SMA and Mobile Agents Actors for Distributed Testing

Salma Azzouzi[1] ,Mohammed Benattou [2] , My El Hassan Charaf[3], Jaafar Abouchabaka[4]

**[1] Laboratory Of Mathematics and Computer, University Ibn Tofail
Kenitra , 14 000, Morocco**

**[2] Laboratory Of Mathematics and Computer, University Ibn Tofail
Kenitra , 14 000, Morocco**

**[3] Laboratory Of Mathematics and Computer, University Ibn Tofail
Kenitra , 14 000, Morocco**

**[4] Laboratory Of Mathematics and Computer, University Ibn Tofail
Kenitra , 14 000, Morocco**

Abstract
The Multi Agent Systems are a paradigm of the most promising technology in the development of distributed software systems. They include the mechanisms and functions required to support interaction, communication and coordination between the various components of such a system. In the context of distributed test the activity of the test is performed by a set of parallel testers called PTCs (Parallel Test Components). The difficulty is in writing communication procedures of coordination and cooperation between the PTCS. In this context, we combine in this paper adaptable mobile agent with multi-agent systems technology to enhance the distributed test. The objective of this work is to eventually have a platform for compliance testing of distributed applications.

Keywords: *Distributed Testing, SMA, Mobile Agent, Actor, Mobile Actor.*

1. Introduction

Agent-based software engineering has become the key issues in modern system design. It provides a high-level abstraction and services for developing, integrating and system managing of distributed system applications. The component-based software engineering has promised, and indeed delivered significant improvements in software development.

Last years, products, models, architecture and frameworks suggest several key issues that will contribute to the success of open distributed systems. However, in practice, the development of distributed systems is more complex. The design process must take into account: the mechanisms and functions required to support interaction, communication and coordination between distributed components. Examples of such applications are systems comprising a set of components that broadcast commands among themselves, multicast controllers that coordinate messages between components, the systems using the alarm signals in non deterministic order, network and application systems, event queues, etc. The typical reaction of such systems is the generation of errors sets: time-outs, locks, channels and network failures.

Our preliminary experience, in the design and the implementation of the distributed testing application [1] of the broadcast and multicast channels. The basic idea of the proposed work [2] is to coordinate the testers by using a communication service parallel to the IUT through a multicast channel. Each tester interacts with the Implementation under Test (IUT) only through the attached port and communicates with the other testers through the multicast channel. The implementation of the proposed model has shown that the execution of distributed testers arise many time-outs problems influencing fault detection during the testing process. Object-oriented based, the development of such applications using the ``classical" objects is very difficult, like many possible ways of activating or deactivating event sources and to dispatch the call-backs. Others proposed research work based on the temporal Finite State Machine define the timing constraint, which the distributed testing application must be respected in real-time execution [3]. However, the implementation of these models generates a great number of synchronized exchange messages. We have proposed different ways to design and implement distributed testing systems using CORBA event services, active objects, and mobile agents.

In this paper, we show how to use the concept of agent and actor of the Javact platform gives a number of advantages, besides using a mobile agent include overcoming network latency, reducing network load, performing autonomous and asynchronous execution, and adapting to dynamic environments ([4], [5])

We envision a model of adaptive mobile agent, able to dynamically reconfigure to adapt to variations in its execution context (For example, If one of the testers crashes. what should the mobile agent do?).

Indeed in proposed prototype our testers are agent actors ([6], [7]) which integrate the concepts of agent and the concept of appointment for the communication. Each agent tester must create other agents to send messages to the actors whom it knows and to dynamically change behaviour by defining its behaviour for the next treatment.

The paper is structured as follows. Section 2 describes the architecture and modelling concept of distributed testing application and raises some synchronization problems in distributed testing implementation. Section 3 describes how JavAct agent actors are used in our model. Section 4 gives some conclusions and identifies future works

2. Distributed Test

The principle of the test is to apply input events to the IUT[1] and compare the observed outputs with expected results. A set of input events and planned outputs is commonly called a test case and it is generated from the specification of the IUT. We consider a test as a conform test if its execution is conform to its specification.

2.1 Architecture

The basic idea of the proposed architecture is to coordinate parallel testers called PTCS (Parallel Test Components) using a communication service in conjunction with the IUT. Each tester interacts with the IUT through a port PCO[2], and communicates with other testers through a multicast channel (Figure1).

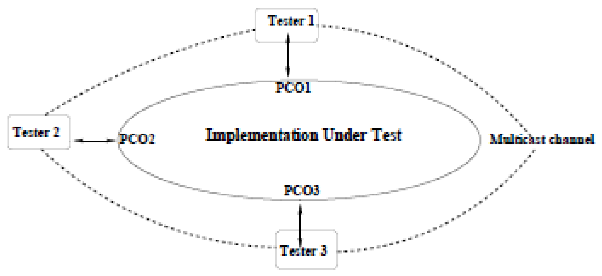

Fig. 1 :Test Architecture

An IUT (implementation under test) is the implementation of the distributed application to test. It can be considered as a "black-box", its behavior is known only by interactions through its interfaces with the environment or other systems. Each tester sends some stimulus to the IUT via their interfaces called PCOs (Points of Control and Observations) and observes the reactions. The external behavior of the IUT is observable via another interface type called IAP (Implementation Access Points). The difference between the PCO and the IAP is that PCOs are the logical points where communications are made, but the IAPs are the physical access points of the IUT. To synchronize the execution of testers, PTCS exchange coordination message. These messages encapsulate information that allows the test system to solve the problems of controllability and observability in order to obtain an optimal test. An example illustrating these problems is well detailed in [2].

2.2 Modeling by automaton

To approach the testing process in a formal way, the specification and the IUT must be modeled using the same concepts. The specification of the behavior of a distributed application is described by an automaton with n-port [8] (FSM Finite State Machine) defining inputs and the results expected for each PCO.

We denote Σk the input alphabet of the port k (PCO number k) and Γk the output alphabet of the port k. Figure 2 gives an example of 3p-FSM with set state FSM) with $Q = \{q0, q1, q2, q3, q4, q5\}$, q0 initial state, $\Sigma 1 = \{a\}$, $\Sigma 2 = \{b\}$, $\Sigma 3 = \{c\}$, and $\Gamma 1 = \{w, x\}$, $\Gamma 2 = \{y\}$, $\Gamma 3 = \{z\}$.

[1] **I**mplementation **U**nder **T**est
[2] **P**oint of **C**ontrol and **O**bservation

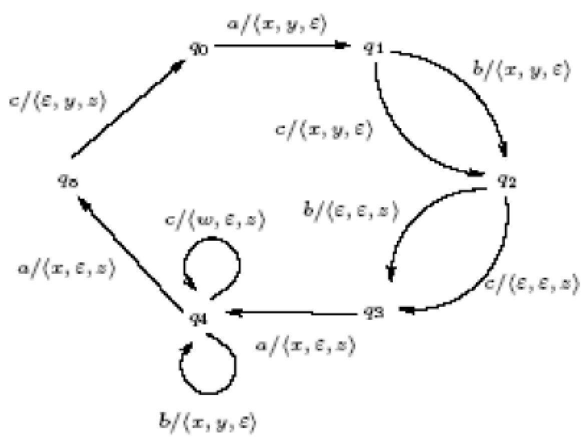
$a/\langle x,y,\varepsilon\rangle$
q_0
q_1
$b/\langle x,y,\varepsilon\rangle$
$c/\langle \varepsilon,y,z\rangle$
$c/\langle x,y,\varepsilon\rangle$
q_5
q_2
$c/\langle w,\varepsilon,z\rangle$
$b/\langle \varepsilon,\varepsilon,z\rangle$
$a/\langle x,\varepsilon,z\rangle$
$c/\langle \varepsilon,\varepsilon,z\rangle$
q_4
q_3
$a/\langle x,\varepsilon,z\rangle$
$b/\langle x,y,\varepsilon\rangle$

Fig. 2 An example of 3p-FSM

A test sequence of an np-FSM automaton is a sequence in the form:$!x_1? y_1!x_2? y_2\ldots! x_t?y_t$ that for i = 1,..,t : $x_i \in \Sigma$, $y_i \subset \bigcup^{n}_{k=1} \Gamma k$ and for each port k $|y_i \cap \Gamma_k| \leq 1$.

-$!x_i$:Denotes sending the message xi to IUT.
-$?y_i$:Denotes the reception of messages belonging to the yi from the IUT.

An example of a test sequence of 3p-FSM illustrated in figure 2 is:

!a?{x.y}!b?{x.y}!c?{z}!a?{x.z}!b?{x.y}!c?{w.z}!a?{x.z} !c?{y.z} **(1)**

Generally, test sequences are generated from the specification of the IUT and characterized by fault coverage. The faults covered by the FSMs methods are: output faults, transfer faults or combination of both of them [9].

2.3The test procedure

The works ([2],[10]) allow generating local test sequences for each tester, thus defining the behavior of the test application in a PCO. In fact, each tester executes a local test sequence, built from the global test sequence of the IUT. A local test sequence has the form α 1α2 α n, where each αi is either:

- !x : message sent (x∈ Σk de IUT)
- ?y : message received (y ∈ Γk de IUT)
- $!C_{h1,\ldots,hr}$: coordination message C sent to testers $h_1,..h_r$.
- $?C_h$: coordination message received from the tester h

For each message αi to send to the IUT or a coordination message, the tester supports the process of sending this message. If αi is a expected message from the IUT or a coordination message, the tester waits for this message. If no message is received, or if the received message is not expected, the tester returns a verdict **Fail** (fail). If the tester reaches the end of its local test sequence, then it gives a verdict **Accept** (accepted). Thus, if all testers return a verdict **Accept**, then the test system ends the test with a global verdict **Accept**.

$$\begin{cases} \omega_1 = !a?x?x?C_3?C_3!a?x?x?w!a?x?C_3 \\ \omega_2 = ?y!b?y!C_3?C_3!b?y!C_3?C_3?y \\ \omega_3 = ?C_2!C_1!c?z!C_1?z!C_2?C_2!c?z?z!C_{\{1,2\}}!c?z \end{cases} \quad (2)$$

2.4 Synchronization Problem

In the distributed test, each tester (PTC) is running its local test sequence produced from the global test sequence of the IUT. Thus, the PTCs are working together but independently, which leads us to manage the problem of synchronization of testers. We will run the first part of each local test sequence w f1, w f2, w f3 from w1, w2, w3, as follows:

$$\begin{cases} \omega_{f1} = !a?x?x?C_3 \\ \omega_{f2} = ?y!b?y!C_3 \\ \omega_{f3} = ?C_2!C_1 \end{cases} \quad (3)$$

Running w_{f1}, w_{f2} and w_{f3} should give the result shown in Figure 3(a) but the execution of our prototype provides an incorrect result given in Figure 3 (b).

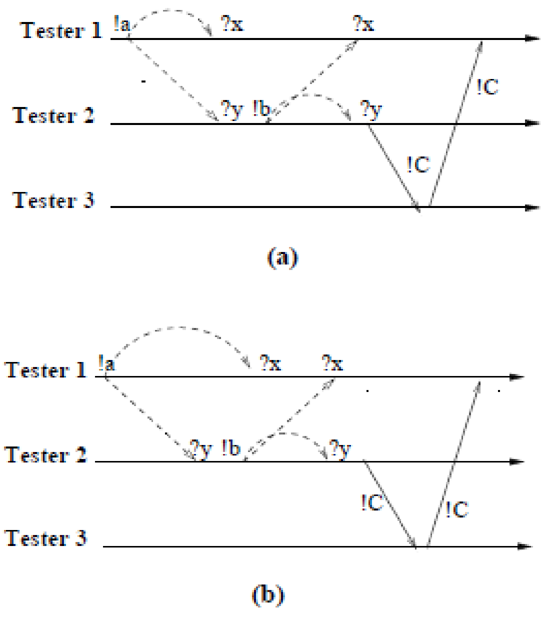

Fig. 3 Example of the synchronization problem

Indeed, in the last diagram the second tester sends the message b the IUT before the first tester receives the message x from the IUT.

So, the execution of local testing is not conform with the specification in (1), where the message 'b' must be sent only if all messages due to the sending of 'a' by the tester-1 are received by the IUT. In [10] the problem is solved by blocking the sending of the message 'b' until the reception of all messages due to the sending of 'a' by the concerned testers. The algorithm [2] applied to the global test sequence (1) generates the local test sequences synchronized for each tester (Figure 2).

$$\left\{\begin{array}{lcl} \omega_1 & = & !a_1^{\{\}}?x_1?x_2?C_3?C_3!a_4^{\{z_3\}}?x_4?x_5?w_6!a_7^{\{w_6,z_6\}}?x_7?C_3 \\ \omega_2 & = & ?y_1!b_2^{\{x_1,y_1\}}?y_2!C_3?C_3!b_5^{\{x_4,y_4\}}?y_5!C_3?C_3?y_8 \\ \omega_3 & = & ?C_2!C_1!c_3^{\{x_2,y_2\}}?z_3!C_1?z_4!C_2?C_2!c_6^{\{x_5,y_5\}}?z_6?z_7 \\ & & !C_{\{1,2\}}!c_8^{\{x_7,z_7\}}?z_8 \end{array}\right.$$

Fig. 2 Local test sequences

The solution proposed in this article integrates mobile agent technology with multi-agent systems for the verification of the receipt of expected messages on different (PCO).

2.5 State of the art

Last years, the rapid growth of distributed systems has led to a need for its coordination. Many frameworks suggest several key issues that will contribute to the success of open distributed systems. The most well known are:

• The Open Group's Distributed Computing Environment (DCE), [11], [12], [13] provides a communication network and middleware platform independent platform for distributed processing systems. It is based on a client /server architecture and does not support object-oriented technologies.

• [14] The OMG Common Object Request Broker Architecture (CORBA) provides an object-oriented framework for distributed computing and mechanisms that support the transparent interaction of objects in a distributed environment. The basis of the architecture is an object request broker (ORB) which provides mechanisms by which objects transparently invoke operations and receive results.

• [15] provides a framework by describing a generic object-oriented architecture that is based on five different viewpoints that enable the description of distributed systems from various perspectives: Enterprise, Information, Computation, Engineering, and Technology viewpoint.

The two last standards integrate the concept of object-oriented architecture The Object-oriented based development of such applications using the "classical" objects is very difficult, like many possible ways of activating or deactivating event sources and to dispatch the call-backs.

• [16] proposes an architecture for fault detection of web services through web services based on passive testing .they proposes an observer (mobile agent) that can be invoked by interested parties when developed and published as a web service. In this model, they have not integrated the concept of Multi-Agent Systems.

• [10] solve the problem of synchronisation by introducing the concept of Mobile Agent but any implementation of the system has be given.

However, in practice, the development of distributed systems is more complex. The environments that support their implementation are unstable and applications must cope with the volatility of resources and services. They must be flexible and be able to adapt dynamically.

While a number of agent-based distributed testing systems have been proposed and the multi-agent systems have been studied, to the best of our knowledge, combining these two technologies has not been applied to this field.

Our approach consists to integrate adaptable mobile agent technology with multi-agent systems to enhance the distributed testing.

3. Mobile Agent Model

3.1 Why the actor model?

Agent technologies are a potentially promising approach for building complex systems that require intelligent behavior from a collection of collaborating, autonomous software components [17]. The architecture that we propose is based on the actor model.

An agent's actors ([6], [7]) integrate the concept of agent and the concept of appointment for the communication. Indeed each agent must create other agents to send messages to the actors whom it knows and to change dynamically behavior by defining its behavior for the next treatment.

The idea of using actors for implementation of distributed systems is not new. However, it seems particularly interesting in the context of distributed systems to large scale by the properties that distinguish the model of actors to the classical object model ([18] ,[19]).

• First, communication by asynchronous messages is well suited to large networks or wireless, where synchronous communications are too difficult and expensive to establish,

• Other, the autonomy of execution and behavioral autonomy promote the integration of the mobility and ensure integrity.

The mobility of actor can be defined based on the treatment of messages in series and behavior change, as well as the dynamic creation and sending messages [7]

3.2 JavAct

JavAct [20] is an environment allowing the development of programs Java competitors and distributed according to the model of the actors. Compared with the traditional mechanisms of low level like the sockets, the RMI and services Web or SOAP, it has a high level of abstraction. It provides primitives of creation of actors, of change of their behaviors, localization, and communication by sending of messages. JavAct has been designed in order to be minimal in terms of code and so maintainable at low cost. Applications run on a domain constituted by a set of places which can dynamically change. A place is a Java virtual machine (Figure 4).
JavAct library provides tools for creating actors, changing their own interface, and also for distribution and mobility, static and dynamic (self-) adaptation, and (local or distant) communications.

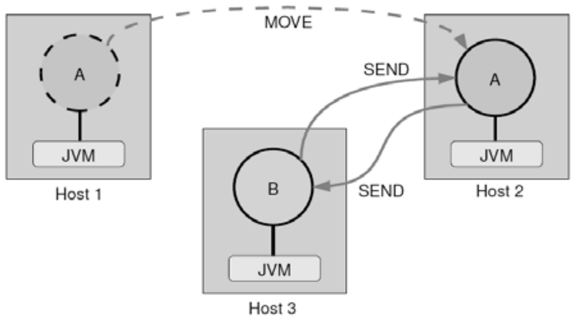

Fig. 4–Illustration of the concept of mobility and Communication on JavAct

3.3 Agent Based Architecture

This section presents our Multi-Agent architecture of distributed testing application. It is made up of a whole of entities (testers) which interact and/or cooperate collectively to give a satisfactory answer to the preset criteria of test. We represent each entity by an agent able to communicate, having knowledge and private behavior as well as its own capacity of execution. Each agent tester must be able not only to store information but also to be able to communicate with other agents testers to confirm the reception of one or more outputs on the PCO where they are attached.

Due to the complex analyzing tasks made by tester for detecting output faults on related IUT PCO, we delegate performed tasks to well defined agents as shown in figure3.

In fact, each Tester Agent has its local test sequence deduced from the global test sequence by the algorithm [2].

The Tester Agent plays the role of moderator in the same tester, and according to its local test sequence, it expresses its needs in terms of synchronization, coordination and of execution to other agents. Taking into consideration the complexity of different tasks to perform, we divided the tasks between three specialized agents:

1. **AMRI(Synchronization Agent):** It is a Mobile Agent for Searching Information to ensure synchronization in the transmission and reception of messages of the test sequences;
2. **AGCTi:** It is a Coordination Agent in Tester 'i' for sending and receiving coordination messages;
3. **AETi**: It is a Execution Agent in Tester 'i' to Control and observation of events on each PCO.

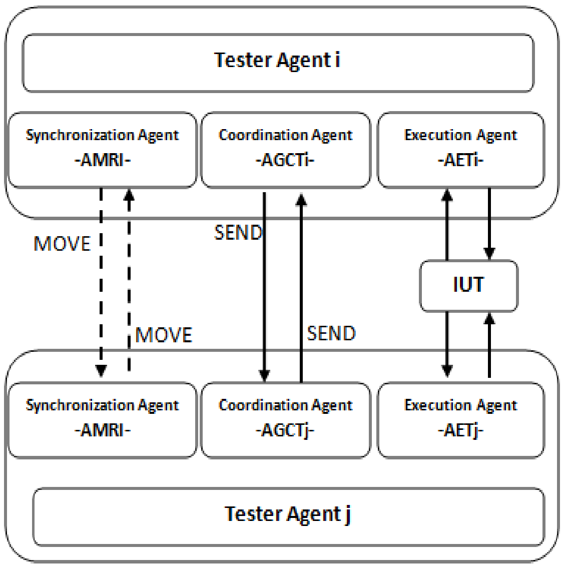
Tester Agent i
Synchronization Agent -AMRI-
Coordination Agent -AGCTi-
Execution Agent -AETi-
MOVE
SEND
IUT
MOVE
SEND
Synchronization Agent -AMRI-
Coordination Agent -AGCTj-
Execution Agent -AETj-
Tester Agent j

Fig. 5– Structure of a tester

4. Distributed Test Model

In this section, we define the behavior agents of our testing distributed testing platform:

1. ***AETs*** : allows tester to apply input event actions to IUT and to receive output results from the IUT.

2. ***AGCTs :*** They make it possible to the testers to exchange messages of coordination. According to the local test sequence of the tester on whom the AGCT is generated, this AGCT will send messages of coordination to the testers concerned while communicating with their AGCTj for i ≠ j. After the reception of a message of coordination, AGCTi will inform tester i to block or continue its execution.

3. ***AMRI:*** It is the only mobile agent of the system. It will be generated by the tester who must make the first sending in the test. Its role is to traverse the testers to observe and collect the messages ?yi received by testers, after each sending of a message !xi. A collected message is put in a box of collection of the AMRI. Once the turn is finished, the AMRI will migrate towards the next tester who has the row of sending of the message !xi+1. It must be provided with List-of-Senders to be able to know the tester who has the row of sending and to migrate towards this tester. The algorithm [2] was generalized to be able to generate the local test sequences and List-Of-Senders.

Algorithm 1.Generating Local test sequences Senders by order of sending

```
    Input $w=!x_1?\, y_1!x_2\ldots..!x_t?\, y_t$ :
        a complete test sequence
    Output : List of senders by order of sending: List-Of-Senders
        Local test sequences: $(w_1,\ldots,w_n)$
for k=1,…,n do $w_k \Leftarrow \varepsilon$ end for
 List-Of-Senders $\Leftarrow \varepsilon$
for i=1,…,t do
  k $\Leftarrow$ Port($x_i$)
  if i >1 then
    Send-To $\Leftarrow$ (Ports($y_i$) $\Delta$ Ports($y_{i-1}$))\{k}
    if sender≠0 then
        Send-To $\Leftarrow$ Send-To\{sender}
    end if
    if sender≠ ∅ then
        $w_k \Leftarrow w_k\, .C_{send\text{-}To}$
        List-Of-Sender <= List-Of-Senders. $T_k^{OSend\text{-}To}$
        For all h ∈ Send-To do
        $w_h \Leftarrow w_h\, .\, +C_k$
        end for
    end if
  end if
1 :       $w_k \Leftarrow w_k\, .\, !x_i$
2 :     List-Of-Sender $\Leftarrow$ List-Of-Senders.$T_k^{xi}$
3 : for all a € $y_i$ do
      $w_{Port(a)} \Leftarrow w_{Port(a)}.a?$
     end for
  if i <t then
    h $\Leftarrow$ Port($x_{i+1}$)
    Sender $\Leftarrow$ 0
    if h ∉ Ports($y_i$) ∪ {k} then
      select
    4 :  case $y_i$= Ø :
          $w_k \Leftarrow w_k\, .\, -C_h$
          $w_h \Leftarrow w_h\, .\, -C_k$
          List-Of-Sender $\Leftarrow$List-Of-Senders.$T_k^{Ch}$
          Sender $\Leftarrow$ h
    5:   case Ports($y_i$)\Ports($y_{i+1}$) ≠ Ø:
        Choose l € Ports($y_i$)\Ports($y_{i+1}$)
          $w_l \Leftarrow w_l\, .\, -C_h$
          $w_h \Leftarrow w_h\, .\, -C_l$
          List-Of-Sender $\Leftarrow$ List-Of-Sender.$T_l^{Ch}$
          Sender $\Leftarrow$ l
   6:   Otherwise :
          Choose l € Ports($y_i$)
          $w_l \Leftarrow w_l\, .\, -C_h$
          $w_h \Leftarrow w_h\, .\, -C_l$
          List-Of-Sender $\Leftarrow$List-Of-Senders.$T_l^{Ch}$
          Sender $\Leftarrow$ l
     end select
    end if
  end if
end for
```

end Algorithm 1

If IUT has n ports, Algorithm 1 is dedicated to compute the n related local test sequences and List-Of-Sender by order of send of n testers from a complete test sequence of IUT. Function Port give the port corresponding to a given message. Function Ports is defined by : Ports(y) = {k/□a □y s.t.k=Port(a))} for a set y of messages .

The local test sequences are basically projections of the complete test sequence over the port alphabets. In fact, Line 1 and 2 adds respectively $!x_i$ to the sequence wport(x_i) and tester port(x_i) to Liste-of-Sender..The Loop of Line 3 adds the reception of messages belonging to yi to the appropriate sequences.

Coordination messages are added to the projections to get the same controllability and observability when using the complete test sequence in centralized method. In this case, we added ?C et !C to the appropriate local test sequences.?C is added to wh and List-of-sender, where h is the tester sending x_{i+1}.!C is added to the sequence of a tester receiving a message belonging to y_i,if $y_i \neq Ø$(Lines 5 and 6),if not !C is added to the sequence of the tester sending xi(Line4).

4. ***Agents Testers:*** Each tester executes its local test sequence in the following way:

   - ***For a communication with the IUT***

- If the message is a reception, the tester waits until its AET informs it of the reception.

- If the message is a sending, the tester awaits the arrival of the AMRI and tests:

1- If the sending is like $!M^{\{\}}$: the Agent tester Inform Execution tester to sends the message and sends the AMRI to collect the messages which will be observed following this sending. Before each new turn, box of collection of the AMRI is initialized.

2- If the sending is like $!M^{\{xi-1,yi-1\}}$ (the message could be sent only if the messages x_{i-1} and y_{i-1} were well observed) the tester will search among information collected by the AMRI during the last turn:

a- If x_{i-1}, y_{i-1} is in box of collection of the AMRI: the tester sends the message and initializes the AMRI for a new turn.

b- If xi-1 and yi-1 do not exist in box of collection. The tester will be able to return the AMRI to check the reception of these messages by the testers concerned.

   - ***For a communication with other testers (Messages of coordination)***

- If the message is a reception, the tester waits until its AGCT informs it of the reception.

- If the message is a sending, the tester informs its AGCT which will communicate with the AGCT of the tester who must receive this message.

5. Conclusion

This article, presents architecture, model and method of distributed test guaranteeing the principles of coordination and synchronization between the various components of the application of distributed testing. We exploited the concepts of mobiles agents and actors agents which make it possible to propose software architectures able to support the dynamic adaptation and to reduce the number of messages between the various components of the distributed test. The implementation , the introduction of the notion of time into the test sequences and the test of the applications like Web services are the prospects for our approach.

References

[1] M.Benattou and J.M. Bruel, " Active Objects for Coordination in Distributed Testing, " International conference on object-oriented information systems, Montpellier, FRANCE, 2002.
[2] O. Rafiq, L. Cacciari, M. Benattou, "Coordination Issues in Distributed Testing", Proceeding of the fifth International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99) pp: 793-799, USA, 1999, CSREA Press .
[3] A.Khoumsi , " A Temporal Approach for Testing Distributed Systems, " IEEE Transactions on Software Engineering, " vol .28 no.11, pp.1085-1103, November 2002.
[4] B.Chena, H. Chengb, J.Palen, Integrating mobile agent technology with multi-agent systems for distributed traffic detection and management systems",Transportation Research Part C: Emerging Technologies Volume 17, Issue 1, February 2009, pp.1-10.
[5] G. Bernard , L. Ismail , "Apport des agents mobiles à l'exécution répartie". Technique et science informatiques, Hermès, Paris, 2002, Vol. 21, No 6/2002, pp. 771-796.

[6] G. Agha : Actors : a model of concurrent computation in distributed systems.M.I.T. Press, Cambridge, Ma., 1986.

[7] J.-P. Arcangeli, L. Bray, A. Marcoux, C. Maurel et F. Migeon : Réflexivité pour la mobilité dans les systèmes d'acteurs. In 4ème École d'Informatique des SYstèmes PArallèles et Répartis (ISPAR'2000), 2000.

[8] A. Gill, ,Introduction to the theory of finite-state machines, Mc Graw-Hill, New Yor- USA, 1962.

[9] A. Petrenko, G. v. Bochmann and M. Yayo, On fault coverage of tests for finite state specification, Computer Network and ISDN Systems 29, 1996.

[10] M. Benattou, "A Multi-Agent Based Architecture For Distributed Testing," proceeding of The 18th International Conference on Software Engineering and Knowledge Engineering, San Francisco, pp : 495-498, 2006 .

[11] Digital Equipment Corporation "Distributed Computing Environment Application Development Reference", Maynard, Maryland, U.S.A.1992.

[12] Lockhart H.W. OSF DCE – Guide to Developing Distributed Applications. McGraw-Hill, New York, U.S.A.1994.

[13] A.Schill DCE - das OSF Distributed Computing Environment, Einführung und Grundlagen. Springer Verlag 1996.

[14] Object Management Group (1995)," The Common Object Request Broker: Architecture and Specification, "Revision 2.6. Framingham, Massachusetts, U.S.A, December 2001.

[15] G.S.Blair and J.-B.Stefani. Open distributed processing and multimedia.Boston MA USA: Addison Wesley Longman ,1998 .

[16] A.Benharref,R.Glitho,R.Dssouli. "Mobile Agents for Testing Web Services in Next Generation Networks", Mobility Aware Technologies and Applications, Vol.3744 ,2005, pp.182-191.

[17] Gorton, I., J. Haack, D. McGee, A. J. Cowell, O.Kuchar and J. Thomson. Evaluating Agent Architectures: Cougaar, Aglets and AAA. , SELMAS:264-278. 2003.

[18] S.Leriche ," Architectures à composants et agents pour laconception d'applications réparties adaptables ",J.P Arcangeli, ingénierie des Langages pour les sYstèmes Répartis et Embarqués (LYRE), Informatique et Télécommunications (EDIT), Université Toulouse III,Toulouse ,France,2006 .

[19]Sébastien Leriche and Jean-Paul Arcangeli. Flexible Architectures of Adaptive Agents: the Agent approach. Dans / In : International Journal of Grid Computing and Multi-Agent Systems, ISSN : 0975-8135, Vol. 1, No. 1, p. 51-71, January-June 2010

[20] J.-P. Arcangeli, F. Migeon, S. Rougemaille, " JAVACT : a Java middleware for mobile adaptive agents", Lab.IRIT,University of Toulouse, *February 5th, 2008,*france .