

# Fault Tolerant Distributed algorithms for Mobile Agents \*

Mohamed Amine Haddar  
ReDCAD-Research unit  
Higher Institute of business  
management of Sfax, Tunisia  
haddar@labri.fr

Mohamed Mosbah  
LaBRI UMR 5800  
ENSEIRB - Bordeaux 1  
University, 351 cours de la  
Libration 33400 Talence, France  
mohamed@labri.fr

Ahmed Hadj Kacem  
ReDCAD-Research unit  
Faculty of Economics and  
Management of Sfax, Tunisia  
ahmed.hadjkacem@fsegs.rnu.tn

Mohamed Jmaiel  
ReDCAD-Research unit  
National Engineering School of  
Sfax, Tunisia  
mohamed.jmaiel@enis.rnu.tn

Yves Métivier  
LaBRI UMR 5800  
ENSEIRB - Bordeaux 1  
University, 351 cours de la  
Libration 33400 Talence, France  
metivier@labri.fr

## ABSTRACT

In this paper we propose a fault tolerance technique for mobile agent distributed algorithms. This technique allows mobile agents to overcome transient faults of machines and network links. We formally present the fault tolerance technique using the model notation proposed for designing mobile agent distributed algorithms. The technique is then formally presented by a set of transitions. Given a mobile agent transition system (respecting our model notation), we have proposed a methodology for integrating fault tolerance behavior to this transition system. To illustrate both the technique and the methodology we present a mobile agent transition system to compute a spanning tree. Respecting the proposed methodology we extend this transition system. The resulting mobile agent transition system tolerate transient faults of the underlying distributed system while computing the spanning tree.

## Keywords

distributed algorithms, mobile agent, fault tolerance.

## 1. INTRODUCTION

Mobile agents are programs that can move through a network under their own control and interact with resources and local environments. This technology, a profound revolution in computer software technology, is a brand new computing technique promoted for solving a large scale of problems in computer science [27]. Among many others, the distributed computing community is presenting an increasing interest into mobile agents due to their considerable reduction of network load and their overcoming of the network latency [17]. There are also new trends to use this technology in mobile distributed systems for which mobile agents can intuitively give promoting solutions for arising problems [1].

We already proposed a generic model to design and to prove mobile agent based distributed algorithms [13, 12].

\*This work has been supported by the French research agency ANR number ANR-06-SETI-015.

This model uses the local computation view, by transitions, to describe formally mobile agent algorithms. To illustrate its expression power and its proof simplicity, we proposed solutions to the spanning tree problem [12] and the election problem [10] considering several network topologies.

As mobile agent-driven applications are facing critical issues, reliability is an import property. In fact, today's distributed systems are dynamic, heterogenous and mobile. They are threatened by several kinds of failures (crash, stop-fail, malicious host, etc).

In traditional distributed systems computational models, fault tolerance policies have taken a "host-central" view. This view is no longer valid when dealing with mobile agent based distributed systems. New fault tolerant policies have been established with a "mobile agent" view. Many works are then proposed using different approaches aiming to secure mobile agents from several kinds of faults. These approaches are not appropriate to the local computation model. In [15], a fault tolerant model is proposed in the *message passing system* which is based on adding correcting rules to the initial rewriting system to tolerate faults.

Since mobile agents are prone to failure on hosting machines or communication links, some works have been proposed for specific failures adequate algorithms such as black holes [6, 7, 5, 16]. Other works have proposed fault tolerant strategies to various classes of faults. A taxonomy of fault-tolerance (key words and formal models) can be found in [9].

Overcoming host and link crashes is a challenging issue since developers have no control over the entire distributed system. A first group of researches have used the checkpointing to solve this problem. The mobile agent information is saved to a stable storage upon arrival to a host. This allow mobile agents to resume from crash failures by using saved information. Many techniques and strategies have been proposed to define the frequency and the manner to make checkpoints [31, 3, 19]. A second group of authors have adopted the replication strategy. Mobile agent computation is divided into stages. After each stage, the mobile agent is replicated and sent to several sites. The mobile

agent can then overcome site failures (crash, fail-stop, etc) [23, 22, 21, 20, 30]. A third group of authors used other alternatives. In [24], W.Qu et al. proposed a mixed approach using both replication and checkpointing. In [28], M.Tosic proposed a generic fault tolerant layer for mobile agents execution. The proposed layer uses the web-services technology and the publish/subscribe messaging model to propose fault tolerant services to mobile agents. More works are available in a recent survey of mobile agent fault-tolerant strategies [25] and a survey of rollback recovery protocols in message passing systems [8].

Regardless the large variety of proposed fault tolerant approaches, we can notice several significant limits :

- A first limit is the wide use of the site (host) dependent approach which needs a setup of specific platforms on each site. This approach seems unrealistic while considering large distributed systems like Internet.
- A second limit is noticed when using replication. Mobile agent replicas are sent to predefined sites. This breaks mobile agents autonomy and compel them following predefined itinerary. This seems in contradiction with mobile agent paradigm.
- Another limit is that considered mobile agent applications uses only client/server or information retrieval schemas.
- To our knowledge, few works allow a mobile agent to choose autonomously its itinerary. [30] allows mobile agent to walk autonomously but prohibits it modifying the visited sites state.

Our work is motivated by the necessity of designing and proving distributed algorithms to be used in faulty systems. We present in this paper an extension of the model we proposed allowing the design of fault tolerant mobile agent algorithms. As a case study, we have proposed a fault tolerant technique using the extended model formal notation. This technique consists of a set of transitions. In the design of this technique, we looked for the simplicity and genericity. In fact, the designed technique can be used by any mobile agent distributed algorithm designed considering fault-free environment to tolerate transient failures of the distributed system entities.

In our model, we try to compel the existing works limits by using a mobile agent dependent approach. This approach is adequate for large scale distributed systems and does not need a setup of a specific platform on every site. In addition, our model allows using all the properties defined in the mobile agent paradigm such as moving autonomously over the distributed system. When it visits a site, a mobile agent is also allowed to modify the site local information. Our contribution is threefold :

- we have extended our model to propose a generic fault tolerant model for use in the local computation view. The user, after considering a class of faults, can propose adequate solution to a given problem. In addition, our fault tolerance model does not restrict mobile agent behavior. We have emphasis on agent properties (such as autonomy, cooperation, mobility etc) in the design of our fault tolerant model.

- We have proposed a fault tolerant technique consisting of a set of transitions which can be added to any mobile agent transition system acquiring it a fault tolerant behavior to overcome transient faults of the distributed system entities.
- We have proposed a methodology to extend any mobile agent transition system by adding the fault tolerance transitions. The resulting transition system acquires a fault tolerance behavior. We have presented, as a case study, an illustration for the spanning tree problem.

This paper is organized as follow: we begin by briefly presenting our model formal notation. As an illustration of the model, we expose a solution to the spanning tree problem in fault-free distributed systems. Next, we present our fault-tolerant technique to deal with transient faults. To get more benefits from this technique, we expose a methodology to add it to any mobile agent distributed algorithm formally described using our model formal notation. To better explain the proposed methodology we use it to add a fault tolerance behavior to the already proposed solution for the spanning tree problem. The last section gives a conclusion and some future works.

## 2. THE MOBILE AGENT COMPUTATIONAL MODEL

In order to formally describe a mobile agent algorithm, we define what we call a *Mobile Agent System* representing a set of mobile agents moving on a distributed system. We model the distributed system by a labeled graph. A mobile agent distributed algorithm is then formally represented by a set of transitions. Each mobile agent transition can change the current place labeling (or state) and the mobile agent state. A transition can also formalize appropriate actions of the mobile agent paradigm (such as cloning, moving, dying, waiting, etc).

### 2.1 Mobile agent system

A mobile agent system, as defined in [4], consists of:

- a collection  $\mathbb{P}$  of places,
- a navigation subsystem  $\mathbb{S}$ ,
- a collection  $\mathbb{A}$  of mobile agents,
- an injection  $\pi_0 : \mathbb{A} \rightarrow \mathbb{P}$  describing the initial placement of mobile agents,
- an initial labeling  $\lambda$  of places and mobile agents.

The labeling  $\lambda$  of places and mobile agents can code anonymous places (all places have the same label), anonymous mobile agents (all mobile agents have the same label) or any initial knowledge of the mobile agent. As examples of initial knowledge, we can cite the number of places, the number of mobile agents, the diameter and the topology of the navigation subsystem, the placement topology of mobile agents and identities or partial identities of places or mobile agents.

The navigation subsystem is described by an undirect connected graph  $G = (V, E)$ , where the vertices  $V$  denote the places and the edges  $E$  denote the bidirectional channels operating between them. Each place  $u$  contains a set of ports. Every port represents an access point to a communicating

channel. Let  $\delta_u$  be a function of ports classification which assigns to each port representing a channel connecting  $u$  to a place  $v$  a unique integer  $\delta_u(v)$  belonging to  $[1, deg(u)]$ . Each mobile agent which migrates from one place to another knows by which channel it leaves by choosing the adequate port number. Thus, the navigation subsystem is defined by  $\mathbb{S} = (G, \delta)$ .

Every place in the mobile agent system is able to host mobile agents by offering basic primitives to make them running. It also contains a *WhiteBoard* which is used to store the place state (label). It is accessible in a mutual exclusion way for reading and writing operations done by mobile agents. The *lock/unlock* operations are envisaged to manage the access to the *Whiteboard* and thus to prohibit the concurrent access (reading or/and writing) to the place state.

Mobile agents communicate only by exchanging information saved in local whiteboards. A mobile agent can use two execution modes. A standard mode (the *Active* mode) in which it can do several tasks. And a "waiting" mode (the *Standby* mode) in which it sleeps on a place waiting for a wake up event (a token allowing it to return to the *Active* mode) (further explanation can be found in the next section). To every mobile agent, we associate a timer by default on the mode OFF. A mobile agent can start its clone timer or its own. For security consideration, a mobile agent is not allowed to start another mobile agent timer. When a mobile agent timer is started (resp. finished), the mobile agent switches automatically to the *Standby* (resp. *Active*) mode.

The mobile agent system is asynchronous: there is no access to global clock. The migration is asynchronous: a migrating mobile agent arrives on a place in a bounded but unknown time. A communicating channel is modeled by a FIFO (first in first out) queue: two mobile agents sent on the same channel must arrive in the same order in which they are sent. Every mobile agent arriving on a place waits its turn, in order to be executed, in the *Able-To-Run* FIFO queue. The computations done by a mobile agent in a place is an *atomic task*. This property is guaranteed by the *lock/unlock* operations systematically done by the mobile agent when it starts/finishes its task on every place.

## 2.2 Mobile agent algorithm

Given a mobile agent system ( $\mathcal{MaS}$ ), we propose a formal notation of the mobile agent algorithm. In  $\mathcal{MaS}$ , a mobile agent performs several actions such as moving across a channel, waiting on a place, etc. These actions can be modeled in  $\mathcal{MaS}$  using transitions. Thus a mobile agent algorithm is described using a transition system. We associate to each mobile agent  $\mathbf{a}$  a transition system  $\mathbb{T}_a$ .

A transition  $T$  describes a mobile agent  $\mathbf{a}$  elementary action on a place  $\mathbf{p}$ . This transition may affect only the mobile agent  $\mathbf{a}$  and the place  $\mathbf{p}$  settings. A transition must allow the mobile agent  $\mathbf{a}$  to create a clone. Thus a transition must describe also the clone settings. A transition has roughly the following form:

$$(\mathcal{M}_A, \mathcal{P}) \Rightarrow (\{\mathcal{M}'_A, \mathcal{M}_A^c\}, \mathcal{P}')$$

Such that

$\mathcal{M}_A$  (resp.  $\mathcal{M}'_A$ ): a multi-set corresponds to the mobile agent settings before (resp. after) the transition.

$\mathcal{M}_A^c$ : a multi-set corresponds to the clone settings (=  $\emptyset$  if there is no clone).

$\mathcal{P}$  (resp.  $\mathcal{P}'$ ): corresponds to the place  $\mathbf{p}$  settings before (resp. after) the transition.

A mobile agent can wait for predefined time  $\tau$  by starting its timer  $\mathbf{T}_m$  (ON/OFF) and switching from the *Active* execution mode to the *Standby* one. It returns to the *Active* execution mode when the timer finishes. The mobile agent can also wait an unknown time by only switching to the *Standby* execution mode. It returns to the *Active* execution mode when it consumes a *Token* created by another mobile agent. The mobile agent setting consists of the mobile agent state ( $\mathbf{A}_S$ ), its execution mode  $\mathbf{M}$  (*Active* or *Standby*) and its timer mode  $\mathbf{T}_m$  (ON/OFF).  $\mathcal{M}_A$  can be then represented as follows:

$$\mathcal{M}_A = \{\{M, A_S, T_m\}\}$$

The place settings consist of the place state  $\mathbf{P}_S$ , the number of tokens on the current place  $\mathbf{T}_K$  and a port number  $\mathbf{N}_p$  denoting the incoming (resp. the outgoing) port in use (resp. to be used) by the mobile agent (-1 if the mobile agent has been staying on the place and will not move).  $\mathcal{P}$  can then be represented as follows:

$$\mathcal{P} = \{P_S, N_p, T_K\}$$

such that:

$P_S$  : the current place state,

$N_p$  : The identity of the port crossed by the mobile agent (= -1 if it does not exist),

$T_K$  : the number of tokens on the current place.

Finally a mobile agent transition has the following form:

$$\begin{aligned} & (\{M, A_S, T_m\}, \{P_S, N_p, T_K\}) \\ & \Rightarrow \\ & (\{\{M', A'_S, T'_m\}, \{M^c, A_S^c, T_m^c\}\}, \{P'_S, N'_p, T'_K\}) \end{aligned}$$

The mobile agent  $\mathbf{a}$  can create a clone. This latter has the same transition system. But it may have a different state  $s_c$ . This state is assigned by  $\mathbf{a}$  when it creates the clone. Let  $\mathbf{a}$  be a mobile agent in the state  $s$  and let  $\mathbf{p}$  be a place in the state  $q$ , the **elementary transitions** associated to  $\mathbf{a}$  transforms  $s$  into  $s'$  and  $q$  into  $q'$  and indicates that:

**Class 1:**  $\mathbf{a}$  is on  $\mathbf{p}$  and it is still there.

$$\begin{aligned} & (\{\{\mathcal{A}_c, s, O_{FF}\}\}, \{q, -1, 0\}) \\ & \Rightarrow \end{aligned} \tag{1}$$

$$(\{\{\mathcal{A}_c, s', O_{FF}\}\}, \{q', -1, 0\})$$

**Class 2:**  $\mathbf{a}$  arrives on  $\mathbf{p}$  through the port  $in$ .

$$\begin{aligned} & (\{\{\mathcal{A}_c, s, O_{FF}\}\}, \{q, in, 0\}) \\ & \Rightarrow \end{aligned} \tag{2}$$

$$(\{\{\mathcal{A}_c, s', O_{FF}\}\}, \{q', -1, 0\})$$

**Class 3:**  $a$  leaves  $p$  through the port  $out$ .

$$\begin{aligned} & (\{\{\mathcal{A}_c, s, O_{FF}\}, \{q, -1, 0\}\}) \\ & \Rightarrow \\ & (\{\{\mathcal{A}_c, s', O_{FF}\}, \{q', out, 0\}\}) \end{aligned} \quad (3)$$

**Class 4:**  $a$  is already on  $p$ , it creates a clone  $a_c$  in the state  $s_c$ .

$$\begin{aligned} & (\{\{\mathcal{A}_c, s, O_{FF}\}, \{q, -1, 0\}\}) \\ & \Rightarrow \\ & (\{\{\mathcal{A}_c, s', O_{FF}\}, \{\mathcal{A}_c, s_c, O_{FF}\}, \{q', -1, 0\}\}) \end{aligned} \quad (4)$$

**Class 5:**  $a$  is killed.

$$\begin{aligned} & (\{\{\mathcal{A}_c, s, O_{FF}\}, \{q, -1, 0\}\}) \\ & \Rightarrow \\ & (\{\}, \{q', -1, 0\}) \end{aligned} \quad (5)$$

**Class 6:**  $a$  starts its timer and switches to the *Standby* mode.

$$\begin{aligned} & (\{\{\mathcal{A}_c, s, O_{FF}\}, \{q, -1, 0\}\}) \\ & \Rightarrow \\ & (\{\{\mathcal{S}_t, s', O_N\}, \{q', -1, 0\}\}) \end{aligned} \quad (6)$$

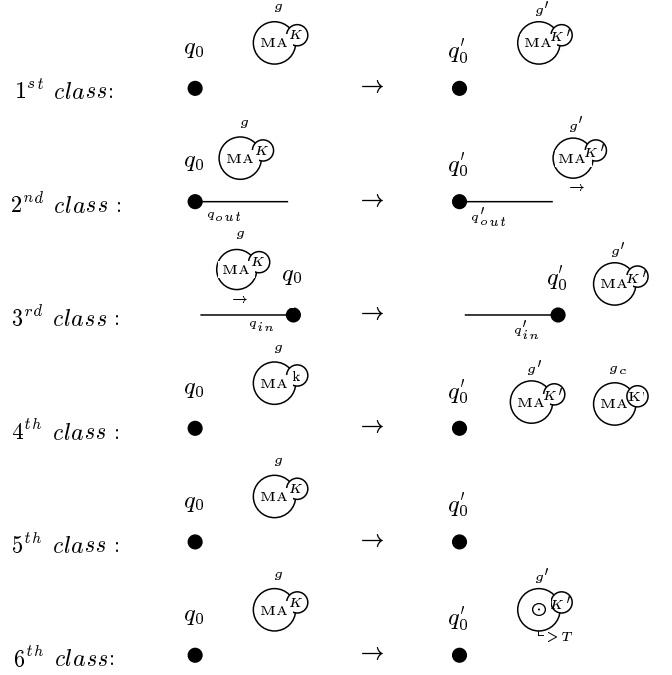
In order to simplify the description of mobile agent algorithms using transition systems, to follow the execution of these systems and to be able to check and to prove properties of elaborated algorithms we propose, in the following subsection, a graphical notation of the various classes of transitions.

### 2.3 Proposed notation for transitions

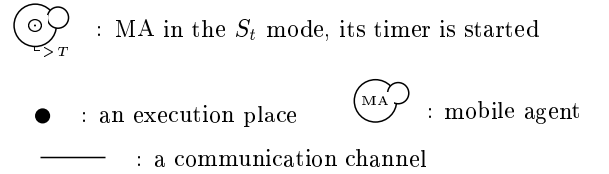
We present in this subsection the graphical notation developed to design transition systems. This notation, inspired from rewriting rules [18], allows the design of transitions in a simple and intuitive way. Moreover, it makes easy the presentation and the description of algorithms properties and the development of proofs. We divide this subsection in three parts. In the first part, we focus on graphical notation of *elementray transitions*. In the second, we present proposed notation to *Run-control* transitions. In the last part, we define and give some examples of *composed transitions*.

#### 2.3.1 Graphical notation for transitions

Let  $a$  be a mobile agent belonging to a mobile agent system and let  $T_a$  be the transition system associated to  $a$ . In Figure 1, we display a graphical notation of the various classes of *elementray transitions* explained in the previous subsection. Transitions presented in Figure 1 make it possible to graphically describe the state of mobile agent system entities before and after a transition. In every transition, we represent only the mobile agent and the place on which it is executed because only these two entities will possibly change their states.



With:



**Figure 1: Proposed notation for elementary transitions**

1. In the first class of Figure 1, the only changed states are  $q$  the state of the place and  $s$  the mobile agent global state. The place state before the transition is  $q=(q_0, q_1, \dots, q_d)$  and after the transition becomes  $q'=(q'_0, q_1, \dots, q_d)$ . The agent global state is  $s=(g, K)$  before the transition and becomes  $s'=(g', K')$  after the transition.
2. In the second class of Figure 1, the mobile agent leaves the place through the port *out* and changes its state, the place state and the port state (*out*). The place state before the transition is  $q=(q_0, q_1, \dots, q_{out}, \dots, q_d)$  and after the transition becomes  $q'=(q'_0, q_1, \dots, q'_{out}, q_d)$ .
3. In the third class of Figure 1, the mobile agent has just arrived on the place through the port *in*. It modifies its state, the place state and the arrival port state (*in*). The place state before the transition is  $q=(q_0, q_1, \dots, q_{in}, \dots, q_d)$  and after the transition becomes  $q'=(q'_0, q_1, \dots, q'_{in}, q_d)$ .
4. In the fourth class of Figure 1, the mobile agent keeps its place, it changes its global state (it is  $s=(g, K)$  and becomes  $s'=(g', K')$ ), it change the state of the place (it is  $q=(q_0, q_1, \dots, q_{in}, \dots, q_d)$  and it become  $q'=(q'_0, q_1, \dots, q_d)$ ) and produces a clone. The global state of the new agent  $a_c$  is  $s_c=(g_c, K')$ .
5. In the fifth class of Figure 1, the mobile agent changes the state of the place from  $q_0$  to  $q'_0$  and dies.
6. In the sixth class of Figure 1, the mobile agent changes its state and the place state, starts its timer and switches to the *Standby* mode.

## 2.4 Case study: spanning tree computation without local detection of the global termination

In this subsection, we illustrate our model by presenting a solution to the spanning tree problem using mobile agents. We formally present the solution by a transition system describing the mobile agent algorithm. We have already proposed a correctness and termination proofs of this algorithm in [14].

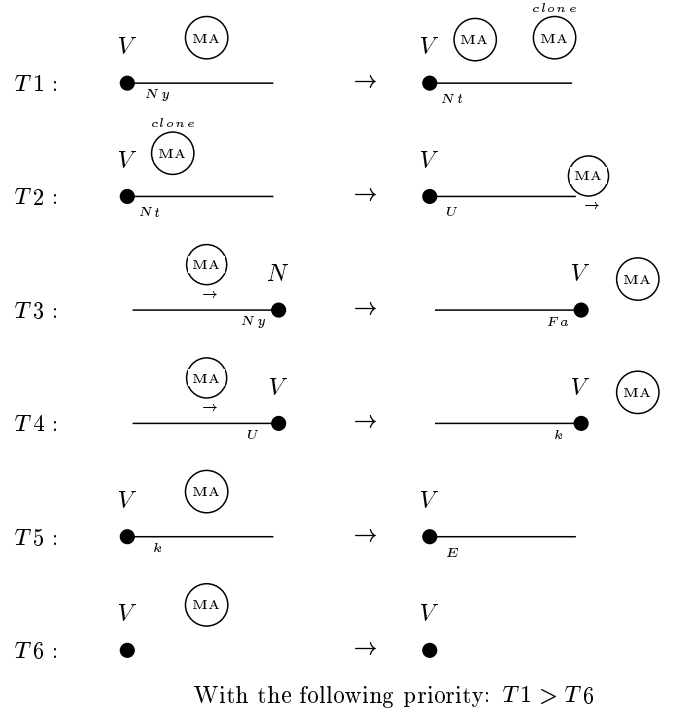
Spanning-tree construction is a classical problem in computer science. In a distributed computing environment, the solution of this problem has many practical motivations. It also has distinct formulations and requirements [26].

In a mobile agent system ( $\mathcal{MaS}$ ), the construction of a spanning tree means to move the system from an initial system configuration ( $C_0$ ), where each place is just aware of its local state, to a system configuration where

1. Each place knows its neighbors in the tree (father and sons)
2. All the channels, but those which ports are in the state E (means excluded), constitute the spanning tree of all places.

The mobile agent system we deal with for the spanning tree problem considers undistinguishable places. There is no knowledge of the size or the topology.

The transitions system  $\mathbb{T}a$  describing the mobile agent algorithm is presented in Figure 2.



**Figure 2: Transition system solving the spanning tree problem without local detection of the global termination**

In Figure 2, we present the graphical notation of the transition system. A mobile agent, running in a place, creates a clone for every port in the state  $Ny$  and changes the port state to  $Nt$  (T1 of Figure 2). When all ports are in the state  $Nt$ , the mobile agent is killed (T6 of Figure 2). Every clone moves from the home place by a port in the state  $Nt$  and changes it to  $U$ . When a mobile agent arrives in unexplored place (in the state  $N$ ), it changes the place state to  $V$  and the arrival port state to  $Fa$  (T3 of Figure 2). When it arrives in an already explored place, it changes the arrival port state to  $k$  (T4 of Figure 2). It stops running after changing the arrival port state from  $k$  to  $E$  (T5 of Figure 2).

In this section, we have presented the model notations. To illustrate these formal notations, we have also presented a case study for the spanning tree problem considering fault-free distributed systems. Using always the model notations, we will present in the next section a technique to tolerate transient faults in mobile agent systems. We also present a methodology to add this technique to any mobile agent algorithm. As a case of study, we use the methodology to acquire a fault tolerant behavior to the solution, solving the spanning tree problem, presented in this section.

## 3. A FAULT TOLERANT TECHNIQUE FOR MOBILE AGENT SYSTEMS

We present in this section a technique tolerating transient faults of distributed system entities. This technique is formally presented using our model notation. We present also a methodology to add this technique to any mobile agent dis-

tributed algorithm. We present, as a case study, the mobile agent transition system solving the spanning tree problem in a faulty environment. This transition system results from adding our technique transitions to the transition system given in the previous section. We already proved the correctness of the resulting transition system but we will not present the proof in this paper due to lack of space.

### 3.1 Failure model for mobile agent systems

In traditional distributed systems, a variety of models have been proposed to describe failing behavior of these systems. All these models have a *host – central* view. These models can not be extrapolated to mobile agent systems in which we must emphasize mobile agents which are prone to failure. In fact, some efforts are established to propose adapted fault models to mobile agent systems [29]. These models emphasize mobile agent faults by presenting *mobile agent- central* view fault models.

In this paper, we present a scheme for tolerating mobile agent crashes due to sites (places) and/or links transient failures. Our scheme provides a technique to operate in presence of transient failures of the underlying distributed system. It also provides a methodology which can be used to add fault tolerant behavior to any mobile agent distributed algorithm formally presented using our model notation.

Let suppose that a fault duration is  $F$ . We suppose that transient failures do not occur in the period of  $F + \tau$  in the neighboring of the faulty entity (places and channels) where  $\tau$  is the mobile agent timer length. We suppose also that a place transient failure do not occur while a mobile agent is running on it. Indeed, we suppose that there is no malicious places: a mobile agent arriving on a failing place crashes. A place or a link failure is temporary: the distributed system may return to a normal state after a labs of time. Finally, we suppose that a place failure do not change the whiteboard (the memory area allocated to the mobile agent system). We can suppose that an update of the whiteboard triggers the same one on a stable storage.

### 3.2 The fault tolerance technique

By using transitions defined in Section 2, we have already designed and proved a catalog of distributed algorithms (see for example [11, 12, 10]). We have assumed while designing these algorithms that the underlying distributed system is fault-free.

We propose in this section a fault tolerant technique. This technique, based on a set of transitions, allows a mobile agent to overcome transient failures. To make a mobile agent algorithm fault tolerate, we need simply to add the designed set of transitions to the transition system describing the mobile agent algorithm (we will describe later the methodology to be used to add the technique transitions to any mobile agent distributed algorithm).

Our technique assumes that:

- Every mobile agent can be distinguished by its role. A mobile agent is created by the mobile agent system to execute an algorithm is called a *Master*: its role is then a *Master*. We assign for mobile agents created by the fault tolerant technique other roles.
- Every *Master* in the mobile agent system has replicas (*clones*). We designate by *image* the mobile agent

clone created by the mobile agent algorithm and *replica* the mobile agent clone created by the fault tolerant technique.

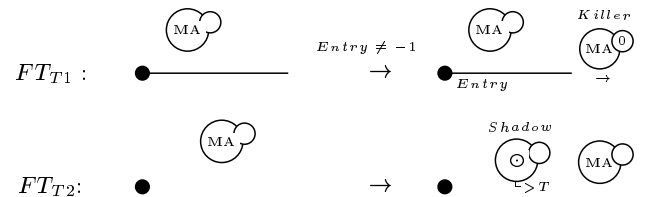
- Every mobile agent (*Master*) existing in the system has a unique *Fault identity*. A *replica* has the same one and an *image* has a distinct *Fault identity*.
- For every *Master* is associated a memory location in every place whiteboard. This memory location is identified by the *Master Fault identity*. This memory is intended to store information needed by the fault tolerant technique.
- A *Master* arriving on a place, can memorize the entry port in an attribute *Entry* (it is initialized to -1).
- A *Master* contains an attribute *Counter* which is incremented on every visited place (it is initialized to 0). This attribute is saved (or updated if already exists) in the *Master* allocated memory.
- The *Master* saves its state in its allocated memory before leaving the place or before being killed.
- The *Master* saves the identity of the leaving port in its allocated memory using an attribute called *Leaving* before leaving the place.

Note that for any *replica* the state will indicate its *role*.

On every place, the *Master* executes some actions dictated by its algorithm. It finishes its task on every place and:

- if it will leave the place, it creates a *replica* in the current place in the role *Shadow* (the *Shadow* agent or *Shadow* for short), in the mode *Standby* and puts on the *Shadow* timer.
- it sends through the *Entry* port (if  $Entry \neq -1$ ) a *replica* in the role *Killer* ( its role is the *Killer* agent or the *Killer* for short). This later aim is to kill the *Shadow* existing on the last visited place.

This step can be formally presented by the two transitions of Figure 3.



**Figure 3: Transitions executed before leaving a host by a *Master***

**Note:** Only the *Master* can execute the transitions of Figure 3 (or a *replica* which is assigned this role).

The *Killer*, which is sent by the *Master* to the last visited place, creates on the arrival place a token marked with the *Fault identity* of the *Master*. This token role is to wake up

the *Shadow* created by the master. If the *Shadow* is already killed (for example by a place transient failure), the token will be automatically deleted.

This step can be formally presented by the transition of Figure 4.

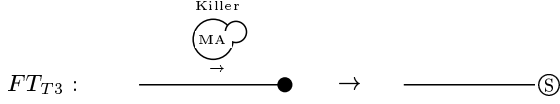


Figure 4: The transition executed by the Killer

The created *Shadow* on a given place may act in two different ways:

- If it receives a token from the *Killer*, it will be just killed,
- else when its timer finishes, it creates and sends by the port *Leaving* a replica in the role *Observer* (the *Observer* agent or *Observer* for short) and returns to the *Standby mode* after restarting its timer.

This step can be formally presented by the two transitions of Figure 5.

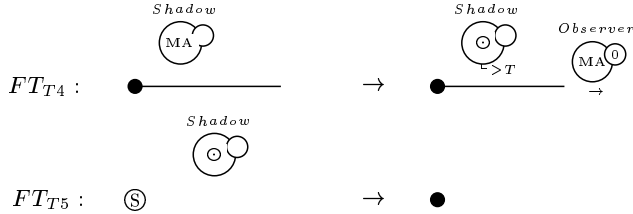


Figure 5: The transitions executed by a Shadow

The *Observer* collects, before leaving its home place using the *Leaving* port, from the whiteboard the saved values of the attributes *Counter* and *State*. Upon arriving on the destination place, it compares the carried *Counter* with the local value (if it exists, else we suppose that it is equals 0). If the carried value is less than the saved value on the current whiteboard, the *Observer* changes its role to *Killer* and returns back. Otherwise, it notices that the *Master* agent is lost by a link or a place transient failure, it changes its role to *Master* and its state to the carried state and does the necessary actions associated with the new role.

This step can be formally presented by the two transitions of Figure 6.

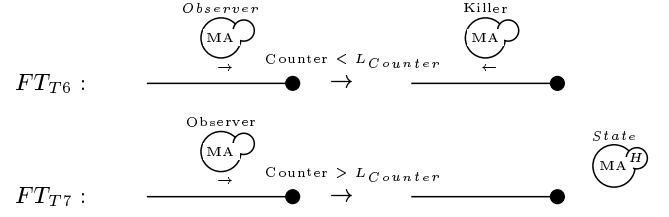


Figure 6: The only transitions executed by an Observer

When the *Master* fails to arrive to the next place (by a link or a place transient failure), the *Shadow* suspects that when its timer finishes. It sends an *Observer* to the next place (using the port *Leaving* saved on the current memory location). The *Observer* may also fail if the transient failure remains, the *Shadow* will send another *Observer* latter. When the *Observer* succeeds arriving to the destination place. The local value of *Counter* must then be less than the *Observer* carried value. In this case, the *Observer* changes its role to *Master*. It does then the task associated with the *Master* role.

When the *Master* finishes its task on a place, it sends a *Killer* to the last visited place to kill the *Shadow*. The *Killer* may be lost by a transient failure. The *Shadow*, after the timer finishes, notices that there is a problem. It sends an *Observer* to the *Master* destination place. The *Shadow* returns then to the *Standby* mode by restarting its timer. The *Observer* notices that the *Killer* is lost and not the *Master* by comparing the carried value of the *Counter* with the local one. The *Observer* changes its role to *Killer* and returns back. If the *Observer* or the *Killer* fails to arrive to the destination place, the *Shadow* will send latter a new *Observer* or it may be killed by a transient failure occurring on its place.

#### 4. ADDING FAULT TOLERANT BEHAVIOR TO MOBILE AGENT DISTRIBUTED ALGORITHMS

Given a mobile agent system, let  $a$  be a mobile agent and  $T_a$  the transition system describing the mobile agent algorithm. To enable the mobile agent fault tolerance, we need to add to  $T_a$  the fault tolerant transitions described below. We must then follow the following steps :

- For each transition  $T_x \in T_a$  belonging to class 3, we add an instance of the two transitions  $FT_{T_1}$  and  $FT_{T_2}$  such that the guards of these two transitions are changed to the guard of  $T_x$ . We give the labels  $FT_{T_1x}$  and  $FT_{T_2x}$  for the added transitions. We add the priorities:  $FT_{T_1x} > T_x$  and  $FT_{T_2x} > T_x$ . If there exists a priority  $T_y > T_x$ , we need to add a new priority  $T_y > \{FT_{T_1x}, FT_{T_2x}\}$ .
- For each transition  $T_x \in T_a$  belonging to class 5, we add an instance of the transition  $FT_{T_1}$  such that the guard of this transition is changed to the guard of  $T_x$ . We give the label  $FT_{T_1x}$  for the added transition. We add the priorities:  $FT_{T_1x} > T_x$ . If there exists a priority  $T_y > T_x$ , we need to add a new priority  $T_y > FT_{T_1x}$ .

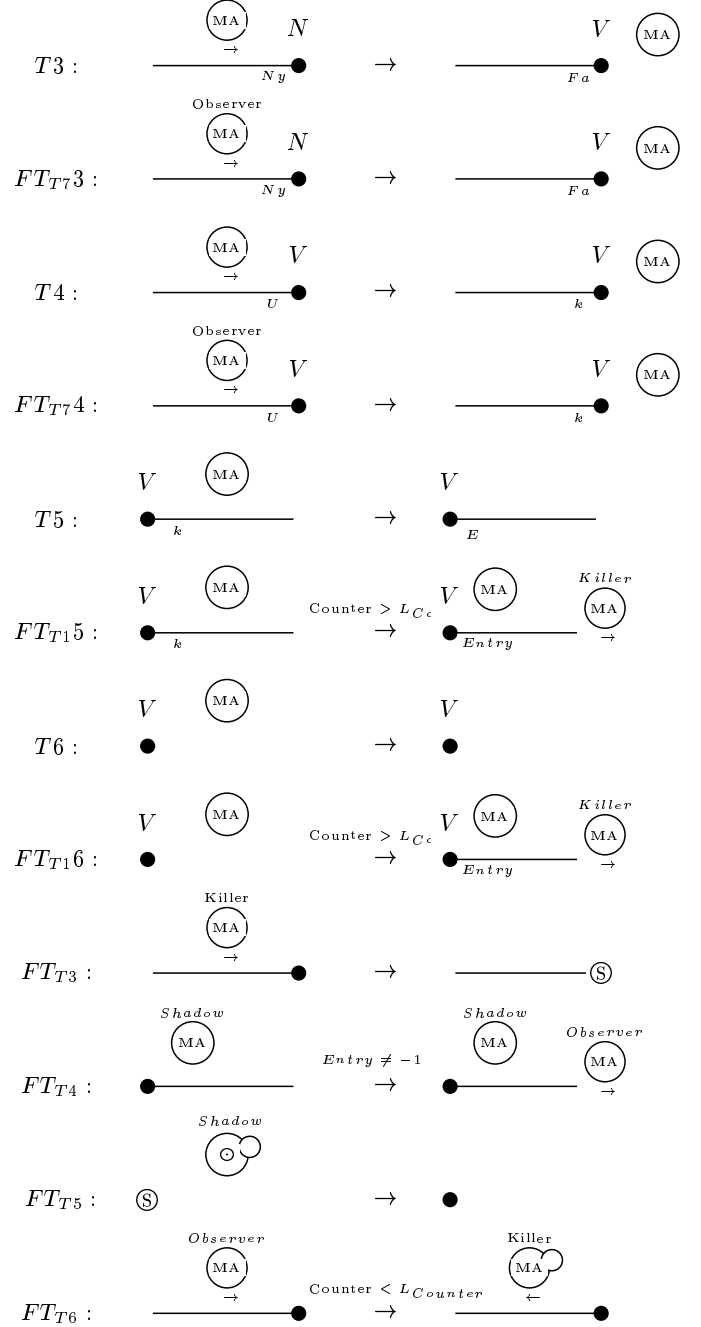
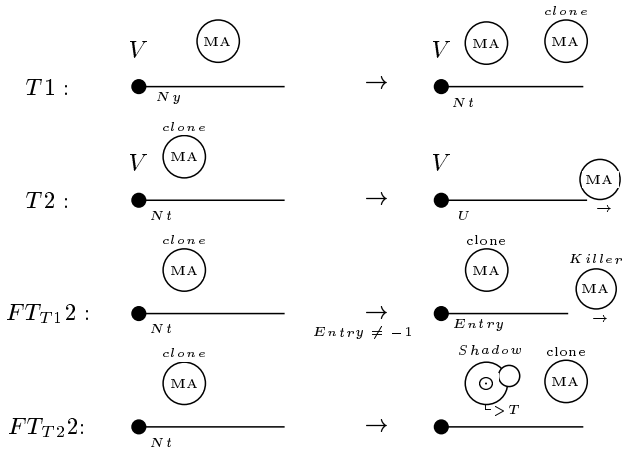
- For each transition  $T_x \in T_a$  belonging to class 2, we add an instance of the transition  $FT_{T7}$  such that the action of this transition is changed to the action of  $T_x$  and we add the guard of  $T_x$  to the guard of  $FT_{T7}$  but the state of the mobile agent is kept to *Observer*. We give the label  $FT_{T7x}$  for the added transition.

- In the last step we add to the obtained transition system the set  $\{FT_{T3}, FT_{T4}, FT_{T5}, FT_{T6}\}$

#### 4.1 Case study: the spanning tree algorithm

To better explain the steps presented above, we consider the mobile agent transition system solving the spanning tree problem without detection of local termination (see Figure 5 in [14]). We present, in Figure 7, the new transition system computing a spanning tree without detection of local termination and tolerating transient faults.

- $T2$  belongs to class 3. We added then the two transitions  $FT_{T12}$  and  $FT_{T22}$  together with the necessary priorities (see Figure 7) as mentioned in the step 1.
- $T5$  and  $T6$  belongs to class 5. We added then the transitions  $FT_{T15}$  and  $FT_{T16}$  together with the necessary priorities (see Figure 7) as mentioned in the step 2.
- $T3$  and  $T4$  belong to class 2. We added then the transitions  $FT_{T73}$  and  $FT_{T74}$  (see Figure 7) as mentioned in the step 3.
- The last added transitions are  $\{FT_{T3}, FT_{T4}, FT_{T5}, FT_{T6}\}$  (see Figure 7) as mentioned in the step 4.



With the following priorities:  $T1 > \{T6, FT_{T16}\}$ ,  
 $FT_{T15} > T5, FT_{T16} > T6, FT_{T12} > T2$ ,  
 $FT_{T22} > T2$

**Figure 7: Transition system solving the spanning tree problem without detection of local termination in spite of transient faults**

The resulting transition system (see Figure 7) describing the mobile agent algorithm acquires the mobile agent a fault tolerant behavior. It then solves the spanning tree problem in spite of transient faults.

## 5. CONCLUSION AND FUTURE WORKS

In this paper, we tried to use the model notation presented in [14] to design fault tolerant mobile agent distributed algorithms. To this aim, we have proposed a fault tolerant technique to be used in mobile agent distributed algorithms already designed to be used in fault-free distributed systems. We have also proposed a methodology to add this technique in already designed mobile agent transition systems. As a case study, we have presented the integration of our technique in already existing transition system solving the spanning tree problem.

We plan in the future to extend ViSiDiA [2] to support faulty distributed systems and thus allowing us to simulate fault tolerant mobile agent algorithms.

## 6. REFERENCES

- [1] H. Baala, O. Flauzac, J. Gaber, M. Bui, and T. El-Ghazawi. A self-stabilizing distributed algorithm for spanning tree construction in wireless ad hoc networks. *J. Parallel Distrib. Comput.*, 63(1):97–104, 2003.
- [2] M. Bauderon, S. Gruner, Y. Metivier, M. Mosbah, and A. Sellami. Visualization of distributed algorithms based on graph relabelling systems. *Electronic Notes in Theoretical Computer Science*, 50(3):227–237, 2001.
- [3] G. Cao and M. Singhal. Mutable checkpoints: a new checkpointing approach for mobile computing systems. *Parallel and Distributed Systems, IEEE Transactions on*, 12(2):157–172, Feb 2001.
- [4] J. Chalopin, E. Godard, Y. Métivier, and R. Ossamy. Mobile agent algorithms versus message passing algorithms. In *10th International Conference On Principles Of Distributed Systems (OPODIS 2006)*, volume 4305 of *Lecture notes in computer science*, pages 185–199. Springer, 2006.
- [5] J. Czyzowicz, D. Kowalski, E. Markou, and A. Pelc. Searching for a black hole in synchronous tree networks. *Comb. Probab. Comput.*, 16(4):595–619, 2007.
- [6] S. Dobrev, P. Flocchini, G. Prencipe, and N. Santoro. Mobile search for a black hole in an anonymous ring. In *DISC '01: Proceedings of the 15th International Conference on Distributed Computing*, pages 166–179, London, UK, 2001. Springer-Verlag.
- [7] S. Dobrev, P. Flocchini, G. Prencipe, and N. Santoro. Searching for a black hole in arbitrary networks: optimal mobile agent protocols. In *PODC '02: Proceedings of the twenty-first annual symposium on Principles of distributed computing*, pages 153–162, New York, NY, USA, 2002. ACM.
- [8] E. N. M. Elnozahy, L. Alvisi, Y.-M. Wang, and D. B. Johnson. A survey of rollback-recovery protocols in message-passing systems. *ACM Comput. Surv.*, 34(3):375–408, 2002.
- [9] F. C. Gärtner. Fundamentals of fault-tolerant distributed computing in asynchronous environments. *ACM Comput. Surv.*, 31(1):1–26, 1999.
- [10] M. Haddar, A. Hadj Kacem, Y. Metivier, M. Mosbah, and M. Jmaiel. Electing a leader in the local computation model using mobile agents. *Computer Systems and Applications, 2008. AICCSA 2008. IEEE/ACS International Conference on*, pages 473–480, 31 2008–April 4 2008.
- [11] M. A. Haddar, A. H. Kacem, Y. Métivier, M. Mosbah, and M. Jmaiel. Distributed algorithms for mobile agents. Technical Report RR-143507, LaBRI, Bordeaux, France, July 2007.
- [12] M. A. Haddar, A. H. Kacem, Y. Métivier, M. Mosbah, and M. Jmaiel. Proving distributed algorithms for mobile agents: Examples of spanning tree computation in anonymous networks. In L. N. in *Computer Science*, editor, *ICDCN*, volume 4904, pages 286–291. Springer-Verlag, 2008.
- [13] M. A. Haddar, A. H. Kacem, Y. Métivier, M. Mosbah, and M. Jmaiel. A distributed computational model for mobile agents. In L. N. in *Artificial Intelligence*, editor, *10th Pacific Rim International Workshop on Multi-Agents, PRIMA 2007*, volume 5044, pages 416–421. Springer-Verlag, 2009.
- [14] M. A. Haddar, A. H. Kacem, M. Mosbah, and M. Jmaiel. A distributed computational model for mobile agents. *Grid Computing and Mobile agent Systems (to appear)*, 1(1), 2009.
- [15] B. Hamid and M. Mosbah. A formal model for fault-tolerance in distributed systems. In *International Conference on Computer Safety, Reliability and Security (SAFECOMP)*. Springer, septembre 2005.
- [16] R. Klasing, E. Markou, T. Radzik, and F. Sarracco. Hardness and approximation results for black hole search in arbitrary networks. *Theor. Comput. Sci.*, 384(2-3):201–221, 2007.
- [17] D. B. Lange and M. Oshima. Seven good reasons for mobile agents. *Commun. ACM*, 42(3):88–89, 1999.
- [18] I. Litovsky, Y. Mtivier, and E. Sopena. *Handbook of graph grammars and computing by graph transformation*, volume 3. World Scientific, 1999.
- [19] M. Lyu, X. Chen, and T. Y. Wong. Design and evaluation of a fault-tolerant mobile-agent system. *Intelligent Systems, IEEE*, 19(5):32–38, Sept.-Oct. 2004.
- [20] K. Park. A fault-tolerant mobile agent model in replicated secure services. In S. B. . H. *Lecture Notes in Computer Science*, editor, *Computational Science and Its Applications, ICCSA 2004*, volume 3043/2004, pages 500–509, 2004.
- [21] K. Park and A. K. Sood. Mare: A fault-tolerant mobile agent system. In H.-K. Kahng, editor, *ICOIN*, volume 3090 of *Lecture Notes in Computer Science*, pages 1035–1044. Springer, 2004.
- [22] T. Park, I. Byun, and H. Y. Yeom. Lazy agent replication and asynchronous consensus for the fault-tolerant mobile agent system. In *NETWORKING*, pages 1060–1071, 2004.
- [23] S. Pleisch and A. Schiper. FATOMAS - A fault-tolerant mobile agent system based on the agent-dependent approach. In *Proceedings of the IEEE Int. Conf. on Dependable Systems and Networks (DSN'01)*, Goterborg, Sweden, 2001. IEEE Computer Society.
- [24] W. Qu and H. Shen. Performance modelling of a fault-tolerant agent-driven system. *Communications, 2005. ICC 2005. 2005 IEEE International Conference on*, 1:153–157 Vol. 1, 16-20 May 2005.
- [25] W. Qu, H. Shen, and X. Defago. A survey of mobile

- agent-based fault-tolerant technology. In *PDCAT '05: Proceedings of the Sixth International Conference on Parallel and Distributed Computing Applications and Technologies*, pages 446–450, Washington, DC, USA, 2005. IEEE Computer Society.
- [26] N. Santoro. *Design and Analysis of Distributed Algorithms (Wiley Series on Parallel and Distributed Computing)*. Wiley-Interscience, 2006.
- [27] D. Shiao. Mobile agents: A new model of intelligent distributed computing. Technical report, IBM DeveloperWorks, China, 2004.
- [28] M. Tasic and A. Zaslavsky. *Generic Fault-tolerant Layer Supporting Publish/Subscribe Messaging in Mobile Agent Systems*, pages 207–214. Enterprise Information Systems VII, 2006.
- [29] A. Tripathi and R. Miller. Exception handling in agent-oriented systems. pages 128–146, 2001.
- [30] J. Xu and S. Pears. A dynamic shadow approach to fault-tolerant mobile agents in an autonomic environment. *Real-Time Syst.*, 32(3):235–252, 2006.
- [31] J. Yang, J. Cao, and W. Wu. Cic: An integrated approach to checkpointing in mobile agent systems. *Semantics, Knowledge and Grid, 2006. SKG '06. Second International Conference on*, pages 4–4, Nov. 2006.