently of its size or configuration, as opposed to previous works based on the Petri nets approach (Dotoli and Fanti (2004)). In the latter, the authors divided the circuit into zones, so the efficiency highly depends on the size of the circuit and making sure that the "zones" are optimal. Plus, we don't need to decide online weither or not to accept a new route (or mission) given to the AGV, since in our approach any of the two AGVs can accept any mission at any time guaranteeing a *conflict and deadlock-free* routing. Although this study is based on a real circuit on the platform of the IUT-University of Nantes, this policy could be applied to any real time system with delays, and in particular could be applied in traffic control. Its advantages are that it allows us to simulate step by step the evolution of our system, detecting the dynamics and the delays that are needed to be added each time our system evolves, in order to conserve the *conflict-free* objective.

For future works, we propose to test the performance of the algorithm by applying it to different real time systems with bidirectional trajectories or more than two resources (in this case the resources were the AGVs), since the *conflict-free* problem can always be reduced to two AGVs at a time. We can find a set of solutions for each pair of AGVs in the system (using our strategy) and intersect the sets of solutions in order to get a valid solution for all pairs, but we risk the absence of solution.

REFERENCES

Arnaud Y., Cury, J., Loiseau, J.J., and Martinez, C. (2009). Using UPAAL for the secure and optimal control of AGV fleet. In Proc. 7th Workshop on Advanced Control and Diagnosis ACD 2009, Pologne, 19-20 novembre 2009.

Berthomieu, B., Diaz, M., Modelling and Verification of Time Petri Dependent Systems Using Time Petri Net. *IEEE Trans. Software Engineering*, vol. 17, no. 3, 259–273, 1991.

Dotoli, M., Fanti, M.P., Coloured timed Petri net model for real-time control of automated guided vehicle systems. *Int. Journal of Production Research*, vol. 42, no. 9, 1787–1814, 2004.

Girault, J., Loiseau, J.J., and Roux, O.H. (2016). On-line compositional controller synthesis for AGV. *Discrete Event Dynamic Systems*, 26, 583–610.

Lime, D., Roux, O.H., Scidner, C., Traoumouez, L.M. (2009). Romeo: A parametric model-checker for Petri nets with stopwatches. *In Int. Conference on Tools and Algorithms for the Construction and Analysis of Systems*, vol. 5505, 54–57.

Maza, S., and Castagna P. (2005). Performance-based structural policy for conflict-free routing of bi-directional automated guided vehicles. *Computers in Industry*, vol. 56, no. 7, 719–733.

Noorul, A., Karthikeyan, T. , and Dinesh, M. (2003). Scheduling decisions in FMS using a heuristic approach. *Int. Journal of Advanced Manufacturing Technology*, vol. 22, no. 5-6, 74–379.

Ramadge, P.J., and Wonham, W.M. (1987). Supervisory control of a class of discrete event processes. *SIAM journal on control and optimization*, vol. 25, no. 1, 206–230.

Subbaiah, K.V., Nageswara Rao, M., and Narayana Rao K. (2009). Scheduling of AGVs and machines in FMS with makespan criteria using sheep flock heredity algorithm. *Int. Journal of Physical Sciences*, vol. 4, no. 3, 139-148.