Self-Stabilizing Scheduling Algorithm for Cooperating Robots

Joyce El Haddad

Lamsade, Université Paris Dauphine

75775 Paris Cedex 16, France

elhaddad@lamsade.dauphine.fr

Serge Haddad

Lamsade, Université Paris Dauphine

75775 Paris Cedex 16, France

haddad@lamsade.dauphine.fr

Abstract

We address the problem of autonomous robots which alternate between execution of individual tasks and peer-to-peer communication. Each robot keeps in its permanent memory a set of locations where it can meet some of the other robots. The proposed self-stabilizing algorithm solves the management of visits to these locations ensuring that after the stabilizing phase, every visit to a location will lead to a communication. We model the untimed behaviour of a robot by a Petri net and the timed behaviour by an (infinite) Discrete Time Markov Chain. Theoretical results in this area are then combined in order to establish the proof of the algorithm.

Keywords

Self-stabilizing systems, randomized distributed algorithms, autonomous robots, Petri net theory, Markov Chain.

INTRODUCTION

In the field of multi-robot cooperation, it is useful to make a distinction between two main issues : the first issue involves the achievement of some global tasks, the second issue involves the simultaneous operations of several autonomous robots, each one seeking to achieve its own task. Our contribution is focused on the first issue. More precisely, we study cooperation between robots that requires communication between them in order to perform global applications. For instance, it would be much better to deploy several robots for the maintenance of nuclear reactors where one robot leads a way to locate the fault component and another follows to replace it with a new one. In such an environment, there are two kinds of protocols to design : a synchronization protocol between neighboring robots in order to establish (temporary) point to point communications and a routing protocol in order to exchange packets between two robots (and in particular distant ones). In this work, we will focus on the synchronization problem between robots as a base for routing.

The robot we consider are *homogeneous* (they all follow the same set of rules), *autonomous* and *anonymous* (they are a priori indistinguishable). Each robot is constraint to move in a limited area and to visit its locations in such a way that every location is infinitely often visited. The obvious requirement is that a robot cannot leave a location without establishing a communication with the other robot associated with this location. When two or more autonomous robots must interact, communication between them is essential. However, in our system communication take place indirectly without help of a global communication mean. Therefore, each robot communicates via the other robots, also participating to the network, to transmit its collected data or to exchange messages. Thus, communications have limited life-time : time needed to forward the packets from one robot to the other. In such environnement, many communication protocols [5, 7, 8] have been designed but they suppose reliable robots.

Although there has been significant work related to cooperating robots, very few researchers have expanded their approaches to incorporate fault tolerance. Our research focus is to introduce self-stabilization as an efficient property that makes the system tolerant to faults and takes into account the limited resources of the robots in term of processor, memory and energy. Roughly speaking, a self-stabilizing protocol is designed to recover from an unsafe state caused by a fault to a safe state by itself. The study of self-stabilization started with the fundamental paper of Dijkstra [2]. Following the pioneering work of Dijkstra a great amount of works has been done in this area [3, 10]. However, with the presence of mobility and dynamic changes, these traditional communication protocols meant for self-stabilizing networks are no more appropriate.

This paper describes the design of a uniform self-stabilizing scheduling protocol upon which a self-stabilizing communication protocol could be derived using standard self-stabilizing algorithms. This protocol solves the management of visits to the locations ensuring that after the stabilizing phase, every visit to a location will lead to a communication.

The rest of the paper is organized as follows. In section 2, we briefly describe the original algorithm and we model it with a Petri net giving a new proof of its correctness and showing how it can be generalized. In section 3, we give a detailed description of its transformation into a self-stabilizing protocol. Its correctness is proved in section 4. We conclude in section 5.

A NON SELF-STABILIZING SCHEDULING PROTOCOL

The current work is based on Bracka et al. [1] scheduling protocol for a robotic network. Let us first describe the hypotheses :

- There is a set of anonymous robots (i.e., identities are not used in the protocol). We will denote it by: $\{r_1, \ldots, r_m\}$.

- There is a set of locations, each one with a unique numeric identifier. We will denote this set: $\{l_1, \ldots, l_N\}$ in increasing order. A pair of robots is associated to each location that can go to this location and establish a temporary communication if they are both present.

- Any robot r_i has in its permanent memory an array of the locations where it can go. This array is sorted in increasing order of the identifiers. n_i will denote its size and $f(i, j)$ for $1 \leq i \leq m$ and $0 \leq j \leq n_i - 1$, will denote the identifier of the jth location of the robot r_i.

- Between any pair of robots r_i and $r_{i'}$, there is a sequence of robots $r_i = r_{i_0}, r_{i_1}, \ldots, r_{i_K} = r_{i'}$ such that for all $0 \leq k < K$, r_{i_k} and $r_{i_{k+1}}$ share a location. This hypothesis ensures that there is a (potential) global connectivity between robots.

The goal of the algorithm is to schedule the visit of the locations for each robot in such a way that every location is infinitely often visited. The obvious requirement is that a robot cannot leave a location without establishing a communication with the other robot associated with this location (we will call its partner, a peer). The proposed scheduling is for each robot to infinitely visit its locations following the order of its array.

In [1], the authors develop a specific (and rather lengthy) proof that no (partial or global) deadlock can occur. With the help of Petri net theory, we give a short and simple proof of the algorithm. In fact, this new proof will be the basis of the self-stabilizing version of this protocol. We assume a basic knowledge of Petri nets syntax and semantics; otherwise, a good introduction to this topic can be found in [9].

We model the behaviour of each robot by a local Petri net. Then the whole protocol is modelized by the union of these nets where transitions with the same identity are merged. Figure 1 represents the local Petri net associated with the robot r_i. We denote it by $N_i = (P_i, T_i, Pre_i, Post_i, M0_i)$ where :

- $P_i = \{p_{(i,0)}, \ldots, p_{(i,j)}, \ldots, p_{(i,n_i-1)}\}$ is the set of places. When $p_{(i,j)}$ is marked, r_i is going to its jth location, or waiting there for the other robot.

- $T_i = \{t_{f(i,0)}, \ldots, t_{f(i,j)}, \ldots, t_{f(i,n_i-1)}\}$ is the set of transitions. When $t_{f(i,j)}$ is fired, the communication has happened at the jth location and the robot goes to its next location.
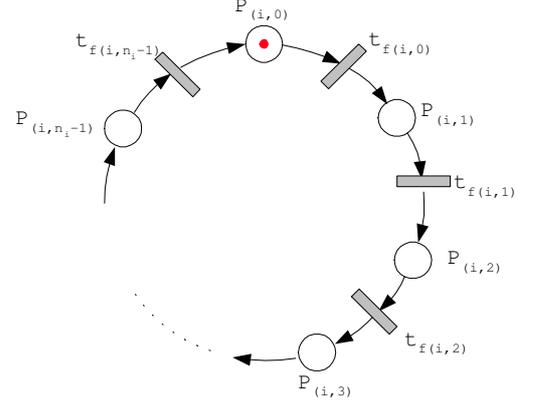
Figure 1: **The cycle of visits for robot** r_i

- Pre_i is the precondition matrix, $Pre_i : P_i \times T_i \to \{0, 1\}$ defined, according to the behaviour, i.e.,

$$Pre_i(p, t) = \begin{cases} 1 & if\ p = p_{(i,j)}\ and\ t = t_{f(i,j)} \\ & for\ some\ j \\ 0 & otherwise \end{cases}$$

- $Post_i$ is the postcondition matrix, $Post_i : P_i \times T_i \to \{0, 1\}$ defined, according to the behaviour, i.e.,

$$Post_i(p, t) = \begin{cases} 1 & if\ p = p_{(i,(j+1)\ modulo\ n_i)}\ and \\ & t = t_{f(i,j)}\ for\ some\ j \\ 0 & otherwise \end{cases}$$

- $M0_i$ is the initial marking defined, according to the behaviour, i.e.,

$$M0_i(p_{(i,j)}) = \begin{cases} 1 & if\ j = 0 \\ 0 & otherwise \end{cases}$$

Then the scheduling protocol is modelled by the global Petri net $N = (P, T, Pre, Post, M0)$ where:

- $P = \biguplus P_i$, is the disjoint union of places of local Petri nets.

- $T = \bigcup T_i$, is the (non disjoint) union of transitions of local Petri nets.

- $Pre, Post$ are the Precondition and Postcondition matrices, defined from $P \times T$ over $\{0, 1\}$, by:

$$Pre(p, t) = \begin{cases} Pre_i(p, t) & if\ p \in P_i\ and\ t \in T_i \\ & for\ some\ i \\ 0 & otherwise \end{cases}$$

$$Post(p, t) = \begin{cases} Post_i(p, t) & if\ p \in P_i\ and\ t \in T_i \\ & for\ some\ i \\ 0 & otherwise \end{cases}$$

- $M0$ the initial matrix is defined by $M0(p) = M0_i(p)$ if $p \in P_i$.
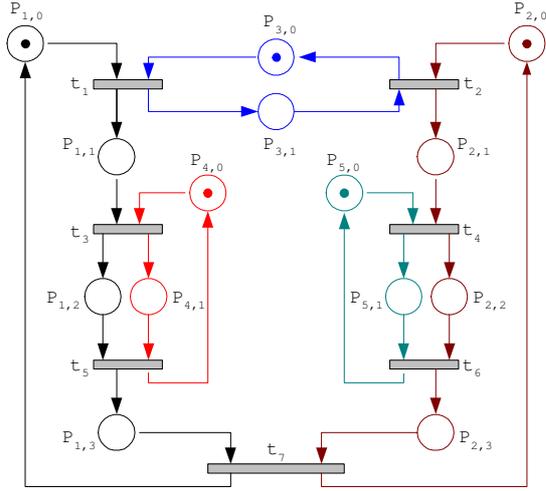
Figure 2: **A global Petri net model for an instance of the protocol**

For instance, consider a system consisting of five robots with seven locations. The following table represents the array of locations for each robot.

Robots	r_1	r_2	r_3	r_4	r_5
	1	2	1	3	4
Locations	3	4	2	5	6
	5	6			
	7	7			

Then the corresponding global Petri net of the above system is shown in Figure 2.

By construction, the global net N belongs to a particular subclass of Petri nets called *event graphs* defined by the restriction that each place has exactly one input transition and one output transition. In Petri nets, the absence of (partial) deadlock is called liveness and its definition states that whatever a reachable marking and a transition, there is a firing sequence starting from this marking and ended by this transition. In other words, whatever the state of the net, every transition is potentially fireable in the future of this state. Many behavioural properties of nets are structurally characterized in the case of the event graphs. However the following lemma will be sufficient for our purposes. We recall the proof since the associated constructions will be used in the proof of self-stabilization.

Lemma 1 *Let N be an event graph such that every cycle has an initially marked place, then N is live.*

Proof : Given a cycle in an event graph, the only transitions that produce or consume tokens of the places of the cycle are the transitions of the cycle. Thus, the number of tokens of every cycle remains constant.

Let us suppose that every cycle is initially marked. The previous remark shows that in every reachable marking, every cycle is marked.

Now we fix a reachable marking M and we define a binary relation $helps_M$ between transitions. $t\ helps_M\ t'$ if and only if there exists a place p with $M(p) = 0$ which is an output of t and a input of t'. Let us denote $precedes_M$ the transitive closure of $helps_M$.

We claim that $precedes_M$ is a partial order. Let us suppose that this is not the case. Then we have two transitions t and t' such that $t\ precedes_M\ t'$ and $t'\ precedes_M\ t$. From the definition of $precedes_M$, it means that there is a path from t to t' and a path from t' to t where every place is unmarked for M. Concatenating them, we obtain an unmarked cycle, which is impossible. Every partial order on a finite set can be extended in at least one total order. Let $t_1, \ldots, t_n$ the ordered list of the transitions (by this order).

We claim that $t_1 \cdot \ldots \cdot t_n$ is a firing sequence starting from M. Indeed, t_1 is fireable since all its input places are marked (for M). Now by induction, let us suppose that $t_1 \cdot \ldots \cdot t_i$ is a firing sequence for M leading to M'. Then all input places of t_{i+1} are marked for M' since such a place was already marked for M, or has been marked by the firing of some t_j with $j \leq i$ (recall that in event graphs, a transition does not share its input places). So the firing sequence can be extended to t_{i+1}. Thus the net is live. $\square$

Now we can easily establish the correctness of the protocol.

Proposition 2 *Let N be a net modelling the protocol for some network of robots. Then N is live.*

Proof : Consider a cycle of N and let t_k be the transition with smallest identifier occurring in this cycle. Let $p_{(i,j)}$ be the input place of t_k inside the cycle, and let $t_{k'}$ be the input transition of $p_{(i,j)}$ inside the cycle. By construction of N, $k = f(i,j)$ and $k' = f(i, (j-1) \bmod n_i)$. The choice of t_k implies that $k < k'$, but f is increasing w.r.t. its second argument. Thus the only possibility for j is 0. As $p_{(i,0)}$ is initially marked, we have proved that every cycle has an initially marked place and we conclude with the help of the previous lemma. $\square$

This result can be straightforwardly generalized to the case of n-ary rendez-vous between robots. However, the networks we study are useful due to their flexibility. Introducing n-ary rendez-vous with $n > 2$ decreases such flexibility. So for sake of simplicity, we will restrict ourselves to the initial case of binary synchronization.

A SELF-STABILIZING SCHEDULING PROTOCOL

In this section, we present a randomized self-stabilizing scheduling protocol adapted from the previous algorithm. At first, we make some additional assumptions.

- Each robot has a timer that wakes up the robot on expiration. In the rest of the paper, we suppose that the

timers are exact. In section 5, we will discuss about this hypothesis. For the robot r_i, this timer is denoted $timeout_i$. The range of the timer is the real interval $[0 \ldots N + 1]$.

- A travel between two locations takes at most 1 tu (time unit). This hypothesis can always be fulfilled by an appropriate choice of the time unit.

- Each robot has a sensor giving it its current position. For the robot r_i, this sensor is denoted $position_i$. This sensor takes its value in the set $\{0, \ldots, n_i - 1\} \cup nowhere$, indicating either the local index of the location where r_i is waiting, or in the case of $nowhere$, indicating that r_i is between two locations.

- $MP_i[0 \ldots n_i - 1]$ denotes the array of locations sorted by increasing order, present in a permanent memory of the robot.

The behaviour of the robot is event-driven: the occurrence of an event triggers the execution of a code depending also on its current state. In our case, there are two events: the timer expiration and the detection of another robot. We denote such an event a peer detection with the obvious meaning that the two robots are both present at some location. We do not consider that the arrival to a location is an event; instead when a robot reaches a location, it just stops. As a robot refills its timer to 1 tu before going to a new location, the timer will expire after the end of the trip, and then the robot will execute the actions corresponding to the arrival. The crucial point here is that, with this mechanism, *the duration of a trip between two locations becomes exactly* 1 tu. A variable $status_i$, that takes as value either $moving$ or $waiting$, has a special role on the behaviour of the robot w.r.t. the events handling. When this variable is set to $moving$, the robot can neither detect another robot, nor can it be detected by another one. Looking at the program of figure 3, it means that even if a robot arrives at a destination where its peer is already waiting, the communication between them will happen *only after the timer of the arriving robot expires*.

As shown in the program, a robot has four actions: $SYNC$, $WAIT$, $RECOVER$ and $MISS$. $SYNC$ and $WAIT$ correspond to the actions of the original algorithm. In order to recognize that a timer expiration corresponds to an arrival, we use the variable $status_i$. It is set to $moving$ when the robot goes to a new location, and set to $waiting$ when the timer of a robot arriving at a location expires. However, $WAIT$ is different from the corresponding action of the previous algorithm as the robot sets its timer to $N + 1$ (recall that N is the number of the locations). When the timer of a robot arriving at a location expires and a peer is already waiting then it will firstly execute its $WAIT$ action, and as its status is becoming $waiting$ both will execute their $SYNC$ action.

When recovering from a crash, the timer of a robot triggers an action. The action $RECOVER$ is executed by a

Constant
 N, n_i;
 $MP_i[0 \ldots n_i - 1]$;
Timer
 $timeout_i \in [0 \ldots N + 1]$;
Sensor
 $position_i \in \{0, \ldots, n_i - 1\} \cup \{nowhere\}$;
Variables
 $status_i \in \{waiting, moving\}$;
 $choice_i \in \{0, 1\}$;

ON PEER DETECTION // $SYNC$
// on r_i arrival or on peer arrival while the other is already waiting
// Necessarily $status_i$ is waiting
 Exchange messages;
 Refill($timeout_i$,1);
 $status_i = moving$;
 Go to $MP_i[(position_i + 1) \bmod n_i]$;

ON TIMER EXPIRATION
 If $(position_i\, ! = nowhere)$ And $(status_i == moving)$ **Then**
 // $WAIT$
 // r_i arrives at the location
  Refill($timeout_i$,$N + 1$);
  $status_i = waiting$;
 Endif

 If $(position_i == nowhere)$ **Then**
 // $RECOVER$
 // recovery from a crash while the robot were between
 // two locations
  Refill($timeout_i$,1);
  $status_i = moving$;
  Go to $MP_i[0]$;
 Endif

 If $(position_i\, ! = nowhere)$ And $(status_i == waiting)$ **Then**
 // $MISS$
 // expiration of the timer while r_i is waiting for a peer
  Uniform-Choice($choice_i$);
  Case $(choice_i)$
   0 : Refill($timeout_i$,1);
   1 : Refill($timeout_i$,1);
    $status_i = moving$;
    Go to $MP_i[0]$;
  Endcase
 Endif

Figure 3: **Protocol for robot** r_i

robot at most once in our protocol (depending on the initial state), and necessarily as the first action of the robot. It happens if the robot is between two locations after the crash. Then the robot goes to its first location.

The key action for the stabilization is $MISS$. It happens either initially, or when the robot is waiting for a peer at a location and its timer has expired. Then the robot makes a (uniform) random choice between two behaviours :

- it waits again for 1 tu;

- it goes to its first location.

When it is called, the random function Uniform-Choice sets its single parameter to a value among $\{0, 1\}$.

An execution of this algorithm can be seen as an infinite timed sequence $\{t_n, A_n\}_{n \in \mathbb{N}}$, where $\{t_n\}$ is a strictly increasing sequence of times going to infinity and each A_n is the non-empty set of actions that have been triggered at time t_n (at most two actions per robot in the case when it executes $WAIT$ and immediately after $SYNC$). With this formalization, we can state what is a stabilizing execution.

Definition 3 *An execution $\{t_n, A_n\}_{n \in \mathbb{N}}$ of the protocol is stabilizing if the number of occurrences of $RECOVER$ and $MISS$ is finite.*

In other words, after a finite time, the protocol behaves like the original algorithm. Let us remind that $RECOVER$ can occur at most once per robot. The next section will be devoted to show the following proposition.

Proposition 4 *Given any initial state, the probability that an execution will stabilize is 1.*

PROOF OF STABILIZATION

Without loss of generality, we consider that the initial state is a state obtained after each robot has executed at least one action. Thus we do not have to take into account the action $RECOVER$. With this hypothesis and for a better understanding of the protocol, a state graph of a robot is presented in figure 4.

Probabilistic Semantics of the Protocol

We assume that the code execution is instantaneous: indeed, it is neglictible w.r.t. the travels of the robots. Thus in our protocol, the single source of indeterminism is the random choice of the $MISS$ action since all trips take exactly 1 tu. Consequently, the probabilistic semantics of our protocol is a Markov chain whose description is given below.

A state of this Markov chain is composed by the specification of a state for each robot. The state of a robot r_i is defined by its vector $< s_i, l_i, to_i, \alpha_i >$, where :

- s_i : the robot's status that takes value in the set $\{waiting, moving\}$ depending on whether the robot is waiting at a location or moving to it,
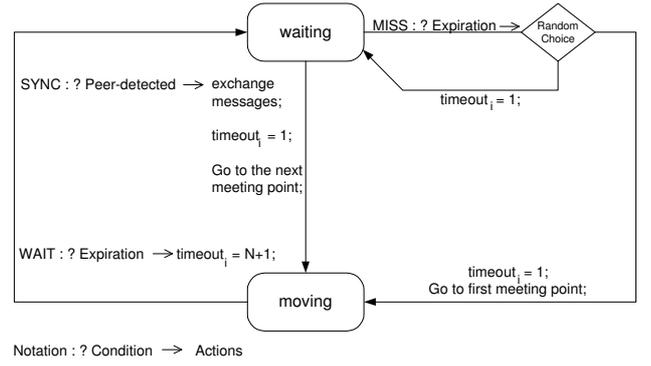
Figure 4: **A state graph for r_i**

- l_i : the location where the robot is waiting or moving to,

- to_i is given by the formula $\lceil timeout_i - 1 \rceil$ where $\lceil x \rceil$ denotes the least integer greater than, or or equal to x. to_i takes its value in $\{0, \ldots, N\}$,

- α_i is given by the formula $\alpha_i = timeout_i - to_i$. It takes its value in $]0 \ldots 1]$. We will call it the residual value.

The last attributes deserve some attention. As we consider states after the execution of the actions, the variables $timeout_i$ are never null: this explains the range of these attributes. Moreover, these attributes are simply a decomposition of $timeout_i$. However the interest of this decomposition will become clear in the next paragraph. So, a state e will be defined by: $e = \prod_{i=1}^{m} < s_i, l_i, to_i, \alpha_i >$.

Let us note that the set of states is infinite and even uncountable since the α_i's take their values in an interval of $\mathbb{R}$. However, we show that we can lump this chain into a finite Markov chain with the help of an equivalence relation that fulfills the conditions of strong lumpability [6].

Definition 5 *Two states $e^1 = \prod_{i=1}^{m} < s_i^1, l_i^1, to_i^1, \alpha_i^1 >$ and $e^2 = \prod_{i=1}^{m} < s_i^2, l_i^2, to_i^2, \alpha_i^2 >$ are equivalent if :*

1. $\forall\, i, s_i^1 = s_i^2, l_i^1 = l_i^2, to_i^1 = to_i^2$,

2. $\forall\, i, j,\ \alpha_i^1 < \alpha_j^1 \iff \alpha_i^2 < \alpha_j^2$

An equivalence class (denoted by c) of this relation is characterized by: $c = \prod_{i=1}^{m} < s_i, l_i, to_i > \times\ position$, where $position$ represents the relative positions of the α_i's. It is easy to show that there are at most $m! \cdot 2^{m-1}$ distinct positions. Thus, the number of equivalence classes is finite. The next proposition establishes the conditions of strong lumpability.

Proposition 6 *Let c and c' two equivalence classes, let e^1 and e^2 be two states of the class c, then :*

$$\sum_{e \in c'} P[e^1, e] = \sum_{e \in c'} P[e^2, e]$$

where P denotes the transition matrix of the Markov chain.

Proof : Consider any two equivalent states $e^1 = \prod_{i=1}^{m} < s_i, l_i, to_i, \alpha_i^1 >$ and $e^2 = \prod_{i=1}^{m} < s_i, l_i, to_i, \alpha_i^2 >$. Let I be the subset of indices i such that α_i^1 is minimal among the residual values; we denote this value α_{min}^1. Let J be the subset of indices j such that α_j^2 is minimal among the residual values α_j^2; we denote this value α_{min}^2. Since e^1 and e^2 are equivalent, $I = J$, and $\forall\, k,\ \alpha_{min}^1 \le \alpha_k^1$ and $\alpha_{min}^2 \le \alpha_k^2$. We denote $to_{min} = Min(to_i \,|\, i \in I)$. Let us now elapse $\alpha_{min}^1\ tu$ from e^1 to lead to f^1 and $\alpha_{min}^2\ tu$ from e^2 to lead f^2. f^1 and f^2 are just intermediate states since no action has happened (see above the formalization of an execution). We now study two cases :

1. $to_{min} > 0$. The state of r_i for $i \in I$ becomes $< s_i, l_i, to_i - 1, 1 >$ in both f^1 and f^2. The state of another robot remains unchanged, except for its residual value that has decreased by $\alpha_{min}^1\ tu$ in f^1, and by $\alpha_{min}^2\ tu$ in f^2. Thus, the new relative positions of the residual values are identical in f^1 and in f^2. Therefore, the intermediate states f^1 and f^2 are equivalent. So, we apply again the same operation until the second situation will happen (and it will happen since, during each iteration, at least one to_i is decreased and none is increased).

2. $to_{min} = 0$. Let us denote $I' = \{i \in I \,|\, to_i = 0\}$. Then the set of robots r_i for $i \in I'$ is exactly the set of robots for which their timer expire in both f^1 and f^2. Thus, their states are identical in f^1 and f^2. They will execute either the $WAIT$, or the $MISS$ action. The set of these new states reached from f^1 (resp., f^2) is obtained as the randomized effect of the $MISS$ actions. If k robots execute their $MISS$ action, there will be 2^k such states associated with f^1 (resp., f^2) each one with the probability $1/2^k$. We call g^1 and g^2 such new intermediate states where the Uniform-choice has given the same result in g^1 and g^2. Then, some r_i's for $i \in I'$ after the $WAIT$ action may execute a $SYNC$ action with some peer. This peer is in the same state in g^1 and g^2, except possibly for the value of its timer. But the condition for the $SYNC$ action is independent of the value timer. So, the $SYNC$ actions will happen, both in g^1 and g^2, leading to the new states h^1 and h^2 that are each one of the 2^k successors of e^1 and e^2, respectively, in the Markov chain. It remains to prove that h^1 and h^2 are equivalent. Since the same actions with the same effect have been executed, h^1 and h^2 may only differ in the timer value. Let us examine the different cases. A robot r_i that has executed a single action $WAIT$ has its timer set to $N + 1$ in both h^1 and h^2 ($\alpha_i = 1$). A robot that has executed at least a $SYNC$, or a $MISS$ action has its timer set to 1 in both h^1 and h^2 ($\alpha_i = 1$). A robot that has not executed an action has its timer

decreased by α_{min}^1 in h^1 and α_{min}^2 in h^2. Consequently, the new relative positions of residual values are the same in h^1 and h^2. $\qquad\square$

A Markov chain can be viewed as a graph where there is an edge between one state s and another s' iff there is a non null probability to go from the former to the latter (i.e., $P[s, s'] \neq 0$). The edge is labelled by this probability. The following lemma (only valid for finite chains) will make the proof of correctness easier.

Lemma 7 (See [4]) *Let S' be a subset of states of a finite Markov chain. Let us suppose that for any state s, there is a path from s to some $s' \in S'$. Then whatever the initial state, the probability to reach (some state of) S' is 1.*

Stable States

In this subsection, we exhibit a condition on states that ensures that, in an execution starting from a state fulfilling such a condition, the $MISS$ action will never occur. We need some preliminary definitions based on the Petri net modelling of the original protocol.

Definition 8 *Let $e = \prod_{i=1}^{m} < s_i, l_i, to_i, \alpha_i >$ be a state of the system. Then the marking M(e) of the net N modelling the protocol is defined by: $M(e)(p_{i,j}) = 1$ If $l_i = f(i, j)$ Else 0.*

In fact, the marking $M(e)$ is an abstraction of the state e where the timed informations and the status of the robot are forgotten.

Definition 9 *Let N be a net modelling the protocol and M be a marking of N, then M is said to be deadlock-free if for the marking M, all the cycles of N are marked.*

In a state modelled by a deadlock-free marking, if we execute the original protocol, then no deadlock will never happen. However, due to the values of the timer, it may happen that for a state e with $M(e)$ being deadlock-free, a $MISS$ action happens (for instance, on timer expiration of a waiting robot while its peer is still moving). Thus we must add timed constraints to the state e.

If M is deadlock-free, then the relation $helps_M$ introduced in lemma 1 defines a directed acyclic graph (DAG) between transitions. We define $level_M(t)$ as the length of the longest path of this DAG ending in t. Here the length of a path is the number of vertices of this path. In figure 5, we have represented the level of transitions for the initial marking of the net of figure 2. We are now ready to define our condition on states.

Definition 10 *Let $e = \prod_{i=1}^{m} < s_i, l_i, to_i, \alpha_i >$ be a state of the system. Then e is stable if M(e) is deadlock-free and, $\forall i,\ s_i = waiting \Rightarrow to_i \ge level_{M(e)}(t_{l_i})$.*
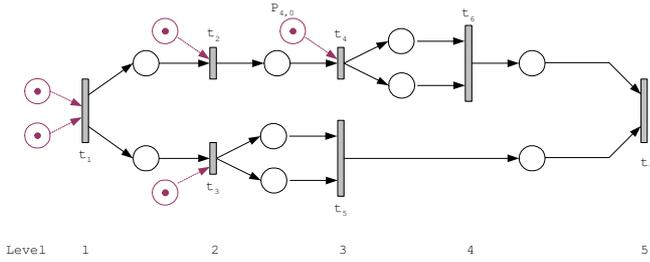
Figure 5: **The level of transitions of a deadlock-free marking**

The next lemma shows that the definition of stable states is appropriate.

Lemma 11 *In an execution starting from a stable state, the action $MISS$ will never happen.*

Proof : We will proceed by induction on the states of the system at times $0, 1, 2, \ldots$. We note e^n the state of the system at time n. e^0 is the initial stable state. Be aware that these states do not correspond to the successive states of the Markov chain, but it does not matter since we will not use here any probabilistic argument. Our induction hypothesis is that until time n no $MISS$ action has happened, and e^n is stable. For $n = 0$, it is just the hypothesis of the lemma. Let us examine what happens between time n and $n + 1$.

Let us look at a robot waiting at a location at time n. Since its timer is greater than 1 (by the stability hypothesis and the fact that $\alpha_i > 0$), it will not expire until $n + 1$. Let us now look at a robot moving to a location at time n. It will arrive during the interval $[n \ldots n + 1]$ and will refill its timer to $N + 1$. In both cases, either a $SYNC$ will happen and the robot will be moving at time $n + 1$, or it will still be waiting. Thus we have proved that no $MISS$ action happens during this interval.

It remains to show that e^{n+1} is a stable state. Since no $MISS$ action happens during the interval, the execution (without taking into account the timer values) corresponds to the execution of the non self-stabilizing protocol. Thus $M(e^{n+1})$ is a marking reached by a firing sequence from $M(e^n)$, and so all the cycles are marked in this new marking.

Let t be a transition of level 1 for $M(e^n)$. t has its two places marked, meaning that the two associated robots are either waiting at the corresponding location, or moving to it. Thus the synchronization will happen before time $n + 1$.

Let t be a transition of level > 1 for $M(e^n)$. t has one of its places unmarked, that means that one of the robots associated with the corresponding location is neither waiting at this location, nor it is moving to. Thus a synchronization at the location is impossible during this interval. So t will not be fired during the interval.

Suppose now that a robot r_i is waiting at time $n + 1$ at a location. If this robot has arrived during the interval, it has

set its timer to $N + 1$, and thus at time $n + 1$, to_i is still equal to N, which is an upper bound for the level.

Finally, suppose that this robot has been waiting during the whole interval. Then its timer (and so to_i) is decreased by one at time $n+1$, but the level of the corresponding transition was greater than one at time n and has not been fired. All the transitions of level 1 have been fired, so its level at time $n+1$ is also decreased by 1 (since the paths to this transition in the new DAG are exactly the paths to it in the old DAG truncated by their origin). Thus the timed constraints are still verified and e^{n+1} is a stable state. $\qquad\square$

From an Initial State to a Stable State

In this section, we show that given any initial state, there is a path from this state to a stable state in the Markov chain. Thus the proposition 4 will follow almost directly from lemmas 7 and 11. The single non trivial observation to make is that given two equivalent states s and s' (see definition 5), then s is stable iff s' is stable since the stability does not involve the residual times. Thus the path found below gives a path in the finite aggregated Markov chain where the final state is a set of stable states.

Lemma 12 *Given any initial state, there is a path in the Markov chain from this state to a stable state.*

Proof : As we look for a path in the Markov chain, each time the $MISS$ action happens, we can choose its random output. So, when in what follows we will choose, during a part of execution, the first choice (staying at the location), we will say that we simulate the original algorithm.

If the initial state is stable, then we are done. So we suppose that the initial state e is not stable. We examine the two following cases :

1. $M(e)$ is deadlock-free.

   Here the timed constraints of stability are not verified by e. By simulating the original algorithm, we claim that a stable state will be reached. First, all successive markings associated with the states will be deadlock-free since they are reachable from $M(e)$ in the net N.

   Second, we decompose time into intervals of 1 tu. During each interval, all the locations corresponding to the transitions of level 1 will be the support of $SYNC$ actions. A robot that will execute such a $SYNC$ action will have its timed constraint fulfilled since it is moving. Moreover, using exactly the same proof as the one of lemma 11, it can be shown that when a timed constraint is fulfilled, it will always be fulfilled. Thus after each robot has executed at least one $SYNC$ action, we have reached a stable state.

2. $M(e)$ is not deadlock-free.

   Since there is a chain of synchronization locations between any pair of robots, applying the original algorithm would lead us to a global deadlock. Thus we

simulate the original algorithm until every robot is blocked alone at a location, and then has executed at least once its $MISS$ action. This means that all timers have their values ≤ 1.

Now we choose for every robot the second alternative of the $MISS$ action. All these actions happen in less than 1 tu. So after the last $MISS$ action has been executed, every robot is still moving to its first location. In this state denoted by e', $M(e')$ is the initial marking of the net modelling the original protocol. Thus $M(e')$ is deadlock-free and we complete the current path by the path of the first case. $\qquad\square$

CONCLUSION

We have designed a uniform self-stabilizing scheduling protocol for a network of robots. The interest of this work is twofold. First, self-stabilization is an important and desirable feature of protocols for these environments. Second, the use of formal models for proofs of stabilizing algorithms is not so frequent. Here, with the help Petri nets theory, we have simplified the proof of the non stabilizing version of the algorithm. A part of the proof of stabilization is also based on this model.

The hypothesis that the timers are exact is only important during the stabilization step. Once the algorithm reaches a stable state, we can show that the protocol still works if the timers are prone to small deviations. Moreover in practice, if the stabilization step is not too long, then the deviations of the timers will not disturb it.

The next stage of our research is to explore how should new robots be incorporate in the system without bringing it down and how to compute shortest paths in order to implements a routing protocol.

REFERENCES

[1] P. Bracka, S. Midonnet, and G. Roussel. Scheduling and routing in an ad hoc network of robots. *IASTED International Conference on Computer Science and Technology*, Cancun, Mexico, May 2003.

[2] E. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11):643–644, 1974.

[3] S. Dolev. *Self-stabilization*. MIT Press, 2000.

[4] W. Feller. *An introduction to probability theory and its applications. Volume I*. John Wiley & Sons, 1968. (third edition).

[5] H. Hu, I. Kelly, D. Keating, and D. Vinagre. Coodination of multiple mobile robots via communication. *Proceedings of SPIE. Mobile Robots XIII and Intelligent Transportation Systems*, pages 94–103, Boston, Massachusetts, Novembre 1998.

[6] J.G. Kemeny and J.L. Snell. *Finite Markov Chains*. Van Nostrand, Princeton, NJ, 1960.

[7] V. Park and M. Corson. A highly adaptive distributed routing algorithm for mobile wireless networks. *Proceedings IEEE INFOCOM, The Conference on Computer Communications, Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies*, 3:1405–1413, Japan, April 1997.

[8] G. Prencipe. Corda: Distributed coordination of a set of autonomous mobile robots. *European Research Seminar on Advances in Distributed Systems, Ersads*, Italy, May 2001.

[9] W. Reisig. *Petri Nets: an Introduction*. Springer Verlag, 1985.

[10] M. Schneider. Self-stabilization. *ACM Symposium Computing Surveys*, 25:45–67, 1993.