Journal of Theoretical and Applied Information Technology
15th December 2012. Vol. 46 No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

TEST EXECUTION CONTROL WITH TIMING CONSTRAINTS FOR TESTING DISTRIBUTED SYSTEMS

SALMA AZZOUZI, MOHAMMED BENATTOU, HASSAN CHARAF
Laboratory of Research in Computer Science and Telecommunication
Faculty of Science Ibn Tofail University
Kenitra, Morocco

ABSTRACT

The development of distributed testing frameworks is more complex, where the implementation process must consider the mechanisms and functions required to support interaction as long as the communication and the coordination between distributed testing components. The typical reactions of such systems are the generation of errors 'set: time outs, locks, observability, controllability and synchronization problems. The first contribution in this study present a way to control the test execution of distributed testing components by introducing the synchronization messages and we show how the problems of control and synchronization can be solved by the same process. In other side, we show that in practice the distributed testing process must not only check if the exchanged events have been observed, but also the dates when these events have been occurred and then the distributed testing frameworks must consider some timing constraints.

Keywords: *Distributed testing; Controllability, Observability; Synchronization; Timing Constraints.*

1. INTRODUCTION

The principle of testing is to apply input events to the implementation under test and compare the observed output events with expected results. Conformance testing may be seen as mean to execute an IUT1 by carrying out test cases, in order to observe whether the behavior of the implementation is conforming to its specification. In the context of distributed systems the IUT may be viewed as a system providing standardized interfaces for interacting with other systems. Based on testing of OSI communicating systems, conformance of an open distributed system can be assessed by attaching a related tester at each provided interface [1]. However, many problems influencing faults detection arise during the conformance testing process if there is no coordination between distributed testers. In fact, the use of multiple testers introduces the possibility of coordination problems amongst remote testers ([1, 2, 3, 4, 5]). These potential problems are known as controllability and observability fault detections which are fundamental features of conformance distributed testing. In this context, most related research works propose to coordinate the distributed testers by using a communication service parallel to the IUT through a multicast channel.

Another problem is due to the implementation of these communication channels. In fact, many time-outs problems arise during the test execution which influences significantly the fault detection. These problems called Synchronization issues has been resolved in [2] by combining mobile agent technology and Multi-Agent System. However, our preliminary experience in the implementation of the mobile agent solution shows that the movements of the mobile agents are complex to manage. So, our first contribution in this article is to propose another way to avoid these problems by introducing the synchronization message in the local test sequences LTS (the LTS determine when a tester can apply its own inputs and whether an output observed is received in response to the correct input). In first sight, it appears that the solution will increase the messages communicated between different components of the test architecture. Conversely, we proof in this paper how these Synchronization messages can resolve both of synchronization and coordination problems and then eliminate coordination messages.

In other side, the introduction of coordination messages leads each tester to determine when to apply a particular input to the IUT and whether a correct output from the IUT is generated in

[1] **I**mplementation **U**nder **T**est

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

response to a specific input, respectively. The distributed testing process must not only check if the output events have been observed, but also the dates when these events have been occurred especially if the system has to respect some timing constraints. In this context, two different time constraints must be considered: transfer time i.e. the time required for a coordination message to travel from a tester to another, and reaction time, i.e. the time elapsed between the reception of an input by the IUT and the sending of the corresponding output by the IUT. Many academic researches [2, 1, 3] made a simplifying assumption that the time required for a transfer time, is greater than the reaction time of the IUT. [6, 7] showed that controllability and observability problems are indeed resolved if and only if the test system observes those timing constraints. In this context, we determine in [29] timing conditions that guarantee communication between components of distributed testing architecture and we propose our Multi-Agent architecture for testing these systems.

The second contribution in this paper presents some technical issues for testing distributed frameworks with timing constraints. The proposed approach consists firstly on introducing a new architecture taking into account the delay of messages exchanged between testers and the IUT, and between testers. The main based idea of the proposed work is to develop an algorithm for generating Timing Local Test Sequences for each tester guarantying to avoid problems of coordination, observation and synchronization.

The paper is structured as follows: Section 2 describes the architecture and some modeling concepts of distributed testing application and presents the synchronization problems arisen in distributed testing execution. section 3 raises and solves synchronization and controllability problems in distributed testing implementation. Section 4 is dedicated to introduce the architecture and modeling concepts of testing distributed applications with timing constraints. Section 5 presents the algorithm allowing the generation of timing local test sequences, and finally section 6 gives some conclusions and identifies future works.

2. DISTRIBUTED TESTING

The principle testing is to apply input events to the IUT 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. Conformance testing may be seen as mean to execute an IUT by carrying out test cases, in order to observe whether the behavior of the implementation is conforming to its specification.

A. ARCHITECTURE

The basic idea of distributed testing 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 PCO2, and communicates with other testers through a multicast channel (Fig1).

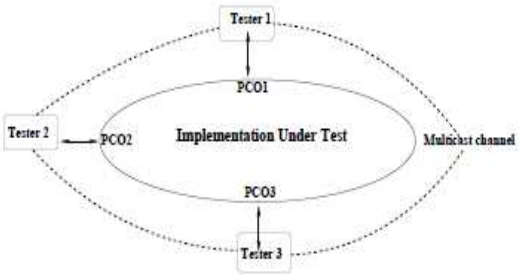

Fig. 1. Test Architecture

An IUT 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 attached interfaces called PCOs (Points of Control and Observations) and from which it observes the output IUT 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. In order to control the test execution, PTCs exchange messages that encapsulate the information avoiding controllability and observability problems.

B. TEST PROCEDURE

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 system is described by an automaton with n-port [8] (FSM Finite State

[2] **P**oint of **C**ontrol and **O**bservation

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Machine) defining inputs and the results expected for each PCO.

We denote Σ_k the input alphabet of the port "k" (PCO number k) and Γ_k the output alphabet of the port k. Fig.2 gives an example of 3p-FSM with Q = {q_0, q_1, q_2}, q_0 initial state, Σ_1 = {a_1 ,a_2}, Σ_2 = {b_1}, Σ_3 = {c_1}, and $\Gamma 1$ = {x_1, x_2 }, Γ_2 = { y_1,y_2}, Γ_3 = {z_1}.

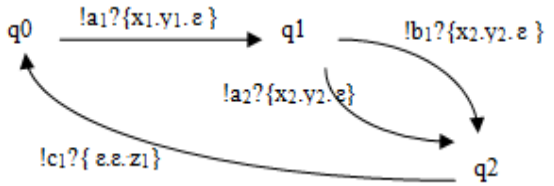

Fig. 2. *An Example Of 3p-FSM*

A test sequence of np-FSM is a sequence in the form:$!X_1?\ Y_1!X_2\ ?\ Y_2\ldots!\ X_t\ ?Y_t$ where for i = 1,..,t, X_i belongs to $\Sigma = \Sigma_1 U \ldots U\ \Sigma_n$ with $\Sigma_i \cap \Sigma_j = \emptyset$ for $i \neq j$ and Y_i is a subset of $\cup^{n}_{k=1} \Gamma_k$ such that , for each port k, $|Y_i \cap \Gamma_k| \leq 1$, i.e Y_i contains at most one symbol from the output alphabet of each port of ***A***.

- $!X_i$: Denotes sending the message Xi to IUT.
- $?Y_i$: Denotes the reception of messages belonging to the Yi from the IUT

An example of global test sequence (GTS) deduced from the 3p-FSM given in Fig.2 is:

$$!a_1?\{x_1.y_1\}!b_1?\{x_2.y_2\}!c_1?\{z_1\} \quad (1)$$

Generally, test sequences are generated from the specification of the IUT and characterized by fault coverage. Several methods exist for generating test sequence from FSM specification. They are mainly used for detecting two basic types of faults output faults and transfer faults [9].

The work [4] allows generating local test sequences for each tester, and thus the behavior of the test application in each PCO is well defined. In fact, each tester executes its local test sequence (LTS), built from the global test sequence of the IUT. The generated LTS encapsulate the information that allows controlling the test execution. Indeed, many problems influencing faults detection during the conformance testing process arises if there is no coordination between distributed testers. These potential problems are known as controllability and observability fault detections which are fundamental features of conformance distributed testing.

The controllability may be defined as the capability of the test system to realize input events at corresponding PCO in a given order, and observability may be defined as the capability of the test system to determine the output events and the order in which they take place at corresponding PCO [4].

To solve such problems, authors in [4] propose an algorithm to generate Local Test Sequences (LTS) from Global Test Sequence (GTS). The following LTS are the results given by applying the proposed algorithm to test sequence (1):

$$\begin{cases} W_1 = !a_1\ ?x_1?x_2?O^3 \\ W_2 = ?y_1\ !b_1?y_2!C^3 \\ W_3 = ?C^2!O^1!c_1?z_1 \end{cases} \quad (2)$$

Where:

- !x denote sending of message x to IUT
- ?y denote receiving of message y from the IUT
- $!C^k$ denote sending coordination message to tester k and $?C^k$ receiving coordination message from tester k.
- $!O^k$ denote sending observation message to tester k and $?O^k$ receiving observation message from tester k.

In distributed testing method, each tester executes its LTS as follows: for each message "x_i" sent to the IUT or a coordination message, the tester supports the process of sending this message. If "x_i" is an 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**.

C. SYNCHRONIZATION PROBLEMS

In the distributed test, each tester (PTC) executes its local test sequence produced from the global test sequence of the IUT. Lets the execution of the each local test sequence (2) W_1, W_2 and W_3 as follows:

The execution of local test sequences (2) must give the result shown in Fig.3(a) but the execution of our prototype provides an incorrect result given in Fig.3 (b).

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

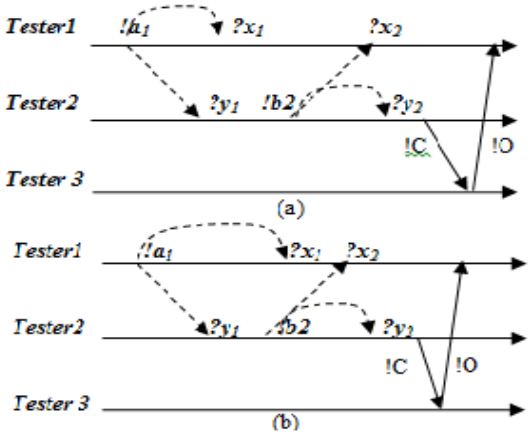

Fig. 3. Example Of The Synchronization Problem

Indeed, in the last diagram Fig.3 (b) the second tester sends the message "b_2" to the IUT before the first tester receives the message "x_1" from the IUT.

So, the execution of local testing is not conform with the specification gives in (1), where the message "b_2" must be sent only if all messages due to the sending of "a1"by the tester-1 are received by the IUT.

The solution [10] proposes to integrate the mobile agent technology for checking the well receipt of expected messages on different (PCO). However, the management of the mobile agent movements makes the testing process more complex to implement. We think that the deployment of mobile agent technology in distributed testing must make some mechanisms where the mobility is more restricted.

To this end, we propose to integrate some synchronization messages to build our local test sequences from the global sequences.

3. RELATED WORKS

Recently, the rapid growth of distributed systems has led to made specific reflections about its coordination. Many frameworks suggest several key issues that will contribute to the success of open distributed systems [11, 12,13,14] and many works has been made to avoid the coordination problems previously explained of testing such frameworks.

In this context, the author in [6] shows that controllability and observability are indeed resolved if and only if the test system respects some timing constraints and he proposes a centralized architecture for distributed testing. In this context, we determine in [29] timing conditions that guarantee communication between components of distributed testing architecture. Another work [15] shows that the use of coordination messages can introduce delays and this can cause problems especially if there are timing constraints.

The work [16] proposes a new method to generate a test sequence using multiple unique input/output (UIO) sequences. The method is essentially guided by the way of minimizing the use of external coordination messages and input/output operations.

In [17], the authors suggest to build a test or checking sequence from the specification of the system under test such that it is free from these problems without requiring the use of external coordination messages. In this context, they propose some algorithms for generating subsequences that eliminate the need for external coordination messages.

The basic idea in [18], [19], [20] is to build a test sequence that causes no coordination problems during its application in a distributed test architecture. For some specifications, such test sequence exists where the coordination is achieved via their interactions with the IUT. However, it is not always true as detailed in [21].

The emphasis of recent works is to minimize the use of external message exchanges among testers [20] or to identify conditions on a given FSM under which the problems in distributed testing can be overcome without using external coordination messages [22, 23].

The work presented in [24] proposes fault detection architecture through web services based on passive testing. They propose an observer (mobile agent) that can be invoked by interested parties when developed and published as a web service. In their model, they don't integrate the concept of Multi-Agent Systems.

Finally, we suggest in a previous article [10] to solve the synchronization problem by introducing architecture combining both concepts of Mobile Agent and MAS (multi-agent system).

Our approach in this paper consists on introducing a new architecture allowing to avoid the synchronization problems by considering the delay of messages exchanged between: (i) testers and the IUT and (ii) between testers.

4. TEST CONTROL

The idea of introducing synchronization messages in the local test sequence appears, at the

Journal of Theoretical and Applied Information Technology
15th December 2012. Vol. 46 No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

first sight, that the solution will increase the messages communicated between different components of the test architecture, but they will eliminate coordination messages. We proof, in this section, how the addition of synchronization messages solves both problems of synchronization and coordination:

A. SYNCHRONIZATION MODEL

We will introduce in this section, some synchronization messages in the Local Test Sequences as shown in Fig4. In fact, all the testers receiving a message belonging to y_i (set of messages received due to sending x_i to the IUT), are going to send a synchronization message to the tester that will send the message x_{i+1}.

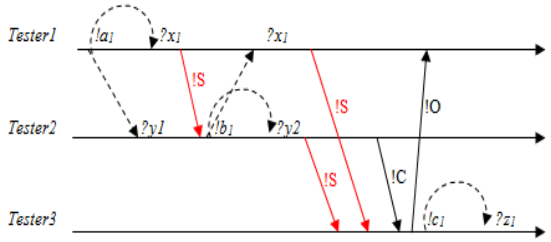

Fig. 4. Synchronization Message Introduced In LTS To Resolve Synchronization Problem

Example: While sending "$!a_1$" by the tester1, a message "$?x_1$"(resp. "$?y_1$".) will be received by the tester1 (resp. tester2). Then, for sending "$!b_2$" by the tester2, it will be sure that the previous messages are well received by the concerned testers.

The problem is in the verification of the reception of "$?x_1$" in the tester1. To this end, once the message "$?x_1$" is received by tester1, it send a synchronization message to tester2 to inform this reception.

The introduction of synchronization messages means that all testers receiving a message belonging to yi, will send a synchronization message to the tester "h" sending x_{i+1}.

Let k ⇐Port($!x_i$) be the port of the tester sending xi and h ⇐ Port($!x_{i+1}$) the port of the one sending x_{i+1} and let y_i (resp y_{i+1}) a set of outputs sent by the IUT in response to the reception of the input xi (resp x_{i+1}).

The function Port gives the port corresponding to a given message and we will define the function Ports as: Ports(y) ={k/∃a ∈y : k=Port(a))} for a set y of messages.

Definition1: we define testers' senders Synchronization messages to tester "h" after sending !xi by:

$$\text{Senders}^{S}i = \begin{cases} \textbf{Singleton } \{k\}, \text{ if } y_i = \emptyset; \\ \text{Ports}(y_i) \text{ , otherwise} \end{cases}$$

We define the set S_i= $\{S_1,...,S_k\}$ with k=1... N-1, as the set containing all the Synchronization messages sent by Sender^{S}_{i}.

The synchronization messages, such as previously defined, introduced in LTS (1) gives the new LTS given in (3):

$$\begin{cases} W_1 = !a_1\,?x_1!S^2?x_2!S^3\boldsymbol{?C^3?S^3}?O^3 \\ W_2 = ?y_1?S_1\,!b_1?y_2\boldsymbol{!S^3!C^3} \\ W_3 = \boldsymbol{?C^2}!O^1?S^1\boldsymbol{?S^2}!c_1?z_1 \end{cases} \quad (3)$$

We deduce from the above result that the number of message communicated between testers is increase. Another interesting remark can be deduced from the same result, that each control message is accompanied by a synchronization message (the messages of control and synchronization are marked in bold). Intuitively this means that the coordination problem is embedded with the synchronization one. To proof this, we define formally, in the next section, the notion of control.

B. DEMONSTRATION

In this section we show that it is possible to reduce the number of messages in LTS (3) by removing all coordination messages. Notice that control problem may arise when the tester sending a message x_{i+1} is neither the one sending xi, nor one of those receiving a message belonging to y_i , we add then a coordination message (!C) to the sequence of a tester "l" receiving a message belonging to yi and (?C) to a tester "h" that send the message x_{i+1} . Then formally, we have:

Definition2: the port sender the coordination message to tester "h" after sending $!x_i$, is defined by:

We denote Senders= Ports(y_i)\Ports(y_{i+1}) and if h ∉ Ports(y_i)∪{k} then:

$$\text{Sender}^{C}_{i} = \begin{cases} \textbf{k}, \quad \text{if } y_i = \emptyset; \\ l \in \text{Senders, if Senders} \neq \emptyset \text{ and } y_i \neq \emptyset; \\ l \in \text{Ports}(y_i), \text{ if Senders} = \emptyset \text{ and } y_i \neq \emptyset; \end{cases}$$

We denote Ci the Coordination message sent by Sender^{C}_{i}.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Proposition: let be $\mathbf{C=\{C_1, C_2,..... C_l\}}$ **, l=1..t** the set of all Coordination messages sent at the end of the testing process and $\mathbf{S = S_1 \cup S_2 \cup ... \cup S_p}$**, l=1..t** the set of all Synchronization messages sent after reaching the end of the test, then we have :

$$\mathbf{C} \subset S \qquad \text{(Eq 1)}$$

Proof: The Analyze given from the definition1 show that if yi ≠Ø then SenderCi ϵ **Ports(yi)** and since **Ports(yi)** ⊂ **SendersSi** (definition 2) then therefore in this case(yi ≠Ø) we have Ci ϵ Si (because $\mathbf{Sender^{C}_i \in Senders^{S}i}$) . Moreover since $\mathbf{Sender^{C}i=k= Senders^{S}i}$, if yi =Ø , we claim that after each sending $!x_i$ belongs to Σ, with i=1,..,t. and whatever yi :

$$\mathbf{Sender^{C}_i \in Senders^{S}i} \qquad \text{(a)}$$

Therefore

$$Ci \in Si \qquad \text{(b)}$$

Indeed, for any Ci ϵ C we have by (b) $C_i \in Si$ and Si ⊂ S .Thus, C ⊂ S and we are done.

The proposition above implies that the notion of coordination is combined with synchronization notion. Thus, Synchronization messages embedded in the local test sequences solve both problems of coordination and synchronization.

In this context, we propose in [30] an algorithm that generates Synchronized local test sequences related to n testers from a Global test sequence GTS of the IUT. Applying Algorithm [30] to the Global Test Sequence (1), we get the following sequences:

$$\begin{cases} W_1 = !a_1\, ?x_1 !S^2 ?x_2 !S^3 ?O^3 \\ W_2 = ?y_1 ?S^1\, !b_1 ?y_2 !S^3 \\ W_3 = !O^1 ?S^1 ?S^2 !c_1 ?z_1 \end{cases} \qquad (4)$$

By removing all controllability messages from the local test sequences (3), we will obtain the same LTS (4) generated by Algorithm [30].

C. TIME PROBLEM

The introduction of Synchronisation and Observation messages in the local test sequences leads each tester to determine when to apply a particular input to the IUT and whether a correct output from the IUT is generated in response to a specific input, respectively. However, the distributed testing process must not only checks if the output events have been observed, but also the dates when these events have been occurred especially if the system has to respect some timing constraints. For example, the execution of the first fragment of the GTS given in (1): $!a_1?\{x_1.y_1\}!b_1?\{x_2.y_2\}$, the tester-1 begins by sending a message "$!a_1$" to the IUT. However, the tester-2 can't send the message "$!b_1$" and must wait until receiving the message "$?y_1$" from the IUT and the message "$?x_1$" must be received by the tester-1.

To do this, we integrate Synchronization message for the verification of the reception of expected messages on different (PCO).

Now, the principal question that can be studied and discussed is how much time the tester-2 and tester-1 can wait for receiving "$?y_1$" and "$?x_1$" respectively, so that the tester-2 can send "$!b_2$" to the IUT?

5. TIME MODEL

As we have shown above the distributed testing process must not only checks if the output events have been observed, but also the dates when these events have been occurred. This section is dedicated to extend results from testing distributed system to deal with testing an implementation under test with some timing constraints.

A. ARCHITECTURE

The new proposed architecture will operate in an environment with some timing constraints. In this context, it is not sufficient to check if the IUT produces the correct outputs "$?y_i$" but it should also check if the timings of outputs are corrects. Moreover, the timing of these outputs "$?y_i$" depends on the timing of the inputs "$!x_i$". Indeed, any message sending by a tester to the IUT must be blocked as long as all output events, caused by the last sending message, have been received by all related testers. In other words, the date of the sending inputs "$!x_{i+1}$" to IUT depends on the dates of the receiving the outputs "?yi" by related testers.

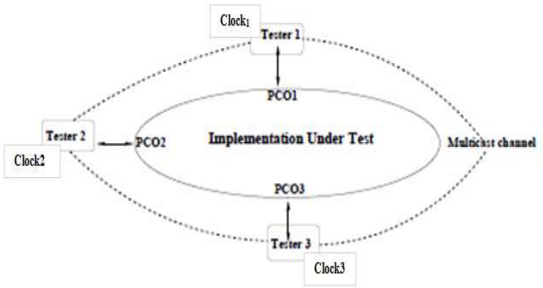

Fig. 5. Test Architecture Of Timing Model

In order to analyze all these timing constraints, we consider that each tester has a clock that compute the delay of messages exchanged between a tester and the IUT ("!xi" and "?yi") and between

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

testers("!/? S/O"). We add then, a set of clocks to model the temporal behavior of the test process and by the way, each port of the distributed system has a corresponding local tester with a local clock.

However, the clocks synchronization among different testers is one of the difficulties to overcome especially when the temporal constraints must be considered in the testing process.

In the aim to check timings constraints in distributed testing correctly, all testers' clocks should be synchronized. In other words, all testers must have the same time reference.

For this purpose, we suggest to deal with the clock synchronization similarly to IEEE 1588 Precision Time Protocol (PTP). It is a new synchronization standard with very high accuracy and particularly proposed for embedded industrial communication systems. PTP provides a mean for networked computer systems to agree on a master clock reference time and a mean for slave clocks to estimate their offset from master clock time [25].

B. FORMAL MODEL

In this section, we propose a formal model to specify the temporal behavior of the distributed testing system. We extend the definition of a timed automaton with n-port to define timing constraint for inputs and the expected result at each PCO in distributed testing model. Timed Automata with n ports is generalized from Timed Automata [26]. A set of clocks and Canonical Enabling Conditions are used to model the temporal behavior of the system. We introduce below some definitions related to Timed Automata with n ports.

***Definition 1**: Timed Automaton with n ports* named as np-TA is defined by $A = (Q, q_0, Act, X, Tr)$ with :

- Q is a finite set of locations;
- q0 ϵ Q is the initial location;
- Act = $\Sigma \cup \Gamma$
  - (i) $\Sigma = \{\Sigma_1, \Sigma_2, .., \Sigma_n\}$ where Σ_i is a finite set of inputs of port i , $\Sigma_i \cap \Sigma_j = \emptyset$ for $i \neq j$ and i,j = 1,2,…,n and $\Sigma = \Sigma_1 \cup \ldots \cup \Sigma_n$
  - *(ii)* $\Gamma = \{\Gamma_1, \Gamma_2, .., \Gamma_n\}$ where Γ_i is a finite set of outputs of port i, $\Gamma_i \cap \Gamma_j = \emptyset$ for $i \neq j$ and i,j = 1,2,…,n and $\Gamma = \Gamma_1 \cup \ldots \cup \Gamma_n$
- $X = \{x_1, x_2, \ldots, x_n\}$ is a finite set of clocks
- Tr is a finite set of transitions.

Definiton 2: The transition is a tuple $(q_1, \gamma, \sigma, r, q_2)$, where:

- $q_1, q_2 \in Q$ are the source and destination locations;
- $\gamma \in G$ is the guard, a conjunction of constraints of the form x~c, where $G = \{\Lambda\ x \sim c \mid x \in X$ and $c \in N$ and $\sim \in [<, \leq, =, >, \geq]\}$;
- $r \in 2^X$ is a set of clocks to reset to zero, called Rest of Tr;
- σ is the reception of an input x (figured as $?x_i$) or sending of an output y (figured as $!y_j$).

$Tr \subset Q \times G \times Act \times 2^X \times Q$.

We note that the clocks in *X* are viewed as a continuous time clock. Continuous time is a real variable that evolves indefinitely and its derivative with respect to time is equal to 1[27]. Each clock's value can be reset at any instant. A transition Tr can be executed if and only if the guard is verified (True) and the clocks in "r" are reset after the execution of Tr.

The use of the concept of an **np-TA** as defined above will make the temporal behavior of the system to be modeled. The np-TA allows also to model constraints on delays between events of a given system. To this end, we introduce a clock "c" to specify that a delay between two transitions Tr_1 and Tr_2 will be in the range of T=[Tmin, Tmax]. We define then the reset of Tr_1 as {c} and the guard of Tr_2 as **c ~T** with $c \sim_1 Tmin \Lambda\ c \sim_2 Tmax$ and $\sim_1 \in [=, >, \geq]$ and $\sim_2 \in [<, \leq, =]$. As shown in Fig.5, the new automaton modeling by 3p-TA (b) extends the 3p-FSM (a) by adding new states and integrating temporal constraints.

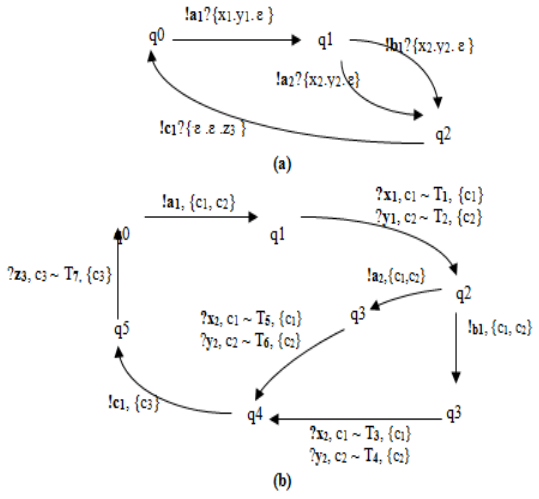

Fig. 6. Example Of A Specification Modeling By 3p-TA

Literally, this new specification requires that for the transition $!a_1?\{x_1.y_1\}$ of (Fig.5.a) if we send

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

"a"$_1$ on port 1 then "x_1" must be received on port 1 between T_{1min} and T_{1max} and "y_2" on port 2 between T_{2min} and T_{2max} (Fig.5.b) otherwise the behavior is not specified (in our context , this case cannot be considered) . We remind that $c_i \sim T_i$ means $c_i \sim_1 T_i min \Lambda\ c_i \sim_2 T_i max$ with $\sim_1 \in [=,>, \geq]$ and $\sim_2 \in [<, \leq, =]$.

C. TIMING CONSTRAINTS

This subsection presents some characteristics of the proposed architecture as well as the time consumed when messages are exchanged between the components of our distributed test system. In the new architecture (Fig 4), temporal constraints should be satisfied. As mentioned before, the correctness of testing distributed systems depends not only on the logical result of a computation, but also on the time when the result was delivered.

For this purpose, we define two types of temporal constraints to be checked in distributed test approach: timing constraints on inter-port and intra-port level respectively. The intra-port timing constraints occur when communication is established between a tester and the IUT, it could be the reaction time required for a tester receiving a message belonging to "y_i" in response to the reception of the input "x_i" by the IUT. And thus, the inter-port timing constraints may be the transfer time required when testers communicate on different ports.

D. INTRA-PORT TIMING CONSTRAINTS

In this subsection we consider only different Intra-port timing constraints when communication is established between tester and IUT. Fig.6 shown how the first part of the GTS (1) $f_{gts} = !a_1?\{x_1.y_1\}$ can be executed and how the time required to execute each message in f_{gts} can be treated. c_1 and c_2 clocks are used to compute the reaction time of "x1" and "y1".

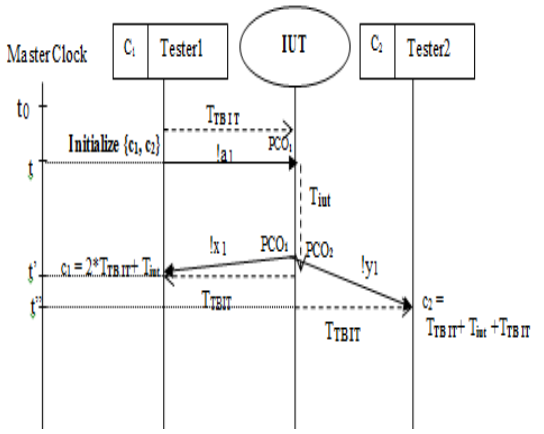

Fig. 7. Transfer Time Required For Receiving Outputs

Where the different computing times used in the above figure are defined as follow:

1- T_{TBIT}: Transfer time between the IUT and the Tester is the time separating: (i) the instant when a Message M is sent by the IUT (resp. the tester) and (ii) the instant when M is received by the Tester (resp. IUT).

2- Tiut: the reaction time of the IUT is an upper bound of the time separating : (i) any instant when an event e is received by the IUT and (ii) the instant when the IUT has terminated to send all the outputs (if any) in response to the reception of e. We emphasize the word "all" because the definition includes possible unexpected outputs (in the case of a non-conformant IUT) [6,7].

3- Time Out is the waiting time that a tester can wait for receiving a message. In case where this time is elapsed the test system should return Failed.

4- Master Clock provides the reference time for all clocks in the testing system.

Therefore, as shown in Fig.6,Intra-port timing constraints can be presented as follows:

- If the message is a sending message "$!x_i$", then there is no time constraint to verify, the message will be sent and we initialize all clocks of testers that should receive messages due to this sent $?(y_i)$.
- If the message is a reception ?a ($a \in y_i$): The guard (denoted ζ_1) must checks that the reception time c measured by the $clock_{port(a)}$ is $T_{TBIT\ (1)}$+ Tiut +$T_{TBIT\ (2)}$ <= c <= Time Out (5)

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

(a) $T_{TBIT\,(1)}$ denote the transfer time between the tester sending "!Xi" and the IUT.

(b) T_{iut} denote the reaction time of the IUT.

(c) $T_{TBIT\,(2)}$ denote the transfer time between IUT and the tester receiving "?a" that be considered as the same time $T_{TBIT\,(1)}$

Formally c verify ζ_1 (denoted $c \models \zeta_1$) when:

$$c \models \zeta_1 \Rightarrow c \sim Tc \Rightarrow c \sim_1 Tcmin \wedge c \sim_2 Tcmax$$

with $\sim_1 \in [=,>,\geq]$ and $\sim_2 \in [<,\leq=]$ and :

$$Tcmin = 2 * T_{TBIT} + Tiut \quad (6)$$

$$Tcmax = Time\ Out \quad (7)$$

E. A.INTER-PORT TIMING CONSTRAINTS

In the IUT, Testers exchange Coordination and Observability messages. In this context, we consider two following cases:

- If tester-i sends a message "!C^j "or " !O^j " to tester-**j** : There is no constraint time to verify, the message is sent and we initialize all clock $clock_i$ of the testers that will receive this message.
- If tester-**j** receives the message "?C^i**"** or "?O^i**"** from tester-**i**:The guard (denoted ζ_2) must check if the time c quantified by the $clock_i$ is as:

$$T_{TBTT} <= c =< Time\ Out \quad (8)$$

$\mathbf{T}_{TBTT}$ is the transfer time between tester-i and Tester-j. T_{TBTT} is defined by the time separating : (i) the instant when a "O" or "S" (Observation /Synchronization message) is sent by the tester and (ii) the instant when "O" or "S" is received by another Tester. Formally c verify ζ_2 (denoted $c \models \zeta_2$) when:

$$c \models \zeta_2 \Rightarrow c \sim Tc \Rightarrow c \sim_1 Tcmin \wedge c \sim_2 Tcmax$$

with $\sim_1 \in [=,>,\geq]$ and $\sim_2 \in [<,\leq=]$ and :

$$Tcmin = T_{TBTT} \quad (9)$$

$$Tcmax = Time\ Out \quad (10)$$

F. B.TIMED TEST SEQUENCE GENERATION

In distributed test method, each tester executes its local test sequence generated from the complete test sequence. Generally test sequences are generated from the IUT specification and characterized by their faults coverage (input faults and output faults). As we have show above, the testing process can avoid the synchronization problems by considering the delay of messages exchanged between testers and the IUT and between testers. We have extended the concept of automaton testing specification to automaton with temporal constraint. In this section we show how we can define the new form of complete test sequences and how the local test sequences could be generated within timing constraints imposed by the new specification.

G. TIMED GLOBAL TEST SEQUENCE TGTS

A Timed global test sequence (TGTS) is a test sequence of an np-TA which corresponds to the sequence of transitions: $Tr_{!x1}.Tr_{?y1}.Tr_{!x2}.Tr_{?y2}\ldots.Tr_{!xt}.T_{?yt}$ where :

- $Tr_{!xi}$ is a transition of sending an output "x_i" in port-k, we denote : <!x,Reseti> . Clocks which will compute the transfer time for each message received in response to this sent are initialized in ***Resert***$_i$.
- $Tr_{?yi}$ represents the outputs sent in the different ports (1,2,….j), with 1<=j<=n in response to the reception of the input "x_i" by the IUT and it has the form: {<?h_1,$guard_1$,$Reset_1$>,<?h_2,$guard_2$,$Reset_2$>….,<?h_j,guardj,Resetj>}.

  (i) Each **guard**$_i$ defines the timing constraint on the reception of "?h_i".

  (ii) **Reset**$_i$ contains the clock of the port receiving "h_i" to be initialized after this transition.

An example of TGTS of 3p-TA (Fig.5.b) is defined by:

- $Tr_{!a1} = (!\mathbf{a_1}, \{c_1,c_2\})$**;**
- $Tr_{?y1} = ?\{(\mathbf{x_1}, c_1 \models \zeta_1, \{c_1\}).(\mathbf{y_1}, c_2 \models \zeta_1, \{c_2\})\}$;
- $Tr_{!b1} = (!\mathbf{b_1}, \{c_1,c_2\})$;
- $Tr_{?y2} = ?\{(\mathbf{x_2}, c_1 \models \zeta_1, \{c_1\}).(\mathbf{y_2}, c_2 \models \zeta_1, \{c_2\})\}$;
- $Tr_{!c1} = (!\mathbf{c_1}, \{c_3\})$**;**
- $Tr_{?y3} = ?\{(\mathbf{z_3}, c_1 \models \zeta_1, \{c_3\})\}$;

This TGTS is written as $Tr_{!a1}.Tr_{?y1}.Tr_{!b1}.Tr_{?y2}.Tr_{!c1}.Tr_{?y3}$:

$$(!\mathbf{a_1}, \{c_1,c_2\}).?\{(\mathbf{x_1}, c_1 \models \zeta_1, \{c_1\}).(\mathbf{y_1}, c_2 \models \zeta_1, \{c_2\})\}.(!\mathbf{b_1}, \{c_1,c_2\}).?\{(\mathbf{x_2}, c_1 \models \zeta_1, \{c_1\}).(\mathbf{y_2}, c_2 \models \zeta_1, \{c_2\})\}.(!\mathbf{c_1}, \{c_3\}).?\{(\mathbf{z_3}, c_1 \models \zeta_1, \{c_3\})\}; \quad (11)$$

The faults covered by Timed Automata with n-ports are classified in:

- faults independent of timing constraints : output faults, transfer faults or combination of both of them
- Timing faults [6]: faults are caused by the violation of timing constraints by the IUT. The test system has to respect timing constraints of inputs and checks if timing constraints of outputs are respected.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

H. TIMED LOCAL TEST SEQUENCES (TLTS)

The introduction of the time concept in the LTS of each tester-k, leads to build Timed Local Test Sequences (TLTS) related to each tester with the form $tr_1.tr_2.tr_3....tr_n$ where each tr_i has the form α_i^k,$guard_i^k$,$Reset_i^k$ and each αi is either:

- $!x_i$: Tester-k sends "x_i" through port k to IUT
- $?y_i$: IUT sends "y_i" through port k to tester-k .
- $!O^k$: Observation message sent to tester-k .
- $?O^k$: Observation message received from tester-k.
- $!S^k$: Synchronization message S sent to tester-k.
- $?S^k$: Synchronization message received by tester -k .

(i) For each message "x_i" sending to the IUT or a Coordination/Observability message, the tester supports the process of sending this message and resets all testers's clock that will receive a message due to this sending in $Reset_i$.

(ii) If "α_i" is an expected message from the IUT or a Synchronization/Observability message, the tester waits for this message. After its reception, the tester checks whether $guard_i$ is true or not and resets the clock in $Reset_i$. If $gaurd_i$ is not true, then test return **Failed**.

The algorithm shown in Fig.7 is dedicated to generate the TLTS from the TGTS. It takes as input a timed global test sequence where we consider each transition as a data structure containing the message to be sent or to be received, the guard to be checked and the list of clocks that will be reset at the end of the transition.

The Loop in (**line 23)** adds the reception of messages belonging to "y_i" to the appropriate sequences. The coordination messages are added to the projections to avoid both Synchronization and Observability Problems:

- To avoid observation problems, each tester receiving a message h ϵ y_{i-1} should be able to determinate that h has been sent by IUT after IUT has received "x_{i-1}" and before IUT receives "x_i". In this case, we added " $?O^k$ " (k is port sending O) and "$!O^k$ "(k is port receiving O) to the appropriate local test sequences **(lines 4 to 19)**
- To avoid Synchronization problems, each tester receiving h ϵ y_i, send a synchronization message to a tester sending xi+1. In this case, we added $?S^k$ (k is port sending S) and $!S^k$ (k is port sending S) to the appropriate local test sequences (**Lines 27-43**).

For a communication with the IUT, we deduce the time constraints (the guard) from the TGTS **(lines 20 to 26).** However, for communication between testers, these constraints will be added to local timed sequences as follow:

- If tester-i sends a message "$!C^j$" or "$!O^j$ " to tester-**j**: there is no constraint time to verify, the message is sent and we initialize all $clock_i$ of testers that will receive this message **(lines 13,18,31,32,40,41,50,51)** .
- If tester-**j** receives the message "$?C^i$" or "$?O^i$" from tester-i: the time c observed from the $clock_i$ must verify the intra-port guard ζ_2 **(lines 17,35,44,54).**

By applying the proposed algorithm to the TGTS giving in (11), we get the following TLTS describing the behavior of tester-1, tester-2and tester-3.

$$\begin{cases} w_{t1}=(!\mathbf{a}_1,\{c_1,c_2\}).(?x_1,c_1|=\zeta_1,\{c_1\}).(!\mathbf{S}^2,\{c_2\}).(?\mathbf{x}_2,c_1| =\zeta_1,\{c_1\})..(!\mathbf{S}^2,\{c_3\}).(?\mathbf{O}^3,c_1|= \zeta_2,\{c_1\}) \\ w_{t2}=(?\mathbf{y}_1,c_2|=\zeta_1,\{c_2\}).(!\mathbf{S}^1,\{c_1\}).(!\mathbf{b}_1,\{c_1,c_2\}).(?\mathbf{y}_2,c_2| =\zeta_2,\{c_2\}.(!\mathbf{C}^3,\{c_3\}) \\ w_{t3}=(!\mathbf{O}^1,\{c_1\}).(?\mathbf{S}^1,c_3|\zeta_2,\{c_3\}).(?\mathbf{S}^2,c_3|\zeta_2,\{c_3\}).(!\mathbf{c}_1,\{ c_3\}) .(?\mathbf{z}_1, c_3|= \zeta_1,\{ c_3\}). \end{cases}$$

Algorithm .Generating Timing Local Test Sequences
Input w=$Tr_{!x1}.Tr_{?y1}.\ Tr_{!x2}.Tr_{?y2}....\ Tr_{!xk}.Tr_{?yk}$
a complete Timing Global Test Sequence (TGTS)
Output: Timed Local test sequences: $(w_{t1},...,w_{tn})$

```
1  for k=1,...,n do wt_k ⇐ ε end for
2  for i=1,...,t do
3  k⇐ Port(Tr!x_i)
////// Generating Observation Messages
4   if i >1 then
5    Send-To ⇐ (Ports (Tr?yi) Δ Ports(Tr?yi-1))\{k}
6    if sender≠0 then
7        Send-To ⇐ Send-To\{sender}
8    end if
9    if sender≠ ∅ then
10       Trw!O^send-To . Message ⇐ !O^send-To
11       Trw!O^send-To . Reset⇐∅
12     For all h ∈ Send-To do
13 Trw!O^send-To . Reset ⇐Trw!O^send-To . Reset .clock_h
       end for
14     wt_k ⇐ wt_k . Trw!O^send-To
15    For all h ∈ Send-To do
16       Trw ?O^k .Message ⇐ ?O^k
```

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

```
17        Trw?O^k.guard ⇐ clock_h |= ζ2
18        Trw?O^k.Reset ⇐clock_h
19        wt_h ⇐ wt_h . Trw?O^k.
        end for
      end if
    end if
20    Trw!xi.Message ⇐  !x_i
21    Trw!xi.Reset ⇐ Tr!xi.Reset
22    wt_k ⇐ wt_k . Trw!xi
23  for all a € y_i do
      Trw?a.Message ⇐  ?a
24    Trw?a.guard⇐ Tr?a.guard
25    Trw?a.Reset ⇐ Tr?a.Reset
26    wt_k ⇐ wt_k . Trw?a.
    end for
//// Generating Coordination Messages
  27  if i <t then
  28   h ⇐Port(x_i+1)
  29    If yi = ∅ then
  30      Trw!S^h. Message ⇐  !S^h
  31      Trw!S^h. Reset ⇐clock_k
  32      wt_k ⇐ wt_k . Trw_S^h.
  33      Trw?S^k . Message ⇐  ?S^k
  34      Trw?S^k . Guard ⇐  clock_k |= ζ2
  35      Trw?S^k. Reset ⇐ clock_k
  36      wt_h ⇐ wt_h .Trw?S^k.
else
   for all a € yi do
  37     Trw!S^k. Message ⇐  !S^k
 38      Trw!S^k. Reset ⇐clock_Port(a)
 39      wt_Port(a) ⇐ wt_Port(a). Trw_S^k.
 40      Trw?S^Port(a) . Message ⇐  ?S^Port(a)
 41      Trw?S^Port(a) . Guard ⇐  clock_Port(a) |= ζ2
 42      Trw?S^Port(a). Reset ⇐ clock_Port(a)
 43      wt_k ⇐ wt_k .Trw?S^Port(a).

    end for
  end if
end for
end Algorithm
```

Fig. 8. Generating Timing Local test sequences

Now, after generating the timed local sequences, we will tackle to compute the test verdict. In fact, during the execution of each timed local test sequence, the tester-k will guarantee (resp. check) the timing constraints (the guard) of the inputs (resp. outputs). More precisely, for each message "xi" sending to the IUT or a coordination/observation message, the tester supports the process of sending this message. If "αi" is an expected message from the IUT or a coordination/observation message, the tester waits for this message. If no message is received in a correct time, 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.

6. CONCLUSION

In practice, the development of the distributed testing system framework is a complex process where the testing systems must not only checks if the output events have been observed, but also the dates when these events have been occurred. In this context, the work presented in this paper is dedicated to extend results from testing distributed system to deal with testing an implementation under test with some timing constraints. There are some contributions in our paper that address those that look at presenting some issues to avoid the coordination, observation and synchronization problems in distributed testing.

We firstly propose that testers exchange synchronization messages, and we proof that this solution resolve both of synchronization and coordination. In other side, we introduce another way to overcome the issues arisen in this context by presenting an algorithm to generate the timed local test sequences that define the behavior of each tester. The main idea beside the proposed work is to consider each transition as a data structure containing the message to be sent or to be received, the guard to be verified and the list of clocks that will be reset at the end of the transition.

Our work is now oriented to develop more consequent testing environments for testing distributed significant application including web services applications, and real-time systems.

REFERENCES

[1] O.Rafiq and L.Cacciari, "Coordination algorithm for distributed testing," *The Journal of Supercomputing*, Volume 24, Number 2, pp 203-211, doi: 10.1023/A:1021759127956 **(2003).**

[2] R.M Hierons, "Testing a distributed system:Generating minimal synchronized,test sequences that detect output-shifting faults".*Information and Software technology*,43(9),pp :551-560,**2001**.

[3] Ural, H., Whittier, D., "Distributed testing without encountering controllability and observability problems, "*Inf. Processing Lett.*, 88(3):pp : 133-141. **2003**.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

[4] 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 .

[5] Tai, K.C., Young. Y.C: "Synchronizable test sequences of finite state machines". *Computer Networks* 13. 1111-1134 (**1998**).

[6] Ahmed Khoumsi, "A Temporal Approach for Testing Distributed Systems, " *IEEE Transactions on Software Engineering*, vol. 28, no.11, pp. 1085-1103 Nov.**2002**,

[7] H.Chuan-dong, J.Fan," Timing issues in distributed testing," *Journal of Zhejiang University SCIENCE* A,8(4), pp :522-528,**2007**.

[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] S. Azzouzi, M.Benattou , H.Charaf "SMA and mobile agents actors for distributed testing," *International Journal of Computer Science Issues*,Volume 7, Issue 5, pp: 231-238 ,September **2010** .

[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] R. M. Hierons and H. Ural, "Checking sequences for distributed test architectures". (*Distributed Computing)* Volume 21,April **2008**, Number 3,pp 223-238,doi: 10.1007/s00446-008-0062-4.

[16] Wen-yu Liu, Hong-wei Zeng and Huai-kou Miao, "Multiple UIO-based test sequence generation for distributed systems ",*Journal of Shanghai University (English Edition)* Volume 12, November **2007**,Number 5, pp 438-443, doi: 10.1007/s11741-008-0512-3.

[17] Jessica Chen, Robert M. Hierons and Hasan Ural, "Testing in the Distributed Test Architecture Formal Methods and Testing". *Lecture Notes in Computer Science*, **2008**, Volume 4949/2008, 157-183, doi: 10.1007/978-3-540-78917-8_5.

[18] Tai, K.C., Young. Y.C: "Synchronizable test sequences of finite state machines". *Computer Networks* 13. 1111-1134 (**1998**).

[19] Luo, G., Dssouli, R., Bochmann, G.v.: "Generating synchronizable test sequences based on finite state machine with distributed ports". *In the 6th IFIP workshop on protocol Test Systems*, pp.139-153. Elsevier, North-Holand (**1993**).

[20] R.M Hierons."Testing a distributed system"Generating minimal synchronized test sequences that detect output-shifting faults".*Information and Software technology*,43(9):551-560,**2001**.

[21] Luo, G., Dssouli, R., Bochmann, G.v.,Venkatram,P.,Ghedamsi,A.:"Test Generation with respect to distributed interfaces" 16,119-132 (**1994**).

[22] J.Chen,R.M. Hierons, and H.Ural. "Conditions for resolving observability problems in distributed testing". *In 24rd IFIP International Conference on Formal Techniques for Networked and Distributed Systems* (FORTE 2004), volume 3731 of LNCS, pages 229-242.Springer-Verlag, **2004**.

[23] J.Chen,R.M. Hierons, and H.Ural. "Resolving observability problems in distributed test architecture". *In 25rd IFIP International Conference on Formal Techniques for Networked and Distributed Systems* (FORTE 2005), volume 3731 of LNCS, pages 219-232.Springer-Verlag, **2005**.

[24] 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.

[25] K. Correll, N. Barendt, and M. Branicky, "Design Considerations for Software Only Implementations of the IEEE 1588 Precision Time Protocol," *In Proc.Conf. on IEEE-1588 Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*, October **2005**.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

[26] R. Alur and D. L. Dill. A theory of timed automata. Theoretical Computer Science, **1994.**

[27] A. Khoumsi, "A new method for testing real time systems," *in Real-Time Computing Systems and Applications*, pp. 441 - 450 ,Dec. **2000**.

[28] Christophe CUBAT DIT CROS"Agents Mobiles Coopérants pour les EnvironnementsDynamiques,"ENSEEIHT Institut National Polytechnique de Toulouse,**2005**

[29] S.Azzouzi, M.Benattou, My El Hassan Charaf, "Real Time Agent Based Approach for Distributed Testing", in *IEEE proceedings of The 3rd International Conference on Multimedia Computing and Systems* (ICMCS'12) May 10-12 2012, Tangier, Morocco.

[30] S. Azzouzi, M.Benattou, H.Charaf "Agent Synchronization issues in distributed testing, ", *on Seventh International Conference ON Intelligent Systems : Theories AND Applications (SITA'12)* 16-17 MAY 2012, Mohammedia,Morocco