Towards a Multi Agents System Coupling Replication and Exception Handling

Mounira BOUZAHZAH[1], Ramdane MAAMRI[2]
[1]LIRE Laboratory
Science and Technologie Institute
University Center of Mila
Mila, Algeria
[2]LIRE Laboratory
Departement of Computer Science, Mentouri University
Constantine, Algeria
m.bouzahzah@centre-univ-mila.dz, rmaamri@umc.edu.dz

ABSTRACT: *Multi agents systems are formed of different independent entities placed in several machines. When an entity or an agent fails, it is the whole system that may be in a failure case. Through this paper, we will propose an approach that may guarantee fault tolerance in multi agents systems using two different techniques which are replication and exception handling. Replication uses redundancy to handle the problem of failures, while, exception handling gives the possibility of recovery in case of fault signal. To use these two different mechanisms within the same system, we use the idea that a multi agents system has a global goal or a mission which it has to achieve; this global goal can be divided into sub goals that may be simple or complex. A simple goal is realized by one single agent that is considered in this case as an important agent and it is replicated. Exception handling is used in case of complex goals. Within this work, we define a complex goal as the one that must be realized by more than one agent. The major aim of our work is to create a strong multi agents system that may benefit of the two techniques proposed to guarantee fault tolerance.*

Keywords: System's Mission, Simple Goal, complex Goal, Replication, Controller, The Goal Agent

Received: 14 November 2012, Revised 22 December 2012, Accepted 27 December 2012

1. Introduction

Using multi agents system in distributed applications design can cause a big obstacle if we consider the problem of system failures. In particular, multi agent systems are subject of faults of different origins. Several solutions have been proposed, by the research community in fault tolerance. A failure has not a specific definition; it depends of the types of faults considered by the proposed work in fault tolerance field. This paper deals mainly with failures that cause the deviation of the service delivered by the considered system this means that one action or more cannot be executed. We based our approach on the fact that each Multi agents system has a global goal to achieve, and which can be decomposed into a set of sub goals realized by different groups of agents. We consider a sub goal as a simple one if it needs one single agent to be achieved. The case of sub goals that are realized by a group of agents determines the second type of goals named complex ones. According to this goal's classification, we can define the group of the most critical agents of the system which consists of agents that realize sub goals by themselves only. At this level we propose to use of the technique called replication to guarantee fault tolerance in the group of agents responsible of realizing simple sub goals. An agent of control is added to the system to manage the group of simple goals, to verify the detection of fault and to use replicas to solve the problems. Each complex goal represents a group of agents that work concurrently to achieve this goal. Exception handling technique is used at this level to guarantee system recovery when an agent fails to carry on a specific action. This second part of our approach is based on the idea of adding specific agents to each goal's group of agent called the goal agent, we use Petri Nets to give a general model the activities of these agents that can detect, evaluate and solve a fault when it occurs. The whole system is controlled by another agent that is called the system's supervisor.

The rest of this paper is organized as follows: section 2 covers the related works in the field of fault tolerance. Section3 gives a description to the proposed approach; this section is divided into three main parts: the classification of agents according to the sub goals determined by the system, the group of critical agents and replication, and Exception handling and the complex sub goals. Section4 describes the roles assigned to both of the goal agents and the controller. It models their activities using Petri Nets formalism. Section 5 represents the general architecture of the system, and finally, Section6 that gives an insight into our future directions and concludes the paper.

2. Related Works

Different approaches are proposed to assure that a multi agents system can continue execution even, if some agents are failed. These approaches use mainly two mechanisms which are: replication and exception handling. A replicated software component is defined as a software component that has a representation on two or several machines [1]. These proposed approaches are classified into two broad families of fault tolerance [2]:

A. Fedoruk and R. Deters [3] introduce a new approach based on replication and proxies. A proxy managed communications between a group of agents and other groups of agents; and controls the interaction between this group of agents and the environment. Fedoruk chooses the passive replication strategy. So, his approach suffers from the same inconvenient as the passive replication.

Z. Guesssoum and al [1] introduce an automatic and dynamic replication mechanism. They evaluate dynamically the criticality of an agent using various data such as: time processing, the role taken by an agent in the system… This mechanism uses the interdependence graph and the agent through this approach has to interrupt its work in order to evaluate the criticality

A. Almeida et al [4] introduce a methodology for replication based on the concept of agent plan of actions each agent of the system evaluates its criticality according to a method proposed then agents that are more critical will be replicated using the framework DARX [5]. This approach treats the case of cooperative systems and it obliges agents to interrupt their executions in order to evaluate their criticality.

The use of replication gives the system the opportunity to achieve its global goal even in case of fault signaled by some replicated agents; but replicating all the agents of the system can cause difficulties when controlling the system. This inconvenient oblige as to use another technique of fault tolerance within the agents that cooperate to achieve complex goals; this technique is called exception handling. The mechanism of exception handling is proposed through different works and using different strategies:

[6, 7] propose specific systems for exception handling. The approach described in [6] introduces the concept of the supervisor which has the role of a handler for a group of system's agents. He defines two types of exceptions: the internal exceptions that are treated by the agent itself and the external ones handled by the supervisor which has a global access to the agents that it supervises. Hagg [7] proposes a strategy for exception handling using sentinels. Sentinels are guardian agents which protect the multi-agent system from failing in undesirable states. They have the authority to monitor communications in order to react to fault. This approach is costly in terms of computation and communication and it causes point of failure since sentinels, also, are subject of fault. These two works are based on the idea of controlling agent, this means the use of a generic agent that controls the execution process and handles exceptions when they occur.

The system described in [8] gives more important to agents' communications in terms in request/response. An exception is treated by handlers that try to solve the problem locally, but if it is not the case the request will be returned toward the agent asking for it. Complex services that invocate other agents to solve a request use a concertation function. This function has information concerning the criticality of each tasks executed and it judges an exception if it is to be ignored or token in consideration.

We can conclude that use the exception handling mechanism can help the system to recover in case of failure but this technique needs a good knowledge concerning specific details of the program.

3. The proposed approach

Combining replication and exception handling is proposed by [9] but this work tries to applies the SaGE proposed in [8] on a replication platform DIMAX [10]. That means applying the exception handling technique in a replicated environment. This means how to consider exceptions in case where many versions of one agent exist.

Our work tries to ameliorate the idea of combining the two previous techniques within the same system but it deals another case of combination. We have noted in the previous section that replicating all the agents of the system can cause a great charge and using exception handling necessitates a good knowledge of details; to solve these two problems we propose to divide the system's agents into groups some of them use the mechanism of replication, the others treat exceptions.

A multi agents system has a global goal or mission that it has to achieve; this goal can be decomposed into sub goals following the given criteria: simple goals are defined as goals that are realized by one single agent; replication is used within this group because agents are considered as too important. A complex goal is defined as the one which needs more than one agent to be achieved. The existing of two or more agents in cooperation to realize one complex goal gives us the idea of using exception handling within this type of groups. So, our approach uses replication an exception handling separately in different groups but they are combined through one multi agents system.

3.1 The agents' groups

Agents' groups or classes are used in different approaches such as: [3] and [6] to facilitate agents' classification. The framework DARX also deals with the notion of groups it uses the replicas groups. This framework associates a group within each replicated agent.

Through this work, we will create one group for agents that realize simple goals and we associate a group for each complex goal. The agents that belong to the simple goals group are considered as critical agents for the multi agents system, because if one of them fails the simple goal that it has to carry on will not be achieved; thus, e ey are replicated. Each agent will have only one replica. It is the agent that executes all the actions and transmits its current state to its replica. If the agent fails, its replica is activated by another agent called the Controller; this agent is the simple goals group's manager. In the case of complex goals, a group is created within each goal considered as complex and a goal agent is added to each group in order to treat exceptions or faults. The goal agent is an agent that controls the executions, detects failures and assures system recovery. It has the authority to ignore an exception signal or to consider it depending on the importance of the action that gives the signal. To handle an exception the goal agent creates for each action existing in the plan associated to the complex goal a list of agents that can execute this action. In case of failures detected and considered by the decision function, the goal agent treats it as follows:

- If the action failed has other agents that can execute it, the goal agent gives request to all the agents that are in the set of action to satisfy it.
- If this action cannot be executed by any agent in the system, the goals must dispose of program's modules that can execute such an action. The designer must give the goal agent the capacity to execute actions that exist in the plan of its associated goal and they are done by only one agent.

3.2 Replication and simple goals group

This approach uses the concept replication to solve the problem of fault tolerance in the simple goal group. There two strategies of replication (active and passive) that are used in different proposed works. The active replication is defined as the existence of several replicas which process concurrently all input messages [11]; this strategy allows speed recovery but leads to a high over-head. And the passive replication which indicates that only one activated replica processes all input messages and transmits its current state to its replicas in order to maintain coherence of the replicated group, and to constitute a point of return in case of failure. This strategy requires less CPU resources. Thus, we choose the use of the last strategy within the simple goals group, but we suggest using only one passive replica each time.

3.3 Exception handling and complex goal groups

Agents that cooperate to achieve a specific goal formed a complex goal group. We suggest the use of exception handling techniques to solve fault problems. At the designing time each action present in the plan that represent the complex goal must has a propriety that reflect the impact that it has on the realization of the whole complex goal. The designer fixes the value of this propriety using different criteria; we propose these two important points:

• According to the complex goal and the global one, the designer can determine semantic information to define the importance of an action.

• The numbers of necessary resources that are required for the execution of an action represent a factor to determine the importance of the action. Thus, when an action requires many resources to be executed, it is considered as being too important.

The propriety that the designer has to associate to each action defines a measure indicates the impact of the action of action on the entire goal and it is called criticality [12].

The designer must study the environment and the goals of the system and then determine action's criticality as a set of ordered and limited values. The **min criticality**: generally represents the value 0 which is associated within an action that has no impact on the goal to be achieved. However, the **max criticality** represents an action that its failure causes the failure of the whole goal.

According to usual arithmetic, the median value of N numbers gives an index to divide a unit into two parts. So, the goal agent uses *the Procedure Decision* in order to determine the critical actions.

```
Procedure Decision ( )
{
  Sum_criticalities = 0;
  For all action in the goal plan {
        Read C (A); / * C (A) the action A criticality * /
        Sum_criticalities = Sum_criticalities + C (A);
    }
    For each action A {
      If C (A) > = (Sum_criticalities / number of goal's actions)
        exception = 1;
      Else
        exception = 0 ;}
/ * Exception = 1 represents a critical action, however, exception = 0 indicates an action with less impact* /
} / * end Decision */
```

The goal agent can ignore an action failure when the other actions continue execution, and this action is considered as having less impact. Otherwise, the fault must be treated; and a request is given for other agents when the failed action can be realized by other agents in the system; otherwise, the goal agent must execute it by itself using specific program modules.

4. Modeling the Controller and the Goal Agent Activities using Petri Nets

Petri Nets [12] are promising tool for modeling system activities mainly those which are known as being concurrent, distributed, and parallel. As a graphical tool, Petri Nets can be used to simulate the activities of an agent.

4.1 The Controller

It is the simple goals group's manager it allows agent replication using the passive strategy which means that one agent executes its goal or plan and transmit its current state to its replica that can replace it in case of failure. The controller verifies and detects failure among its group's agents using the idea that allows the system to check if an agent is still alive and that it does not function in a synchronous environment [13]. The controller achieves this service within the use of a clock that initializes the control messages sent to the group's agents. Each agent will be controlled using the time criteria [14]. Thus, a timer called *failure _ timer* that represents the max response time of the agent will be used to calculate the time. The controller uses ***the procedure Detection_failure*** which calculates the response time of agents, if it never exceeds the max response time, a failure is detected. Since the detection of failure, the passive replica will be active and another passive replica will be added to the system.

```
Procedure Monitor (i) {
   failure _ timer [i] = Max failure_time;
)} / * End Monitor * /
Procedure Detection_failure ( ) {
    For all agent i watched {
      failure_timer [i] = failure_timer [i] - 1;}
    For all (agent i watched )AND ( failure_timer  [i] = 0)
             {
             Activate_replica (replica passive i);
             Replication_ passive (agent i);
             Replace (agent i, replica_actif i);}
}/ * End detection_failure* /
```

Since the controller is also an agent we propose that it will have a passive replica that keeps information concerning the last task performed by the agents and provides a checkpoint in case of failure. The detection of failure and the replacement of the controller are tasks performed by the system's supervisor following the similar method used by the controller.

Once the controller activities are described, we can model them using Petri Nets. This formalism allows parallel processes modeling. Figure 1 shows the controller model.

P0- Waiting state.

T1- Receiving message.

P1- Message received.

T2- Checking the type of the message:

• A contract message

• A checking message sent by the system supervisor

• An agent's response.

P2- Message asking for contract.

T3- Establishing contract.

P3- Contract established.

T4- Replicating the concerned agent.

P4- Agent replicated.

T5- Creating the checking message for the replicated agent.

P5- Controlling message.

T6- Initializing the failure time.

P6- The timer initialized.

T7- Sending the controlling message.

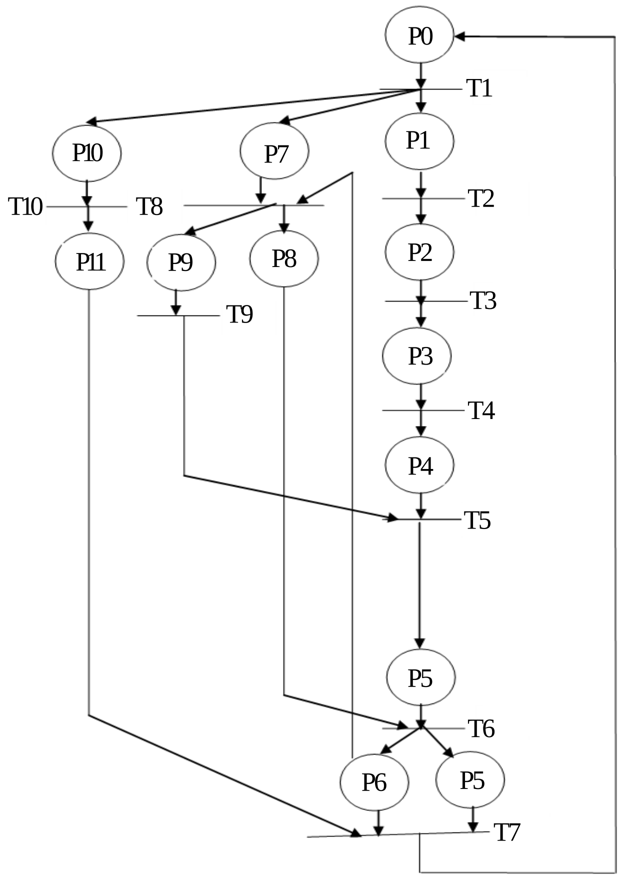


Figure 1. The controller model

P7- Agent’s response.

T8- Checking if the response is at time.

P8- Agent alive.

P9- Failing agent.

T9- Activating the passive replica.

P10- Checking message sent by the agent supervisor.

T10- Creating a response.

P11- Response created.

4.2 The Goal Agent

The goal agents are agents added to the system in order to treat exceptions that occur in complex goal groups. The goal agent, first, has to detect faults among agents that participate to achieve the complex goal. Failure detection is done using the same technique as the controller. If an agent is considered as a failed one, the action that it has to achieve gives an exception signal. The goal agent verifies the criticality of the action and returns a decision that may ignore the exception or take it in consideration. The exception is handled by the goal agent using the method mentioned in the previous sections. The following figure (Figure2) describes the goal agent activities using Petri Nets.

P0- Waiting state.

T1- Receiving message.

P1- Message received.

T2- Checking the type of the message:

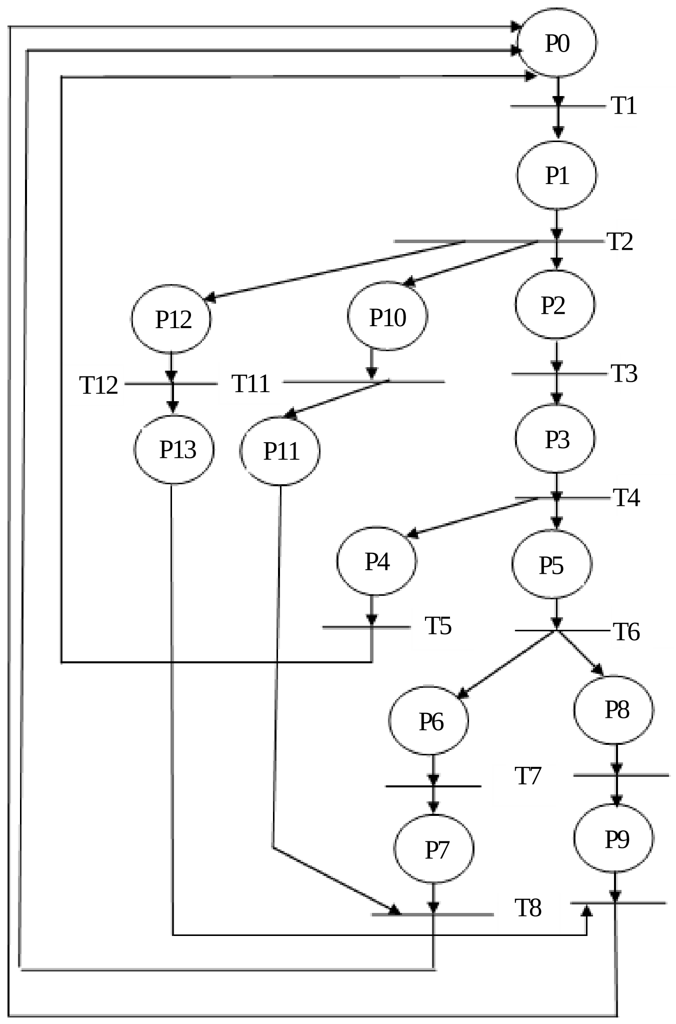


Figure 2. The goal agent model

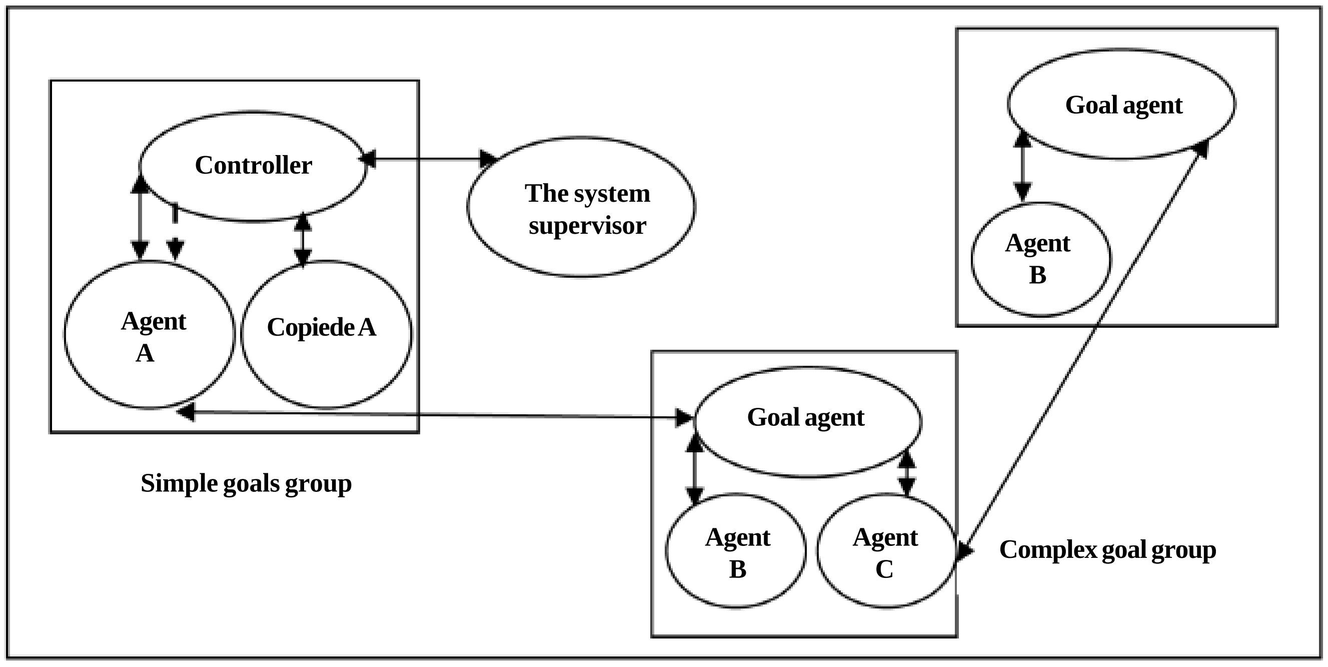
• A failure message.
• A checking message sent by the system supervisor.
• An agent's response.

P2 - Failure message.
T3- Treatment of failure
P3 - The result.
T4- Analyzing the result.
P4- Failure ignored.
T5- Ignoring the exception.
P5- failure considered.
T6- Treating the failure.
P6- Action done by one agent.
T7- Using program's modules to treat exceptions.
P7- Exception handled.
T8- Recovering the failure.

P8- Action done by other agents.
T9- Creating message for other agents to execute the action.
P9- Message created
T10- Sending message.
P10- Agent message.
T11- Adding the agent to the complex goal group.
P11- Agent added.
P12- Checking message sent by the agent supervisor.
T12- Creating a response.
P13- Response created.

5. The System's Architecture

In order to guarantee fault tolerance in dynamic multi-agent systems, we have added three types of agents that allow error detection and data recovering. The general architecture of the system is given by the following figure (Figure 3):



The system's architecture

The agents that are in the system includes: firstly, the controller and the goal agents these two kinds of agent are described in section 4, Secondly, the system's agents, and finally, the system supervisor. This agent has a procedure called ***the procedure history*** which saves the state of the environment at fixed interval of time in order to constitute a checkpoint in case of failure.

6. Conclusion and Perspectives

Our work treats the problem of failures in dynamic multi agent systems. It is based on the idea of dividing the whole system into groups using the notion of goal and to introduce two main mechanisms known in field of fault tolerance: the replication and exception handling. These two techniques are used separately but within the same system, the fact that allows the creation of a strong system that may benefit from the advantages given by the two techniques. The idea of dividing the global system into groups of sub goals makes it possible to separate the two strategies of fault tolerance. We propose the use of other added agents in the system to guarantee more effectiveness of the approach. Petri Nets give us a strong formalism to model the activities of the controller and the goal agents. Finally, we are interested in validating this work through a simulation that can provide real results using JADE platform.

References

[1] Guessoum, Z., Briot, JP., Faci, N., Marin O. (2004). An adaptive replication mechanism for SMA fault tolerant, *JFSMA*.

[2] Baroudi, R. (2008). An Approach for Estimating Criticality Agents Based on Fuzzy Classification, Mémoire de stage Master Computer ISI, LIP6, University Paris IX.

[3] Fedoruk, A., Deters, R. Improving fault-tolerance by replicating agents, *In*: Proceedings AAMAS-02, Bologna, Italy, p. 144-148.

[4] Almeida, A., Aknine S.et al. (2005). Method of replication based on plans for the fault tolerance of multi-agent systems, *JFSMA*.

[5] Marin.O, Marin.B, Sens.P. (2003). DARX: A Framework For Tolerant Support Of Agent Software, *In*: Proceedings of the 14th International Symposium on Software Engineering Reability, *IEEE*.

[6] Anand, T., Robert, M. (2000). Exception Handling in Agent Oriented Systems, Springer-Verlag.

[7] Hagg, S. (1996). A Sentinel Approach to Fault Handling in Multi-Agent Systems, *In*: Proceedings of the Second Australian Workshop on Distributed AI, Cairns, Australia, August 27.

[8] Souchon, F., Christophe, D., Christelle, U., Sylvain, V., Jacques, F. (2002). SaGE: une proposition pour la gestion des exceptions dans les system multi agents, Internal repport-LIRMM-02205.

[9] Kchir, S. (2010). Gestion des Exceptions dans un Système Multi Agents avec Replication, Mémoire de Master 2, Laboratoire d'Informatique de Robotique et de Micro-electronique, Montpellier.

[10] Faci, N., Guessoum, Z., Sailor, O. (2006). DIMAX: A Fault Tolerant Multi-Agent Platform, SELMA '06.

[11] Wiesmann, N., Pedone, F., Schiper, M., Kemme, B., Alonso, G. (2000). Database replication techniques: a three parameter classification, *In*: Proceedings of 19th IEEE Symposium on Reliable Distributed Systems (SRDS2000) Nüenberg, Germany, October . IEEE Computer Society.

[12] Almeida, A., Aknine, S.et al. (2006). A Predective Method for Providing Fault Tolerance in Multi-Agent Systems, *In*: Proceedings of the IEEE / WIC/ACM International Conference of Intelligent Agent Technologie (IAT'06) .

[13] Bourdeaud'huy, T. (2004). Techniques d'abstraction pour l'analyse et la synthèse de réseaux de petri, doctorat these, Lille.

[14] Huang YF, Chuang MH. (2006). Fault tolerance for home agents in mobile IP, Computer Networks 50.