International Journal of Innovative
Computing, Information and Control
Volume **9**, Number **9**, September **2013**

ICIC International ©2013 ISSN 1349-4198
pp. 3779–3797

AN EXPERT SYSTEM BASED ARCHITECTURE FOR TESTING DISTRIBUTED SYSTEMS

My EL Hassan Charaf, Mohammed Benattou and Salma Azzouzi

LARIT: Laboratory of Research in Computer Science and Telecommunication
Faculty of Science
Ibn Tofail University
Kenitra 14000, Morocco
{ charaf; salma.azzouzi }@gmail.com; mbenattou@yahoo.fr

Received June 2012; revised December 2012

Abstract. *In the distributed test context, where a set of parallel testers exchange some I/O messages to perform the test, 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 such as time outs, locks, channels and network failures. Nowadays, rule based systems provide an interesting approach for representing and interpreting such kind of messages exchange using the artificial intelligence features. In this paper, we suggest two algorithms allowing the generation of rules to be respected by the test system to avoid the coordination/synchronization problems. Then, we explain the advantages of the study and how the proposed architecture based on expert systems can avoid the use of the coordination messages and resolve the coordination problems. Thus, the testers will exchange only the observation messages which will reduce significantly the use of external messages and I/O operations.*
Keywords: Expert system, Distributed test, Rules, Controllability and observability problems, Synchronization

1. **Introduction.** Nowadays, the distributed computing becomes the key issue in modern systems design. It provides new high promises for Web-based applications. However, the inherent complexity of the architecture of distributed systems requires special testing techniques. In fact, contrary to the centralized test where the entire activity of the test (injection of stimulis and observing reactions of the implementation under test: (IUT)[1] is performed by a single entity; this activity is performed by a set of parallel testers called (PTCS)[2] in the distributed context [5]. However, many problems influencing faults detection during the conformance testing process arise if there is no coordination between PTCs. In fact, the use of multiple testers introduces the possibility of coordination problems amongst remote testers. These potential problems are known as controllability and observability fault detections which are fundamental features of conformance in distributed testing [14]. Concretely, to test Web-Based application for example, the model of the Web application can be obtained by partitioning the application into collections of web pages and software modules that implement some logical function. Then, the web pages that include more than one HTML form, each of which is connected to a different back-end software module, can also be modeled as multiple Logical Web Pages in order to facilitate testing of these modules. Generally, we simulate the implementation under

[1]IUT : Implementation Under Test
[2]PTC : Parallel Tester Component

test as a "black-box", and its behavior is known only by interactions through its interfaces with the environment or other systems. Additionally, the implementation process is more complex and must consider the mechanisms and functions required to support interaction as long as the communication and the coordination between the distributed testing components. The typical reactions of such systems are the generation of set of errors: time outs, locks, channels and network failures [2].

To avoid the problems related to distributed test, many works [3,5,14,16] propose to introduce some coordination messages which 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. In this approach, the difficulty lies in writing the procedure for coordination between the PTCs. Given its importance in the validation process, it should not reject a conform implementation or lead to the acceptance of an incorrect one. Compared with these works that deduce local test sequences for each tester from the global one and including some coordination and observation messages to ensure coordination between testers, we suggest in this paper two algorithms allowing the generation of rules to be respected by the test system to avoid the coordination problem without requiring the use of coordination messages because such messages exchange can introduce delays especially if there are some timing constraints. The basic idea behind introducing the rule's concept in the distributed test context is that the exchange of messages to perform the test is sequential. In fact, for each transition in the test process, the next messages to be sent to the IUT depend mainly on the previous messages received even from the IUT or from other testers. The idea is to write the rules to be respected by the testers to guarantee their coordination. To communicate with the IUT, the testers follow some instructions described through these rules. When the necessary conditions (facts) have arisen, the tester proceeds in applying results as described in its local rules. The proposed algorithms are inspired from works done by [3,5]. The first algorithm generates a matrix of local rules to be fulfilled by each tester. However, as detailed in the article, we can notice that the verdict of the test can be obtained by calculating if all the local rules have been respected in each tester during the test execution. In fact, for each message xi sent to the IUT or an observation message, the tester supports the process of sending this message. If xi is an expected message from the IUT or an observation 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 all the local rules of the tester have been satisfied, 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. Thus, in the point of view of the Test system, the coordination is ensured using some global rules. The second algorithm generates the list of facts and the global rules to be respected by the whole system. In the other hand, the emphasis of recent works is to minimize the use of external coordination message exchanges among testers [11,14] or to identify conditions on a given FSM under which controllability and observability problems can be overcome without using external coordination messages [13,15]. The idea of such works is to construct a test sequence that causes no controllability or observability problems during its application in distributed test architecture. For some specifications, such test sequence exists where the coordination is achieved via their interactions with the IUT [12]. However, this case is not always true as detailed in [9,13]. In this paper, we explain the advantages of the study and how the proposed architecture can avoid the use of the coordination messages and resolve the coordination problems. Thus, the testers will exchange only the observation messages which will reduce significantly the use of external messages and I/O operations. In fact, each tester executes only a part of the global reasoning, and diffuses through the network the obtained results. By the way,

other testers can use these results to participate in the global reasoning. To this end, we present rules and facts as components of a Petri net to benefit of its formalism. Then, we explain when we have the facts and rules list of the Petri Net diagram, represented in a matrix form (A: Antecedents, C: Consequences) and the initial state of the system $M0$, we can then deduce using simple arithmetic operations (sensitized/fired rules), the state of the system and decide if some rules can be enabled. The work presents another way to overcome the coordination/synchronization problems and avoid the exchange of the external coordination messages among remote testers during the test. As detailed in the paper, the objective of introducing rules is to ensure coordination using rule based systems that provide an interesting approach for representing and interpreting such kind of messages exchange using the artificial intelligence features. This kind of systems permits the implementation of highly flexible systems capable of adapting themselves to different situations by seeking to express an automatism in a similar way to as would make it a human being: "IF antecedents THEN consequents". Additionally, such systems are able to take decisions concerning possible malfunctions and decided if the process of test returns a failed verdict or an accepted one.

This paper presents some technical issues for testing distributed frameworks with Expert System. The proposed approach consists firstly of exploring the benefits of Expert systems to concept communication between different components of the distributed test application. To this end, the paper is organized as follows. The second section describes the concept of distributed testing, architecture and the test procedure while referring to the problems of control, observation and synchronization. The third section introduces our algorithms to generate the rules that will be used to avoid the coordination/synchronization problem. In the fourth section, we present rules and facts as components of a Petri net to benefit of its formalism and finally a practical example of a distributed chat group application is given in the fifth section to demonstrate the effectiveness and efficiency of the main results and the motivation of the practical use of the results developed.

2. **Distributed Test.**

2.1. **Architecture.**

The basic idea is to coordinate parallel testers using a communication service in conjunction with the IUT. Each tester interacts with the IUT through a port called the Point of Control and Observation (PCO)[3] and communicates with other testers through a multicast channel (Figure 1). 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.

2.2. **Modeling by automaton.**

To approach the testing process in a formal way, the specification and the Implementation Under Test (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 (FSM Finite State Machine) [1] defining inputs and the results expected for each PCO. A multi-port FSM with n ports (np-FSM) A is a 6-tuple (Q, Σ, Γ, δ, λ, q_0), where: Q is the finite set of states of A; $q_0 \epsilon Q$ is the initial state of A; Σ is a n-tuple $(\Sigma_1, \Sigma_2, \ldots, \Sigma_n)$ where Σ_k is the input alphabet of port k, and $\Sigma i \cap \Sigma j = \varnothing$ for $i <> j$. We denote $\overline{\Sigma}$ the input alphabet $\Sigma_1 \cup \Sigma_2 \cup \ldots \cup \Sigma_n$ of A; Γ is a n-tuple $(\Gamma_1, \Gamma_2, \ldots, \Gamma_n)$ where Γ_k is the output alphabet of port k, and $\Gamma_i \cap \Gamma_j = \varnothing$ for $i <> j$. We write Π for the output alphabet $(\Gamma_1 \cup \{\})x(\Gamma_2 \cup \{\})x \ldots x(\Gamma_n \cup \{\})$ of A; δ is the

[3]PCO : Point of Control and Observation

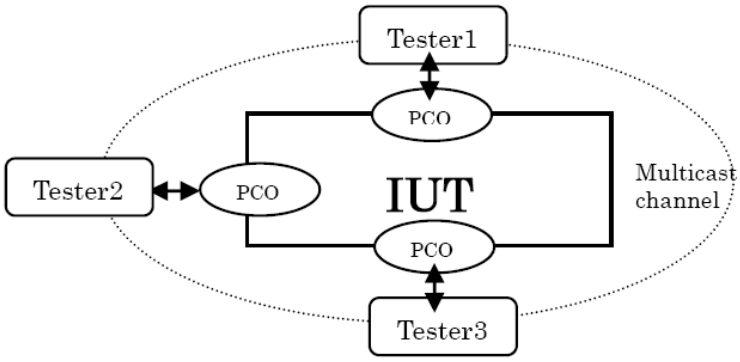

FIGURE 1. Test architecture

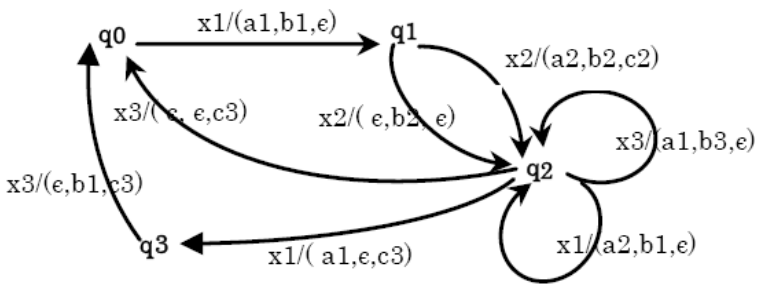

FIGURE 2. An example of 3p-FSM

transition function, it is a partial function $Q \times \overline{\Sigma} \to Q$; λ is the output function, it is a partial function $Q \times \overline{\Sigma} \to \Pi$. A transition of an np-FSM is a 4-tuple $t = (q, \alpha, \gamma, q')$ where q, $q' \epsilon Q$, $\alpha \epsilon \overline{\Sigma}$ and $\gamma \epsilon \Pi$ are such that $\delta(q, \alpha) = q'$ and $\lambda(q, \alpha) = \gamma$.

Figure 2 gives an example of 3p-FSM with $Q = \{q_0, q_1, q_2, q_3, q_4\}$, q_0 initial state, $\Sigma_1 = \{x_1\}$, $\Sigma_2 = \{x_2\}$, $\Sigma_3 = \{x_3\}$, and $\Gamma_1 = \{a_1, a_2, a_3\}$, $\Gamma_2 = \{b_1, b_2, b_3\}$, $\Gamma_3 = \{c_1, c_2, c_3\}$. 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 $i = 1, \ldots, t : x_i \epsilon \overline{\Sigma}$, $y_i \subset \cup_{k=1}^{n} \Gamma_k$ 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 y_i from the IUT.

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

$$!x_1?\{a_1, b_1, \epsilon\}!x_2?\{a_2, b_2, c_2\}!x_1?\{a_2, b_1, \epsilon\}!x_3?\{a_1, b_3, \epsilon\}!x_1?\{a_1, \epsilon, c_3\}!x_3?\{\epsilon, b_1, c_3\}. \quad (1)$$

Generally, test sequences are generated from the specification of the IUT and characterized by fault coverage. Several methods exist for generating test sequences from I/O FSM specifications. They are mainly for detecting the following types of fault: output faults, transfer faults or combination of both of them [2]. An edge with an incorrect output has an output fault which is generally observable. Any generation method providing a test suite traversing each transition of an FSM at least once is capable of detecting such faults. An edge with an incorrect starting state or ending state has a transfer fault. Since states are not directly observable, these faults are relatively more difficult to detect. In the distributed test architecture, the application of a test sequence may introduce some issues known as controllability and observability problems. These problems occur if a tester cannot determine either when to apply a particular input to the IUT, or whether a particular output from the IUT has been generated in response to a specific input, respectively.

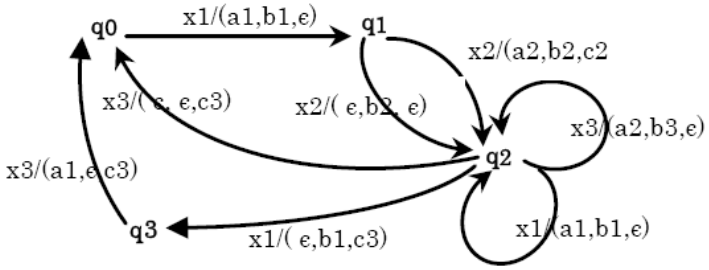

FIGURE 3. Example of a faulty IUT for Figure 2

2.3. **Distributed test problems.** Many kinds of problems can arise in the distributed test context; we define these notions by referring [3].

Controllability problem can be defined from Test System view as capability of a Test System to force the IUT to receive inputs in the given order. It arises when Test cannot guarantee that IUT will receive event of transition (i) before event of transition $(i+1)$. In other way, it is the capability of the test system to realize input events at corresponding PCOs in a given order.

Observability problem can be defined from Test System view as capability of a Test System to observe the outputs of the IUT and decide which input is the cause of each output. For distributed test architecture where a transition contains at most single output for each output, the observability problem arises when two consecutive transition (i) and transition $(i+1)$ occurs on the same port k but only one of the transitions has an output in port k and the other one is an empty transition with no output. In this case the Test System cannot decide whether transition (i) or transition $(i+1)$ is the cause of output.

EXAMPLES: Let us explain these situations by giving a faulty IUT related the global test sequence (1) as shown in the Figure 3.

The projections of (1) on ports alphabets are required to get the test sequence related to each tester. Projections w_1, w_2, w_3 of the global test sequence are $w_1 = !x_1?a_1?a_2!x_1?a_2?a_1!x_1?a_1$, $w_2 = ?b_1!x_2?b_2?b_1?b_3?b_1$ and $w_3 = ?c_2!x_3?c_3!x_3?c_3$. By applying these sequences to the IUT of Figure 3 using the remote method, we will have these situations:

Situation 1. When the IUT is in state q_2, it gets both x_1 on port 1 and x_3 on port 3 respectively. Then, either it follows the intended path reading x_1 before x_3, or it reads x_3 before x_1. In the first case, tester 1 receives a_1 before a_2 and it detects an output fault. However, if the IUT decides to read a_2 before a_1 then none of the testers is able to detect this fault.

Situation 2. When the IUT is in the state q_2, it receives x_1 and then it sends b_1 and c_3 instead of a_1 and c_3. In the state q_3, it receives x_3 and sends a_1 and c_3 instead of b_1 and c_3. In this situation, we can notice that there are two successive output faults but none of the testers can detect them.

To resolve such problems, authors in [11] propose an algorithm to generate local test sequences from the global test sequence. We will get the following local test sequences by applying the algorithm mentioned above to the global test sequence (1):

$$\begin{cases} w_1 = !x_1?a_1?a_2\mathbf{!O_3}!x_1?a_2?a_1\mathbf{!O_{\{2,3\}}}!x_1?a_1\mathbf{?O_3}, \\ w_2 = ?b_1\mathbf{!O_3}!x_2?b_2?b_1!C_3?b_3\mathbf{?O_1?O_3}?b_1, \\ w_3 = \mathbf{?O_2}?c_2\mathbf{?O_1}?C_2!x_3\mathbf{?O_1}?c_3\mathbf{!O_{\{1,2\}}}!x_3?c_3. \end{cases} \tag{2}$$

As shown in the obtained local test sequences, some coordination messages (C_k) are added to the projections to avoid both the controllability and observability problems when using the complete test sequence. We notice two kinds of coordination messages:

- $!C_{\{t_1,..,t_r\}}(!O_{\{t_1,..,t_r\}}$ resp.) the sending of a coordination message (observation message resp.) to the testers $t_1..t_r$.
- $?C_t(?O_t$ resp.) the receipt of a coordination message (observation message resp.) from the tester t.

Synchronization problem. As explained above, the algorithm in [3] allows the generation of local test sequences to be performed by each tester. When these local test sequences have been obtained, each tester is running its local test sequence produced from the global test sequence of the IUT. Thus, the testers are working together but independently, which leads us to manage the problem of synchronization of testers.

The first fragments of the local test sequences obtained in (2).

$$\begin{cases} w_{f1} = !x_1?a_1, \\ w_{f2} = ?b_1!O_3!x_2, \\ w_{f3} = ?O_2?c_2. \end{cases} \tag{3}$$

The execution of the fragments w_{f1}, w_{f2} and w_{f3} should give the result shown in Figure 4(a) but the execution of our prototype provides an incorrect result given in Figure 4(b). Indeed, in last diagram the second tester sends the message x_2 the IUT before the first tester receives the message a_1 from the IUT. So, the execution of local testing is not conform with the specification in (1), where the message 'x_2' must be sent only if all messages due to the sending of 'x_1' by the tester-1 are received by the IUT. In the following of this paper, we will take – for simplicity, the test sequence of 3p-FSM shown in Figure 1 defined as:

$$!x_1?\{a_1, b_1, \epsilon\}!x_2?\{a_2, b_2, c_2\}!x_3?\{\epsilon, \epsilon, c_3\}. \tag{4}$$

2.4. **Related works.** Many works have been made to avoid the problems described in the previous section. Indeed, the author in [4] shows that controllability and observability are indeed resolved if and only if the test system respects some timing constraints. Then the article determines these timing constraints and other timing constraints which optimize the duration of test execution. In this context, we determine in other work [21] timing conditions that guarantee communication between components of distributed testing architecture and we propose a distributed architecture for testing distributed Real-Time Systems then we propose our Multi-Agent architecture for testing these systems. In [5], the authors explain how both controllability and observability problems can be overcame through the use of coordination messages among remote testers.

The work [6] proposes a new method to generate a test sequence utilizing 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 [7], the authors suggest to construct a test or checking sequence from the specification of the system under test such that it is free from these problems without requiring the

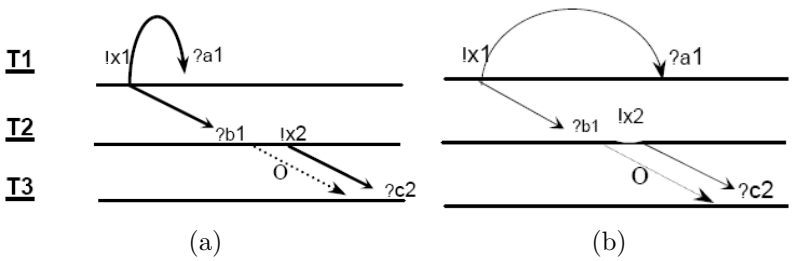

FIGURE 4. The execution result

use of external coordination messages. In this context, they propose some algorithms for constructing subsequences that eliminate the need for external coordination messages.

Another work [8] shows that the use of coordination messages can introduce delays and this can cause problems where there are timing constraints. Thus, sometimes it is desired to construct a checking sequence from the specification of the system under test that will be free from controllability and observability problems without requiring the use of external coordination message exchanges. To this end, the authors suggest an algorithm that achieves this.

The main idea in [9-11] is to construct a test sequence that causes no controllability or observability 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 [12]. However, this case is not always true as detailed in [9,13].

The emphasis of recent works is to minimize the use of external coordination message exchanges among testers [11,14] or to identify conditions on a given FSM under which controllability and observability problems can be overcome without using external coordination messages [13,15].

Finally, our work is mainly based on [5,16] and the algorithm proposed in [3] for writing test coordination procedures in a distributed testing architecture.

The paper can be considered as a continuity of [17,19] and [20] where we propose to introduce some concepts issued from Artificial Intelligence especially the use of agents, rule based systems or the MAS (multi-agent system) incorporated with ontology. In the next section, we propose our solution by defining some rules to be implemented in each tester to overcome problems related to the distributed test.

3. **Testing Rules Generation.** The basic idea behind introducing the rule's concept in the distributed test context is that the exchange of messages to perform the test is sequential. In fact, for each transition in the test process, the next messages to be sent to the IUT depend mainly on the previous messages received even from the IUT or from other testers. The idea is to write algorithms to deduce –from the global test sequence– the rules to be respected by the testers to guarantee their coordination. In fact, each rule is composed by two parts, conditions and results. These components are shared between the IUT and the testers as facts. To communicate with the IUT, the testers follow some instructions described through these rules. When the necessary conditions (facts) have arisen, the tester proceeds in applying results as described in its local rules.

Let us take the global test sequence $!x_1?\{a_1, b_1, \epsilon\}!x_2?\{a_2, b_2, c_2\}!x_3?\{\epsilon, \epsilon, c_3\}$ defined in (4). It can be translated on a set of rules as follows:

- If the tester T_1 send a message x_1 to the IUT $\boldsymbol{(!x_1.T_1)}$ then the tester T_1 will receive a message a_1 from the IUT $\boldsymbol{(?a_1.T_1)}$ and the tester T_2 will receive a message b_1 from the IUT $\boldsymbol{(?b_1.T_2)}$.
- If the message a_1 is received in the tester T_1 $\boldsymbol{(?a_1.T_1)}$ and the message b_1 is received in the tester T_2 $\boldsymbol{(?b_1.T_2)}$. Then the tester T_2 will apply the message x_2 to the IUT $\boldsymbol{(!x_2.T_2)}$.

At this stage, we have an observability problem so we will introduce an observation message O_3 to be sent by tester T_2 to the tester T_3. In this case, the next rule is as follows:

- If the tester T_2 send a message x_2 to the IUT $\boldsymbol{(!x_2.T_2)}$ then the tester T_1 will receive a message a_2 from the IUT $\boldsymbol{(?a_2.T_1)}$ and the tester T_2 will receive a message b_2 from the IUT $\boldsymbol{(?b_2.T_2)}$ and the tester T_3 will receive a message c_2 from the IUT $\boldsymbol{(?c_2.T_3)}$ and the tester T_2 will send an observation message O_3 to tester T_3 $\boldsymbol{(!O_3.T_2)}$.

All these rules can be expressed over each tester as local rules as follows:

- $!x_1.T_1 \dashrightarrow ?a_1.T_1$; $!x_1.T_1 \dashrightarrow ?b_1.T_2$;
- $?a_1.T_1 \dashrightarrow !x_2.T_2$; $?b_1.T_2 \dashrightarrow !x_2.T_2$;
- $!x_2.T_2 \dashrightarrow ?a_2.T_1$; $!x_2.T_2 \dashrightarrow ?b_2.T_2$; $!x_2.T_2 \dashrightarrow ?c_2.T_3$; $!x_2.T_2 \dashrightarrow !O_3.T_2$

However, we can notice that the verdict of the test over the whole system can be obtained by calculating if all the local rules have been respected in each tester during the test execution. Thus, in the point of view of the Test system, the coordination is ensured using the global rules as follows:

- $!x_1.T_1 \dashrightarrow ?a_1.T_1^{\wedge}?b_1.T_2$,
- $?a_1.T_1^{\wedge}?b_1.T_2 \dashrightarrow !x_2.T_2$,
- $!x_2.T_2 \dashrightarrow ?a_2.T_1^{\wedge}?b_2.T_2^{\wedge}?c_2.T_3^{\wedge}!O_3.T_2$.

In the next subsections, we explain how we can generate (local/global) rules from a given global test sequence. To this end, we introduce two algorithms to achieve the rules generation.

3.1. **Local testing rules.** In this section, we describe the algorithm1 allowing the generation of the local rules to be respected by the testers to avoid the coordination problem. The algorithm is inspired from woks done by [3,5]. Contrary to their works that generate some local test sequences from the global test sequence and that introduce coordination and observation messages, the proposed algorithm generates local rules Rij to be fulfilled by each tester. We denote by '/' the set difference and 'Δ' the symmetrical difference: AΔB = (A/B) U (B/A). The function Port gives the port corresponding to a given message. For a set y of messages, the function Ports is defined by: $Ports(y) = \{k | \exists a \epsilon y$ s.t. $k = Port(a)\}$.

Let us take the global test sequence defined in (4) as the input of the algorithm below. The algorithm generates a matrix of local rules by browsing the 't' messages to be sent to the IUT in the global test sequence (line 1), e.g., 't' represents the number of lines of the matrix of local rules. Then, the local rules will be constructed as:

- Each message 'm' belonging to yi is a part of a rule in the matrix as a consequence of sending message xi **(lines 5, 6, 7)**, we denote '$!RR_i$' the set of local rules generated in response of sending the message xi.

$$^{!}RR_i = \{R_{ij} : !x_i.T_i \rightarrow ?m.T_j / j = 1..p\},\ p \text{ is the cardinality of } y_i.$$

- Each message 'm' belonging to y_i is a part of a rule in the matrix as an antecedent of sending message x_{i+1} **(lines 12, 13, 14)**, we denote '$?LR_i$' the set of local rules needed to send the message x_{i+1}.

$$^{?}LR_i = \begin{cases} \{R_{ij} : ?m.Tj \rightarrow !x_{i+1}.T_{i+1} / j = 1..p\},\ p \text{ is the cardinality of } y_i: & \text{if } i < t. \\ \emptyset & \text{otherwise.} \end{cases}$$

To avoid observation problems, each tester receiving a message $h \epsilon y_{i-1}$ 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. **(lines 19, 20)**. Afterwards, we introduce the observation messages to write rules for avoiding this problem **(lines 21, 22, 23, 24)**. We denote:

$$\zeta_i = \{Ports(y_i) \Delta Ports(y_{i-1})\} \backslash \{port(xi)\} \tag{5}$$

We define also '$^{!}OR_i$' and '$^{?}OR_i$' the sets of local rules generated to overcome the problem of observability as:

- $^{!}OR_i = \{R_{ij} :\ ^{!}x_i.T_i \rightarrow^{!} O_k.T_i / k \epsilon \zeta_i$ and $j = 1..q\}$
- $^{?}OR_i = \{R_{ij} :\ ^{!}O_k.T_i \rightarrow^{?} O_i.T_k / k \epsilon \zeta_i$ and $j = 1..q\}$; q is the cardinality of ζ_i.

```
Algorithm1 .Generating local rules from a global test sequence
        Input w=!x1? y1!x2.....!xt? yt : The Global Test Sequence
                    yi : vector of   messages received when sending message xi
         Output : Local Rules, Rtn   matrix of rules as :
                                t   : number of messages sent
                                 n : number of rules for each message sent xi
1. for i=1,...,t   do
2.          k= Port(xi)
3.          l= Port(xi+1)
4.          h=1
5.       for all j є yi   do
6.              if (j <> e)
7.             Rih :      !xi.Tk→ ?j.Tport(j)
8.              h=h+1
9.              end if
10.         end for
11.         n=h+1
12.      for all j є yi   do
13.             if (j <> e)
14.            Rin : ?j. Tport(j)→ !x i+1.Tl
15.            n=n+1
16.            end if
17.    end for
18.         o= n
19.         if i>1 then
20.         Send_To<=(Ports(yi ) ΔPorts(yi -1))/{k}
21.          if Send_To <> ensemble_vide then
22.          Rio :      !xi.Tk→ !OSend_To.Tk
23.          for all p є Send_To
24.          Rio :      !Op.Tk → ?Ok.TP
25.      End for
26.      End if
27.      End if
28. end for
End Algorithm1
```

Thus, we can deduce the number of local rules to be generated for each message xi to send to the IUT as:

$$£_i = card(^!RR_i) + card(^?LR_i) + card(^!OR_i) + card(^?OR_i) \text{ for } i = 1..t-1,$$

$$£_t = card(^!RR_i) + card(^!OR_i) + card(^?OR_i).$$

$$£_i = 2 * (p + q) \text{ for } i = 1..t-1 \text{ and } £_t = p + 2 * q \tag{6}$$

We will obtain then the local rules matrix Rnm with n the number of lines which correspond to the number of messages 't' to be sent to the IUT and m the number of columns that correspond to the $\max\{£_i / i = 1..t\}$.

In the case of the global test sequence defined in "Equation (4)" we will have:

- For $i = 1$, $p = 2$ and $q = 0$ then $£_1 = 4$.
- For $i = 2$, $p = 3$ and $q = card\ (\{1,2,3\}\Delta\{1,2\}\backslash\{2\}) = 1$ then $£_2 = 8$.
- For $i = 3$, $p = 1$ and $q = card\ (\{3\}\Delta\{1,2,3\}\backslash\{3\}) = 2$ then $£_3 = 5$.

Therefore, by applying Algorithm 1 to the global test sequence (4) defined in our example, the obtained matrix is a R_{38} matrix composed by the elements R_{ij} defined as Table 1.

As signaled at the beginning of this section, we can notice that the verdict of the test can be obtained by calculating if all the local rules have been respected in each tester during the test execution. In fact, for each message xi sent to the IUT or an observation message, the tester supports the process of sending this message. If xi is an expected message from the IUT or an observation message, the tester waits for this message. If no message is received, or if the received message is not expected, the tester returns a

TABLE 1. The matrix of local rules deduced from (4)

$\mathbf{R_{11}} : !x_1.T_1 \dashrightarrow ?a_1.T_1$	$\mathbf{R_{21}} : !x_2.T_2 \dashrightarrow ?a_2.T_1$	$\mathbf{R_{31}} : !x_3.T_3 \dashrightarrow ?c_3.T_3$
$\mathbf{R_{12}} : !x_1.T_1 \dashrightarrow ?b_1.T_2$	$\mathbf{R_{22}} : !x_2.T_2 \dashrightarrow ?b_2.T_2$	$\mathbf{R_{32}} : !x_3.T_3 \dashrightarrow !O_1.T_3$
$\mathbf{R_{13}} : ?a_1.T_1 \dashrightarrow !x_2.T_2$	$\mathbf{R_{23}} : !x_2.T_2 \dashrightarrow ?c_2.T_3$	$\mathbf{R_{33}} : !x_3.T_3 \dashrightarrow !O_2.T_3$
$\mathbf{R_{14}} : ?b_1.T_2 \dashrightarrow !x_2.T_2$	$\mathbf{R_{24}} : ?a_2.T_1 \dashrightarrow !x_3.T_3$	$\mathbf{R_{34}} : !O_1.T_3 \dashrightarrow ?O_3.T_1$
$\mathbf{R_{15}} : \epsilon$	$\mathbf{R_{25}} : ?b_2.T_2 \dashrightarrow !x_3.T_3$	$\mathbf{R_{35}} : !O_2.T_3 \dashrightarrow ?O_3.T_2$
$\mathbf{R_{16}} : \epsilon$	$\mathbf{R_{26}} : ?c_2.T_3 \dashrightarrow !x_3.T_3$	$\mathbf{R_{36}} : \epsilon$
$\mathbf{R_{17}} : \epsilon$	$\mathbf{R_{27}} : !x_2.T_2 \dashrightarrow !O_3.T_2$	$\mathbf{R_{37}} : \epsilon$
$\mathbf{R_{18}} : \epsilon$	$\mathbf{R_{28}} : !O_3.T_2 \dashrightarrow !O_2.T_3$	$\mathbf{R_{38}} : \epsilon$

verdict **Fail** (fail). If all the local rules of the tester have been satisfied, 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**. Thus, in the point of view of the Test system, the coordination is ensured using some global rules (r_i).

3.2. **Global testing rules.** In this subsection, we will introduce Algorithm 2 allowing the production of the global rules that will be satisfied by the whole test system. To this end, we denote $RRij$ (and $LRij$ resp.) the right (and left resp.) parts of the local rules Rij in the matrix Rnm. Both $RRij$ and $LRij$ are defined only if $Rij <> \epsilon$. Then, we define ${}^{R}R$ and ${}^{L}R$ as follows:

- ${}^{R}R = \{{}^{R}R_{ij}/$ for $i = 1..n$ and $j = 1..m\ \}$,
- ${}^{L}R = \{{}^{L}R_{ij}/$ for $i = 1..n$ and $j = 1..m\ \}$

Therefore, we will obtain the list of facts F of the system defined as:

$$F = {}^{R}R \Delta {}^{L}R \tag{7}$$

For our example, the list F of facts is defined as:

$$\begin{aligned} F\ =: \{\ & !x_1.T_1 - ?a_1.T_1 - ?b_1.T_2 - !x_2.T_2 - ?a_2.T_1 - ?b_2.T_2 - ?c_2.T_3 - !O_3.T_2 \\ & - !x_3.T_3 - ?O_2.T_3 - ?c_3.T_3 - !O_1.T_3 - !O_2.T_3 - ?O_3.T_1 - ?O_3.T_2\} \end{aligned}$$

In Algorithm 2, we browse the set of facts F **(line 2)** and for each fact related to a sending message **(line 3)** we construct then:

- The right side of the global rule rR if the fact is belonging to LR (**lines 8 and 9**).
- The left side of the global rule rL if the fact is belonging to RR (**lines 10 and 11**).

We obtain then the global rules ri as:

- $ri : f \rightarrow rR$ (**line 16**),
- $ri : rL \rightarrow f$ (**line 20**).

Therefore, the list of the global rules R can be deduced from the matrix of local rules by applying Algorithm 2 as mentioned below:

$$R = : \{r_1 - r_2 - r_3 - r_4 - r_5 - r_6 - r_7 - r_8\}$$

- $r_1 : !x_1.T_1 \dashrightarrow ?a_1.T_1^{\wedge}?b_1.T_2$,
- $r_2 : ?a_1.T_1^{\wedge}?b_1.T_2 \dashrightarrow !x_2.T_2$,
- $r_3 : !x_2.T_2 \dashrightarrow ?a_2.T_1^{\wedge}\ ?b_2.T_2^{\wedge}?c_2.T_3^{\wedge}!O_3.T_2$,
- $r_4 : ?a_2.T_1^{\wedge}?b_2.T_2^{\wedge}?c_2.T_3 \dashrightarrow !x_3.T_3$,
- $r_5 : !O_3.T_2 \dashrightarrow ?O_2.T_3$,
- $r_6 : !x_3.T_3 \dashrightarrow ?c_3.T_3^{\wedge}!O_1.T_3^{\wedge}!O_2.T_3$,
- $r_7 : !O_1.T_3 \dashrightarrow ?O_3.T_1$,
- $r_8 : !O_2.T_3 \dashrightarrow ?O_3.T_2$.

```
Algorithm2 .Generating Global rules from the matrix Rnm
        Input: Rnm The matrix of local rules, F: List of facts
        Output: List of the global rules
Begin
1.  p=1
2. For all f e F do
3.  If f is a Sending_Fact then
4.      r^R= e
5.       r^L= e
6.      For i=0 to n-1 do
7.              For j=0 to m-1 do
8.                      If    ^LR_ij =f then
9.                              r^R= r^R ^ ^RRij
10.                             Else If    ^RRij =f   and f <> Observation_Fact then
11.                             r^L= r^L ^ ^LRij
12                      End if
13.             End For
14.       End For
15.         If      r^L <> e then
16.           r_p :    r^L   → f
17.           p=p+1
18.       End if
19.         If      r^R <> e then
20.           r_p :      f → r^R
21.           p=p+1
22.       End if
23.  End if
24.  End For
End Algorithm 2
```

In the next section, we introduce the expert systems that will implement rules and facts described previously. Then, we define interactions between different components of the system and finally, we describe the test procedure.

4. **Test Prototype Based on Expert Systems.** An expert system is typically composed of at least three primary components. These are the knowledge base which is a collection of rules or other information structures derived from the human expert, the inference engine that enables the expert system to draw deductions from the rules in the KB and finally the working memory which contains the data that is received from the user during the expert system session.

4.1. **Architecture.** A distributed expert system has been proposed to avoid the synchronization problem described above. It is mainly based on the use of distributed nodes connected to the IUT where nodes participate in the distributed system execution.

Each node executes only a part of the global reasoning, and diffuses through the network the obtained results. By the way, other nodes can use these results to participate in the reasoning. As shown in the Figure 5, the system is composed of the following components:

- The IUT (Implementation Under Test) is the implementation to be tested.
- Some ESTi (Experts System Testers) connected to the IUT using a PCOi (Point of Control and Observation) to exchange inputs/outputs messages.
- A global KB (Knowledge Base) that store facts, global rules, variables and the States vector.

Each ESTi uses its inference engine and its working memory to communicate with the KB for making a global reasoning.

After obtaining the lists of facts and global rules and designing our test architecture, we describe in the next section the behavior of the test system using the Petri Net formalism

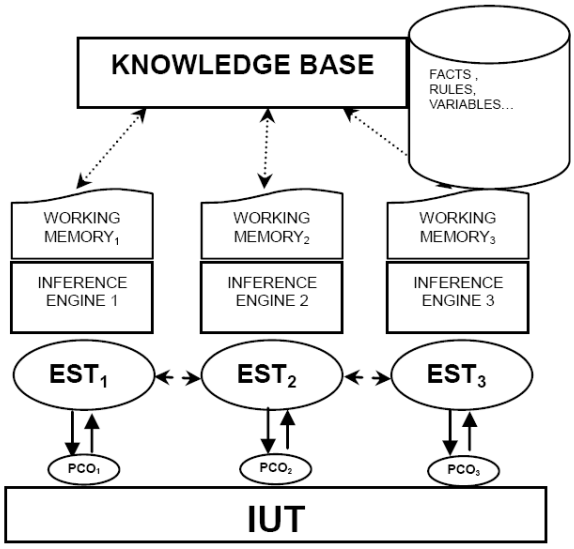

FIGURE 5. Architecture of the distributed test system

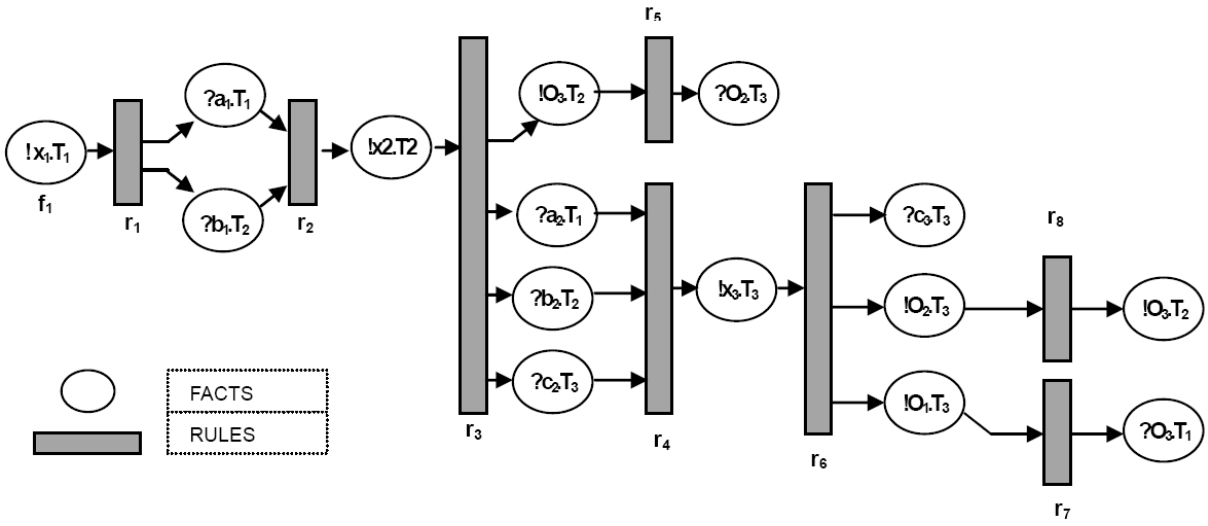

FIGURE 6. The Petri net representation associated to the global test sequence (4)

4.2. **Behavior of the test system.** A Petri net (also known as a place/transition net or P/T net) is one of several mathematical modeling languages for the description of distributed systems. A Petri net is a directed bipartite graph, in which the nodes represent transitions (i.e., events that may occur, signified by bars) and places (i.e., conditions, signified by circles) [18].

In our case, the places represent facts and transitions represent rules. The diagram above represents facts and rules deduced from the global test sequence "Equation (4)" by applying Algorithm 2 explained in the previous section.

Let the structures be defined as follows:

i A: matrix of antecedents, $Aij = 1$ if the fact fj is antecedent in rule ri else $Aij = 0$.
ii C: matrix of consequents, $Cij = 1$ if the fact fj is consequent in rule ri else $Cij = 0$.
iii M: The state (marking) of a Petri net is defined as follows:

$M : P\text{->} N$, i.e., a function mapping the set of places onto $\{0, 1, 2, \ldots\}$.

In our case, this function M is defined as: $M : F\text{->} \{0,1\}$, i.e., a function mapping the set of facts onto $\{0,1\}$. $M0$ is the initial state; $M0 = (1,0,0,0,0,0,0,0,0,0,0,0,0,0,0)$.

The matrices A and C of antecedents and consequents are defined for this case as:

$$A = \left\{ \begin{array}{cccccccccccccc} 1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\ 0&1&1&0&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&1&1&1&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&1&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&1&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&0&1&0&0 \end{array} \right\}$$

$$C = \left\{ \begin{array}{cccccccccccccc} 0&1&1&0&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&1&1&1&1&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&1&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&1&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&1&1&1&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&1 \end{array} \right\}$$

We denote $A(.,rj)$ (respectively $C(.,rj)$) the row associated to the rule rj in the matrix of antecedents A (resp. matrix of consequents C).

Sensitized rules: In a Petri net, a rule rj is sensitized for a marking M if and only if $M \geq A(.,rj)$. The $\geq$ is a vectors comparison and it will be done fact by fact as follows:

$$\forall f \in F, \quad M(f) \geq A(f, r_j) \tag{8}$$

In our example below, let us take a marking $M = (1,1,1,0,0,0,0,0,0,0,0,0,0,0,0)$ and calculate if the rules $r2$ and $r3$ are sensitized for this marking or not. We have: $A(.,r2) = (0,1,1,0,0,0,0,0,0,0,0,0,0,0,0)$ and $A(.,r3) = (0,0,0,1,0,0,0,0,0,0,0,0,0,0,0)$.

By comparing M with the rows bellow: $M \geq A(.,r2)$ and $A(.,r3) \geq M$, we can conclude that the rule $r2$ is sensitized for the marking M but the rule $r3$ is not.

Fired rules: In a Petri net, a sensitized rule rj for a marking M can be fired and the next marking M is defined as follows:

$$M = M - A(.,rj) + C(.,rj) \tag{9}$$

The marking vector M is composed by positive or null values because $M \geq A(.,rj)$ for the sensitized rule rj. In the example above **if the rule rj is fired the next marking will be** $M = (1,0,0,1,0,0,0,0,0,0,0,0,0,0,0)$.

As conclusion, when we have the facts and rules list represented in a matrix form (A,C) and the initial state of the system $M0$, we can then deduce using simple arithmetic operations the state of the system and decide if some rules can be enabled.

4.3. **Test procedure.** We describe in this section the test procedure:

(i) For sending an input to the IUT, the Expert System Tester (ESTi) checks the knowledge base to test if the rule is sensitized using the marking M.

(ii) When an ESTi apply an input to the IUT, the IUT sends some outputs messages to the concerned ESTj.

(iii) After receiving the outputs messages from the IUT, each ESTj check using its Inference Engine (IEj) and its Working (WMj) if the message received is the expected one.

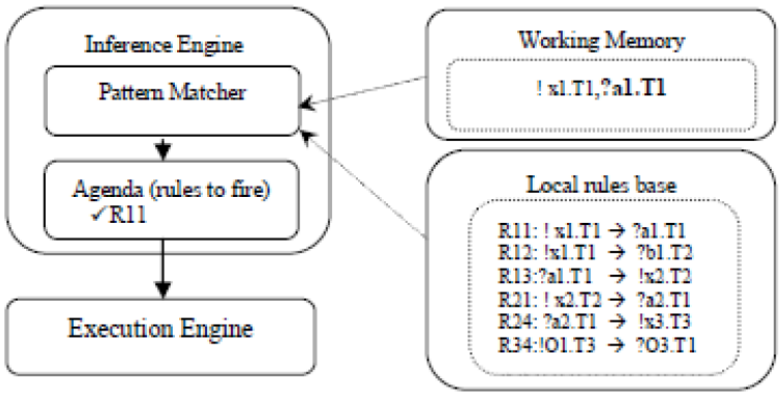

FIGURE 7. Structure of a tester ESTi

As shown in the description of the tester ESTi (Figure 7), the pattern matcher will match the local rules against the results received from other components of the test system.

a) If the result is OK => The local rule will be stored in the Agenda in order to notify the Knowledge Base (rule fired).
b) Else => Test Failed.

(iv) The rules Rij of testers concerned by validating a rule ri must be fired to decide if the next rule can be sensitized.

5. **Testing Carrying Out.** This section describes a testing example of a COBRA application to compare the both prototype approaches of the test. We notice that in CORBA, the terms client and server are used to specify the role played by a component in a distributed application and the object interface specifies only the operations and types that the objects supports. The application and its interfaces are briefly presented. Related local test sequences and rules are constructed from a complete test sequence, and then test results are given and discussed.

5.1. **Distributed chat group application.** The example to be tested and described below is based on a distributed chat group application. Actually, for sending a message to a chat group, the user has to join this group, and then the text message submitted by the user will be sent to a central server. The server then forwards these messages to other clients which have joined the same group. The role of the central server is to manage all messages sent to all chat groups. It receives the messages from client applications and forwards these messages to other clients appropriately.

The application has two interfaces:

- User interface allowing subscribers to post and to retrieve messages;
- Manager interface allowing one to add and to remove subscribers, to forward messages to the clients appropriately and to provide some statistical information about the total number of messages sent through the system for example.

We will use the Interface Definition Language (IDL) for the specification of these interfaces as it is given in Figure 8.

5.2. **Test execution.** To illustrate the test execution, we use the following test sequence Figure 9. This sequence describes the sending of a message by a user to other users who have joined the group through the manager.

```
interface User {
        /*Exceptions*/
         Exception can_not_sent{string reason;};
        Exception not_allowed{};
        Exception not_such_message{};
        /*Methods*/
        oneway void newMessage (in string msg);
        void  Deposit_message(in string msg)
              raises(can_not_sent);
        void ClearBuffer();
        void Client_connected();
       Message Get_Message(in Client c,in id_mess ref)
              raises(not_allowed,not_such_message);
        };

interface Manager {
        /*Exceptions*/
        Exception can_not_register{string reason};
        Exception non_such_client{};
        /*Methods*/
        oneway void registerClient (in CallBack objRef,in string name)
                    raises(can_not_register);
        oneway void removeClient (in CallBack objRef, in string name)
                    raises(non_such_client);
        oneway void SendMessage (in string msg);
        oneway void ReceiveMessage (in string msg);
        void Increment_nb_msg();
        void Connection_verified();
         };
```

FIGURE 8. IDL interfaces of the distributed chat group application

```
!U1 : Deposit_message(msg)
?U1 : ClearBuffer()
?M  : ReceiveMessage(msg)
!U2 : Client_connected()
?M : Connection_verified()
!M  : SendMessage(msg)
?M : Increment_nb_msg()
?U2 : NewMsg(msg)
```

FIGURE 9. Example of global test sequence

Manager	USER1	USER2
?IUT: Receive_message(msg)	!IUT: Deposit_message(msg)	?CC: Coordination_message(Sender)
?IUT: Connection_verified()	?IUT:ClearBuffer()	Sender== USER1
!CC: Observation_message(User2)	!CC: Coordination_message(User2)	!IUT: Client_connected()
!IUT: SendMessage(msg)		?CC: Observation_message(Sender)
?IUT: Increment_nb_msg()		Sender=MANAGER
		?IUT: New_message(msg)

FIGURE 10. Local test sequences

Local test sequences corresponding to this global test sequence are given above (Figure 10). Notice that a coordination message is sent by the user 'User1' to the user 'User2' to coordinate the sending of a message from 'User1' to 'User2'.

Each tester uses two interfaces: IUT, related to the IUT interface associated with the tester, and CC related to the interface with the coordination channel. These sequences come from testing experiments with the prototype on students' realizations of chat group application.

On the other hand and according to the algorithms explained in the previous sections, the local/global rules corresponding to the situation explained above are given as follows:

R_{11} : !Deposit_message(msg).U1 → ?ClearBuffer().U_1
R_{12}: !Deposit_message(msg).U1 → ?ReceiveMessage(msg).M
R_{13}: ?ClearBuffer().U_1 → !Client_connected().U_2
R_{14}: ?ReceiveMessage(msg).M → !Client_connected().U_2
R_{21}: !Client_connected().U_2 → ?Connection_verified.M
R_{22}: ?Connection_verified.M →!SendMessage(msg).M
R_{31} : !SendMessage(msg).M→?NewMsg(msg).U_2
R_{32}: !SendMessage(msg).M→?Increment_nb_msg().M
R_{33}: !SendMessage(msg).M → !O^2.M
R_{34}: !O^2.M → ?O^3. U_2

FIGURE 11. Local rules

r_1 : !Deposit_message(msg).U_1 → ?ClearBuffer().U_1 ^ ?ReceiveMessage(msg).M
r_2 : ?ClearBuffer().U_1 ^ ?ReceiveMessage(msg).M → !Client_connected().U_2
r_3 : !Client_connected().U_2 → ?Connection_verified.M
r_4 : ?Connection_verified.M → !SendMessage(msg).M
r_5 : !SendMessage(msg).M → ?NewMsg(msg).U_2 ^ ?Increment_nb_msg().M ^ !O^2.M
r_6: !O^2.M → ?O^3. U_2

FIGURE 12. Global rules

To illustrate more this example, we use the petri net representation for the facts and the global rules.

The matrices A and C of antecedents and consequents corresponding to this case are defined as:

$$A = \begin{Bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{Bmatrix}$$

$$C = \begin{Bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{Bmatrix}$$

Flow diagram. Let F and R the lists of facts and global rules respectively be deduced from the global sequence test of the distributed chat group application and $M0$ the initial marking.

Since $M0$ is the initial state, and as described in the diagram Figure 14, the tester EST2 will apply input "!Deposit_message(msg)" to the IUT. Then $r1$ is fired and the marking

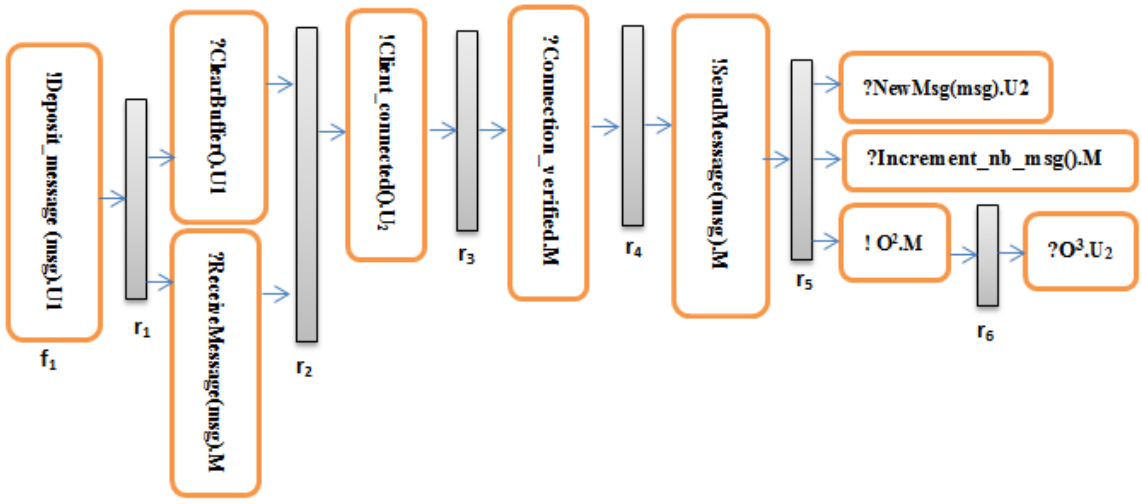

FIGURE 13. The Petri net representation associated to the example

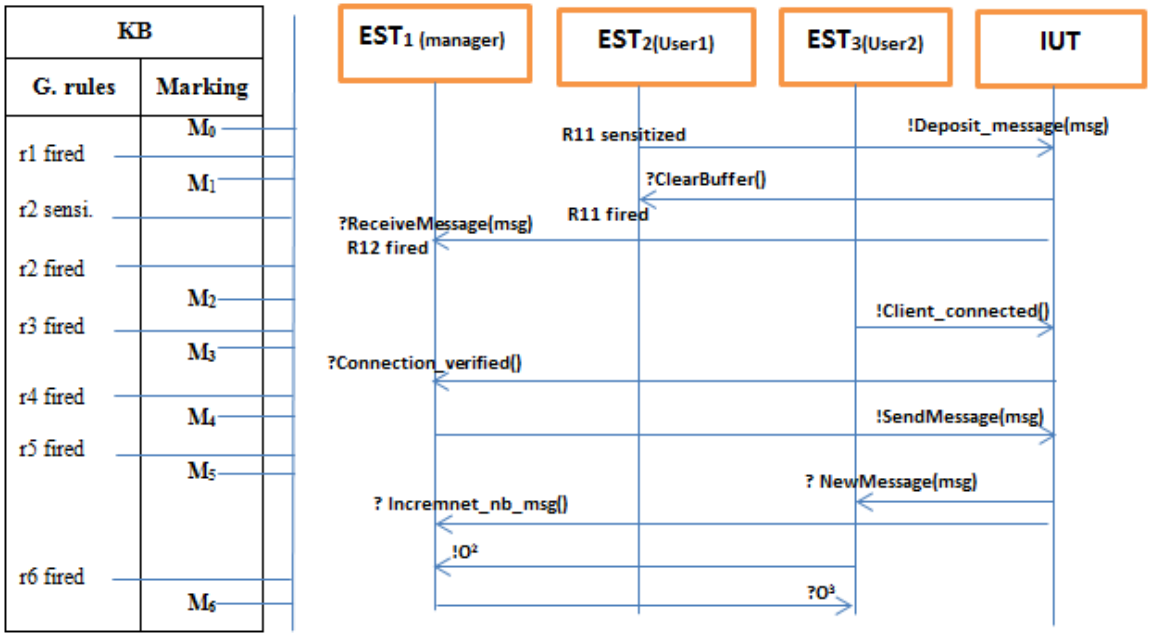

FIGURE 14. The flow's diagram for exchanges between ESTi and IUT

will be $M1$. $M1 = M0 - A(., r1) + C(., r1)$; $M1 = (0, 1, 1, 0, 0, 0, 0, 0, 0, 0)$. While other testers receive results – induced by applying the input "!Deposit_message(msg)" – EST2 checks if it receives the "?ClearBuffer()" and EST1 checks if "?ReceiveMessage(msg)" is received too. If so, the local rules are fired. Else the test fails. While all local rules $(R11, R12)$ participating in the global rule $r2$ are fired then the global rule $r2$ is fired then the marking is updated to $M2 = M1 - A(., r2) + C(., r2)$; $M2 = (0, 0, 0, 1, 0, 0, 0, 0, 0, 0)$.

The Tester3 can then apply the input "!Client_connected()" to the IUT. If so, the rule $r3$ is fired and the marking is updated to $M3$; $M3 = (0, 0, 0, 0, 1, 0, 0, 0, 0, 0)$. Else the test fails.

Afterwards, the tester1will receive a message "?Connection_verified()" from the IUT and as the previous case the rule $r4$ is fired and the marking is updated to $M4$; $M4 = (0, 0, 0, 0, 0, 1, 0, 0, 0, 0)$. Else the test fails.

Once the verification of the client connection is done by the tester1 (manager), it will apply the input "!SendMessage(msg)" to the IUT then the rule $r5$ is fired and the marking is updated to $M5$; $M5 = (0, 0, 0, 0, 0, 0, 1, 1, 1, 0)$. Else the test fails.

Finally, the tester3 will receive the input "?NewMeessage(msg)" and the tester1 (manager) receive the input "?Incremnet_nb_msg()" and an observation message will be sent in this case by the tester3 to the tester1. If so the rule $r6$ is fired and the marking is updated to $M6$; $M6 = (0, 0, 0, 0, 0, 0, 1, 1, 0, 1)$. Else the test fails.

Finally, by introducing the concept of rules based Expert Systems we propose in this article another way to overcome the coordination/synchronization problems. As explained, the main motivation of the practical use of the results developed is to avoid the use of the coordination messages and resolve the coordination problems. Thus, the testers will exchange only the observation messages which will reduce significantly the use of external messages and I/O operations. In fact, as shown in the previous diagram, each tester executes only a part of the global reasoning, and diffuses through the network the obtained results. By the way, other testers can use these results to participate in the global reasoning.

6. **Conclusion.** As mentioned in the related works above, many researches have been made to coordinate testers and by the way to minimize or to eliminate the use of coordination messages because such messages can introduce delays especially if there are some timing constraints. This article presents an architecture, a model and a method of distributed test that guarantee the principles of coordination and synchronization between various components of the distributed test platform. In fact, compared with other works which attempt to deduce local test sequences from the global one and including some coordination and observation messages to ensure coordination between testers, we suggest in this paper to deduce local rules to be fulfilled by each tester to guarantee coordination between them. The work presents a way to avoid the exchange of the external coordination messages among remote testers during the test. As explained, this has done by introducing the notions of rule based expert systems to propose a prototype of test. The petri nets formalism in the matrix form is used to introduce the firing and sensitizing notions and to calculate the state of the system by referring to the marking vector. The implementation of this approach by writing the inference engine using the Prolog formalism, and testing web services applications are the perspectives of our approach.

REFERENCES

[1] A. Gill, *Introduction to the Theory of Finite-State Machines*, Mc Graw-Hill, New York, USA, 1962.
[2] A. Petrenko, G. V. Bochmann and M. Yao, On fault coverage of tests for finite state specifications, *Computer Networks and ISDN System*, vol.29, pp.81-106, 1996.
[3] O. Rafiq and L. Cacciari, Coordination algorithm for distributed testing, *The Journal of Supercomputing*, vol.24, no.2, pp.203-211, 2003.
[4] A. Khoumsi, A temporal approach for testing distributed systems, *IEEE Transactions on Software Engineering*, vol.28, no.11, pp.1085-1103, 2002.
[5] M. Benattou, L. Cacciari, R. Pasini and O. Rafiq, Principles and tools for testing open distributed *Proc. of the IFIP TC6 the 12th International Workshop on Testing Communicating Systems, Method and Applications*, pp.77-92, 1999.
[6] W. Liu, H. Zeng and H. Miao, Multiple UIO-based test sequence generation for distributed systems *Journal of Shanghai University (English Edition)*, vol.12, no.5, pp.438-443, 2007.
[7] J. Chen, R. M. Hierons and H. Ural, Testing in the distributed test architecture formal methods and testing, *Lecture Notes in Computer Science*, vol.4949, pp.157-183, 2008.
[8] R. M. Hierons and H. Ural, Checking sequences for distributed test architectures, *Distributed Computing*, vol.21, no.3, pp.223-238, 2008.

[9] K. C. Tai and Y. C. Young, Synchronizable test sequences of finite state machines, *Computer Networks*, vol.13, pp.1111-1134, 1998.

[10] G. Luo, R. Dssouli and G. v. Bochmann, Generating synchronizable test sequences based on finite state machine with distributed ports, *The 6th IFIP Workshop on Protocol Test Systems*, pp.139-153, 1993.

[11] R. M. Hierons, Testing a distributed system: Generating minimal synchronized test sequences that detect output-shifting faults, *Information and Software technology*, vol.43, no.9, pp.551-560, 2001.

[12] G. Luo, R. Dssouli, G. v. Bochmann, P. Venkatram and A. Ghedamsi, Test generation with respect to distributed interfaces, *Computer Standards & Interfaces*, vol.16, no.2, pp.119-132, 1994.

[13] J. Chen, R. M. Hierons and H. Ural, Conditions for resolving observability problems in distributed testing, *The 24th IFIP International Conference on Formal Techniques for Networked and Distributed Systems (FORTE 2004), LNCS*, vol.3731, pp.229-242, 2004.

[14] L. Cacciari and O. Rafiq, Controllability and observability in distributed testing, *Information and Software Technology*, vol.41, pp.767-780, 1999.

[15] J. Chen, R. M. Hierons and H. Ural, Resolving observability problems in distributed test architecture, *The 25th IFIP International Conference on Formal Techniques for Networked and Distributed Systems (FORTE 2005), LNCS*, vol.3731, pp.219-232, 2005.

[16] O. Rafiq, L. Cacciari and M. Benattou, Coordination issues in distributed testing, *Proc. of the 5th International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA'99*, USA, pp.793-799, 1999.

[17] M. E. H. Charaf, M. Benattou, S. Azzouzi and J. Abouchabaka, Using an ontology for modeling the communication in the distributed test, *IEEE Proc. of the 3rd International Conference on Web and Information Technologies ICWIT'10*, Marrakech, Morocco, pp.321-334, 2010.

[18] J. L. Peterson, *Petri Net Theory and the Modeling of Systems*, Prentice Hall, Inc, Englewood Cliffs, 1981.

[19] M. E. H. Charaf, M. Benattou and S. Azzouzi, A rule-based multi-agent system for testing distributed applications, *IEEE Proc. of the 3rd International Conference on Multimedia Computing and Systems (ICMCS'12)*, pp.967-972, 2012.

[20] M. E. H. Charaf, M. Benattou and S. Azzouzi, An expert system approach for distributed testing, *Proc. of the 7th International Conference on Intelligent Systems: Theories and Applications (SITA'12)*, pp.16-17, 2012.

[21] S. Azzouzi, M. Benattou and M. E. H. Charaf, Real time agent based approach for distributed testing, *IEEE Proc. of the 3rd International Conference on Multimedia Computing and Systems (ICMCS'12)*, pp.10-12, 2012.