The paper presents some technical issues for testing distributed frameworks with rules based Agent System. The proposed approach consists firstly on exploring the benefits of rule based systems to design communication between different components of the distributed test application and then it deals with the distribution of rules over the testers to ensure their coordination without the need of a centralized manager system. The presented work describes our algorithm allowing the generation of the local rules to be respected by each tester and how we can use JESS in a multi-agent system. We compared also our results to those obtained in the centralized approach, where the antecedents of a rule can be in different testers, thus the tester can’t decide when to enable the rule and that we will need a shared knowledge base to manage such situations.

Keywords: agents, distributed test, rules, controllability and observability problems, synchronization

1. INTRODUCTION

In the distributed testing context, the use of multiple testers introduces the possibility of coordination problems amongst remote testers [1-5]. These potential problems are known as controllability and observability fault detections which are fundamental features of conformance in distributed testing. In such context, 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 set of errors: time outs, locks, channels and network failures.

To avoid these problems, many works [3, 5, 14, 16] propose to introduce some coordination messages which lead each tester to determine when to apply a particular input to the IUT (Implementation Under Test) and whether a correct output from the IUT is generated in response to a specific input, respectively. However, 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 (Finite State Machine) under which controllability and observability problems can be overcome without using external coordination messages [13, 15]. In [8], authors show 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.

Our work is mainly based on the algorithms proposed in [3, 22, 28] for writing test coordination procedures in a distributed testing architecture. It can be considered as a continuity of our works [17, 19, 20, 22, 27] where we propose to introduce some concepts issued from Artificial Intelligence to overcome such problems. The idea is to write the rules to be respected by the testers to guarantee their coordination. Compared to the approaches cited in [11-15], the objective of introducing rules is to ensure coordination between testers using the artificial intelligence features. The basic idea behind introducing the concept of rules 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. For this purpose, we suggest a rule based systems that provide an interesting approach for representing and interpreting such kind of messages exchange. These systems permit 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 which increase significantly the flexibility of the test system implemented.

Therefore, we notice as detailed in the centralized approach [20, 27] that the tester can’t decide when to enable the rule because the antecedents of a rule can be in different testers. These results lead us to use a shared knowledge base to manage such situations. However, the system control in this case is entirely driven by the knowledge base state and when a large number of rules are involved then understanding the interactions between multiple rules affected by the same facts become very difficult and hard to manage which reduce significantly the execution performance. Also, the system of the test in such approach should include additional backup mechanisms of the shared knowledge base in case of breakdown. In this context, the proposed rule generation technique in this paper deals with the distribution of rules over the testers to ensure coordination in the test system without requiring a shared knowledge base. For this purpose, we introduce the synchronization messages to guarantee the ordering of the events over the rules. In addition, we point out that even if the introduction of synchronization messages will cause the performance reduction, it will face up-on the other hand-to both the coordination and synchronization issues. In addition, we propose in this paper a multi-agent architecture with distributed testing rule generation algorithm that can be used for developing automated testing systems of distributed applications with highly flexibility when compared to systems with global rules in the centralized approach. More precisely, expert systems as described in the proposed prototype [19, 20, 27] are generally not required to be capable of cooperating flexibly with other expert systems. To this end, we introduce in this paper multi-agent systems that are ideally suited to represent problems that have multiple problems solving entities. Such systems have the traditional advantages of distributed and concurrent problem solving strategies. Also, by embedding JESS in JADE agents, we aim to improve the ability of the agents in regard to conflict resolution using negotiation skills by incorporating more advanced negotiation rules.

This paper presents some technical issues for testing distributed frameworks with
rules based Agent System. The proposed approach consists firstly on exploring the benefits of rule based systems to design communication between different components of the distributed test application and then it deals with the distribution of rules over the testers to ensure their coordination without the need of a manager system. To this end, the paper is organized as follow: The second and the third section describe the concept of distributed testing, architecture, the test procedure while referring to the problems of coordination/synchronization and some related works. The fourth section introduces an algorithm to generate the rules that will be used to avoid the coordination/synchronization problems and the last section presents our prototype of test using rule-based agents.

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) and communicates with other testers through a multicast channel (Fig. 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.

![Test architecture](image)

**Fig. 1.** Test architecture.

2.2 Modeling by Automaton

To approach the testing process in a formal way, the specification and the Implementation Under Test 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. We denote \( \Sigma_k \) the input alphabet of the port \( k \) (PCO number \( k \)) and \( \Gamma_k \) the output alphabet of the port \( k \). Fig. 2 gives an example of \( 3p \)-FSM with \( Q = \{ q_0, q_1, q_2, q_3 \} \), \( q_0 \) 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: \( x_1?y_1!x_2?y_2!x_t?y_t \) that for \( i = 1, \ldots, t \), \( x_i \in \Sigma = \Sigma_1 \cup \cdots \cup \Sigma_n \) with \( \Sigma_i \cap \Sigma_j = \emptyset \) for \( i \neq j \), \( y_i \in \Gamma_k \) and for each port \( k \) \( |y_i \cap \Gamma_k| \leq 1 \). We denote ‘!*x_i’ sending the message \( x_i \) to IUT and ‘?y_i’ the re-
ception of messages belonging to the \( y \) from the IUT. An example of a test sequence of 3p-FSM illustrated in Fig. 2 is:

\[
{x_1}^? \{a_1, b_1, \epsilon\}!x_2^? \{a_2, b_2, c_2\}!x_1^? \{a_2, b_1, c_2\}!x_3^? \{a_1, b_3, \epsilon\}!x_1^? \{a_1, \epsilon, c_3\}!x_3^? \{\epsilon, b_1, c_3\}
\]

(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]. 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.

\[\text{Fig. 2. An example of 3p-FSM.}\]

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:** It 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’s the capability of the test system to realize input events at corresponding PCOs in a given order.

**Observability problem:** It 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, it arises when two consecutive transition \((i)\) and transition \((i+1)\) occur 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 Fig. 3.

The projections of (1) on ports alphabets are required to get the test sequence related to each tester. Projections \(w_1, w_2, w_3\) for testers \(T_1, T_2, T_3\) of the global test sequence
are:

\[ w_1 = !x_1 ? a_1 !a_2 !x_1 ? a_2 !a_1 !x_1 ? a_1, \quad w_2 = ?b_1 !x_2 ? b_2 ? b_1 ? b_2 ? b_1 \text{ and } w_3 = ?c_2 !x_3 ? c_2 !x_3 ? c_3. \]

By applying these sequences to the IUT of Fig. 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 \( T_1 \) receives \( a_1 \) before \( a_2 \) and it detects an output fault. However, if the IUT decides to read \( x_3 \) before \( x_1 \) 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.

![Fig. 3. Example of a faulty IUT for Fig. 2.](image)

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):

\[
\begin{align*}
    w_1 &= !x_1 ? a_1 !a_2 !x_1 ? a_2 ? a_1 !O_{1,2,3} !x_1 ? a_1 ? O_{1}, \\
    w_2 &= ?b_1 !O_{1} !x_2 ? b_2 ? b_1 ? C_1 ? b_3 ? O_{2}? b_3, \\
\end{align*}
\]  

(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:

- \(+C\{t_1, ..., t_r\}\) resp.) the sending of a coordination message (observation message resp.) to the testers \( t_1, ..., t_r \).
- \(+O\{r\}\), resp.) the receipt of a coordination message (observation message resp.) from the tester \( t \).
For example, the tester $T_3$ in the transition $!x_3?\{a_1, b_3, e\}$ is neither the one sending $x_1$, nor the one of those receiving a message belonging to $\{a_2, b_1, e\}$, we add then a coordination message ($!C_3$) to the sequence of the tester $T_2$ receiving a message belonging to $\{a_2, b_1, e\}$ and ($?C_3$) to the sequence of the tester $T_3$ that send the message $x_3$.

Also, the tester $T_3$ in the transition $!x_2?\{a_2, b_2, c_2\}$ will receive a message $c_2$ even if it doesn’t receive any message belonging to $\{a_1, b_1, e\}$, after that tester $T_1$ sends the message $x_1$ and it is not the tester that will send $x_2$. So we will introduce an observation message ($!O_3$) to the sequence of the tester $T_2$ and ($?O_2$) to the sequence of the tester $T_3$.

**Synchronization Problem**: As explained above, the algorithm in [3] allows the generation of local test sequences to be performed by each tester. Then, each tester is running its local test sequence. Thus, the testers are working together but independently, which leads us to manage the problem of synchronization of testers.

Let us execute the first fragments of the local test sequences obtained in Eq. (2):

\[
\begin{align*}
w_{f1} &= !x_1? a_1, \\
w_{f2} &= ?a_1 !O_1 !x_2, \\
w_{f3} &= ?O_2 ?c_2.
\end{align*}
\] (3)

Running the fragments $w_{f1}, w_{f2}$ and $w_{f3}$ should give the result shown in Fig. 4 (a) but the execution of our prototype provides an incorrect result given in Fig. 4 (b).

Indeed, in last diagram the second tester sends the message $x_2$ to 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 Eq. (1), where the message $x_2$ must be sent only if all messages due to the sending of $x_1$ by the tester $T_1$ are received by the IUT.

In the following of this paper, we will take – for simplicity, the test sequence of $3p$-FSM shown in Fig. 2 defined as:

\[
!x_1?\{a_1, b_1, e\}!x_2?\{a_2, b_2, c_2\}!x_3?\{e, c, c_3\}.
\] (4)

### 3. RELATED WORKS

Many works has been made to avoid the problems described in the previous section. Indeed, the author in [4] shows that controllability and observability are 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 overcome 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 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]. Our work is mainly based on [5, 16] and the algorithms proposed in [3, 22] for writing test coordination procedures in a distributed testing architecture. It can be considered as a continuity of [17, 19, 20, 22, 27] where we propose to introduce some concepts issued from Artificial Intelligence to overcome such problems.

Unlike [20, 27], where we introduce our algorithms allowing the generation of the global rules to be respected by the whole system and explain how such systems can avoid the use of the coordination and by the way, only observation messages will be exchanged by testers which will reduce significantly the use of external messages and I/O operations, we suggest in this article our solution by introducing another algorithm to generate rules to be implemented in each tester. To this end, we refer to results obtained in [22, 28] where we introduce another kind of messages called synchronization messages to achieve the distribution of rules over of testers. 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.
4. TESTING RULES

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.

4.1 Testing Rules Production

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

- If the tester $T_1$ send a message $x_1$ to the IUT ($!x_1.T_1$) then the tester $T_1$ will receive a message $a_1$ from the IUT ($?a_1.T_1$) and the tester $T_2$ will receive a message $b_1$ from the IUT ($?b_1.T_2$).
- If the message $a_1$ is received in the tester $T_1$ ($?a_1.T_1$) and the message $b_1$ is received in the tester $T_2$ ($?b_1.T_2$). Then the tester $T_2$ will apply the message $x_2$ to the IUT ($!x_2.T_2$). At this stage, we have an observability problem because the tester $T_3$ will receive a message $c_2$ ($?c_2.T_3$) even if it doesn’t receive any message after that tester $T_1$ send the message $x_1$ ($!x_1.T_1$) and it is not the tester that will send $x_2$ ($!x_2.T_3$). So we will introduce an observation message $O_3$ to be sent by tester $T_3$ to the tester $T_3$. In this case, the next rule is as follow:
- If the tester $T_2$ send a message $x_2$ to the IUT ($!x_2.T_2$) then the tester $T_1$ will receive a message $a_2$ from the IUT ($?a_2.T_1$) and the tester $T_2$ will receive a message $b_2$ from the IUT ($?b_2.T_2$) and the tester $T_3$ will receive a message $c_2$ from the IUT ($?c_2.T_3$) and the tester $T_2$ will send an observation message $O_3$ to tester $T_3$ ($!O_3.T_2$).

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

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

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:

- $!x_1.T_1 \rightarrow ?a_1.T_1; !x_1.T_1 \land ?b_1.T_2;$
• $a_1.T_1 \land b_1.T_2 \rightarrow !x_2.T_2$;
• $!x_2.T_2 \rightarrow ?a_2.T_1 \land b_2.T_2 \land c_2.T_2 \land !O_3.T_2$.

In the centralized approach presented in a previous paper [27], we explained algorithms to generate such global/local rules then we detailed the need of a manager (global knowledge base) to take decisions and deduced the test verdict. To this end, we explain using the Petri Net formalism [18] how using facts and rules list represented in a matrix form $(A, C)$ and the initial marking of the system $M_0$, we can then deduce using simple arithmetic operations the state of the system and decide if some rules can be enabled.

The main advantage of such systems is that we can avoid the use of the coordination messages and by the way, only observation messages will be exchanged by testers which will reduce significantly the use of external messages and I/O operations. In other hand, the need of a manager system decreases the performance of the test system (all testers check out from that source). In this context, this paper deals with the distribution of rules over testers by introducing some synchronization messages. Thus, we will generate distributed rules related to our testers from a Global test sequence of the IUT to ensure their coordination without needing a manager system.

4.2 Testing Rules in the Distributed Approach

In this section, we describe the algorithm allowing the generation of the local rules to be respected by the testers to avoid the coordination/synchronization problems. The algorithm is inspired from works [3, 19, 22, 27, 28]. Contrary to their works [3] that generate local test sequences from the global test sequence and that introduce coordination and observation messages and as continuity of [19, 20, 27], the proposed algorithm deals with the distribution of rules over testers by introducing some synchronization messages [22, 28]. In this context, Algorithm1 will generate distributed rules related to our testers from a Global test sequence of the IUT.

We denote by ‘\-' the set difference and ‘$\Delta$’ the symmetrical difference: $A \Delta B = (A \setminus B) \cup (B \setminus A)$. The function Port gives the port corresponding to a given message. For a set $y$ of messages, the function Ports is defined by: $\text{Ports}(y) = \{ k | a \in y \text{ s.t. } k = \text{Port}(a) \}$.

Let us take the global test sequence defined in (4) as the input of the algorithm above. The algorithm generates a list of local rules by browsing the ‘$t$’ messages to be sent to the IUT (line2). Then, the local rules will be constructed as:

• Each message ‘$m$’ belonging to $y$ is a part of a rule as a consequence of sending message $x_i$ (lines 10, 20).
• Each message ‘$m$’ belonging to $y$ is a part of a rule as an antecedent of sending message $x_{i+1}$ (lines 32, 38).

To avoid observation problems, each tester receiving a message $j \in y_{i-1}$ should be able to determinate that $j$ has been sent by IUT after IUT has received $x_{i-1}$ and before IUT receives $x_i$ (lines 14, 15). Afterwards, we integrate the observation messages into local rules for avoiding this problem (lines 17, 44, 50). To avoid Synchronization problems, each tester receiving $j \in y$, send a synchronization message to a tester sending $x_{i-1}$ (lines 24, 29). In this case, we integrate the synchronization message into local rules (lines 25, 32).
Algorithm 1

Begin

1. \( m = 1 \)

2. for \( i = 1, \ldots, t \) do

3. \( k = \text{Port}(x_i) \)

4. \( l = \text{Port}(x_{i+1}) \)

5. \( \text{RR} = e \)

6. \( \text{RL} = e \)

7. \( \text{OBSERV} = \text{false} \)

8. for all \( j \) belong to \( y_i \) do

9. if \( (j << e) \)

10. \( \text{RR} = \text{RR} \land \text{?j.Tport}(j) \)

11. end if

12. end for

13. if \( i > 1 \) then

14. Send_To \( \subseteq \) (Ports(y_j) \( \Delta \) Ports(y_{i-1})); \{k\}

15. if Send_To \( \subseteq \emptyset \) then

16. \( \text{OBSERV} = \text{true} \)

17. \( \text{RR} = \text{RR} \land O_{\text{send Tport}}T_j \)

18. end if

19. end if

20. \( r_m: x_i.T_i \rightarrow \text{RR} \)

21. \( m = m+1 \)

22. if \( i < t \)

23. for all \( j \) belong to \( y_j \) do

24. if \( \text{port}(j) \subseteq j \) then

25. \( r_m: j.T_{\text{portadj}}(j) \rightarrow \text{?O_k.T_{portadj}}(j) \]

26. \( m = m+1 \)

27. end if

28. end for

29. for all \( j \) belong to \( y_j \) do

30. if \( (j << e) \)

31. \( \text{RL} = \text{RL} \land \text{?j.Tport}(j) \)

32. end if

33. end for

34. end for

35. \( \text{end if} \)

36. end if

37. end for

38. \( m = m+1 \)

39. end if

40. end if

41. if OBSERV = \text{true} then

42. for all \( j \) belong to \( y_j \) do

43. if \( \text{port}(j) \subseteq j \) then

44. \( r_m: j.T_{\text{portadj}}(j) \rightarrow \text{?O_k.T_{portadj}}(j) \]

45. \( m = m+1 \)

46. end if

47. end for

48. \( \text{end if} \)

49. end for

50. \( \text{end if} \)

51. end if

52. end if

53. end if

54. end if

55. End for

End

Table 1. The distributed rules list deduced from (4).

<table>
<thead>
<tr>
<th>Rules</th>
<th>Tester</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r_1: x_1.T_1 \rightarrow a_{12}.T_1 )</td>
<td>( T_i )</td>
</tr>
<tr>
<td>( r_2: ?a_1.T_1 \rightarrow S_{12}.T_1 )</td>
<td>( T_i )</td>
</tr>
<tr>
<td>( r_3: ?S_{12}.T_1 \rightarrow h_{12}.T_2 )</td>
<td>( T_i )</td>
</tr>
<tr>
<td>( r_4: x_{12}.T_2 \rightarrow a_{12}.T_1 )</td>
<td>( T_i )</td>
</tr>
<tr>
<td>( r_5: ?a_{12}.T_1 \rightarrow S_{12}.T_1 )</td>
<td>( T_i )</td>
</tr>
<tr>
<td>( r_6: h_{12}.T_2 \rightarrow S_{12}.T_2 )</td>
<td>( T_i )</td>
</tr>
<tr>
<td>( r_7: ?S_{12}.T_1 \rightarrow c_{12}.T_2 )</td>
<td>( T_i )</td>
</tr>
<tr>
<td>( r_8: ?c_{12}.T_2 \rightarrow ?O_{12}.T_2 )</td>
<td>( T_i )</td>
</tr>
<tr>
<td>( r_9: ?x_{12}.T_2 \rightarrow c_{12}.T_2 )</td>
<td>( T_i )</td>
</tr>
<tr>
<td>( r_{10}: ?a_{12}.T_1 \rightarrow ?O_{12}.T_2 )</td>
<td>( T_i )</td>
</tr>
<tr>
<td>( r_{11}: ?h_{12}.T_2 \rightarrow ?O_{12}.T_2 )</td>
<td>( T_i )</td>
</tr>
</tbody>
</table>

Therefore, by applying the algorithm 1 above to the global test sequence defined in our example (4), the obtained rules \( r_1 \) are defined as shown in Table 1.

Compared to results obtained in the centralized approach (Table 2), where the antecedents of a rule can be in different testers, thus the tester can’t decide when to enable
Table 2. The rules list of (4) as generated in centralized approach.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r_1</td>
<td>!x_1.T_1 \rightarrow ?a_1.T_1 \land ?b_1.T_2</td>
</tr>
<tr>
<td>r_2</td>
<td>?a_1.T_1 \land ?b_1.T_2 \rightarrow !x_2.T_2</td>
</tr>
<tr>
<td>r_3</td>
<td>!x_2.T_2 \rightarrow ?a_2.T_2 \land ?b_2.T_2 \land ?c_2.T_3 \land !O_3.T_3</td>
</tr>
<tr>
<td>r_4</td>
<td>?a_2.T_2 \land ?b_2.T_2 \land ?c_2.T_3 \rightarrow !x_3.T_3</td>
</tr>
<tr>
<td>r_5</td>
<td>!O_3.T_3 \rightarrow ?O_2.T_3</td>
</tr>
<tr>
<td>r_6</td>
<td>!x_3.T_3 \rightarrow !O_3.T_3</td>
</tr>
<tr>
<td>r_7</td>
<td>!O_1.T_3 \rightarrow ?O_3.T_3</td>
</tr>
<tr>
<td>r_8</td>
<td>!O_2.T_3 \rightarrow ?O_3.T_3</td>
</tr>
</tbody>
</table>

the rule, that we will need a shared knowledge base to manage such situation [19, 20, 27].

However, the introduction of the synchronization messages increases significantly the number of rules needed to ensure the distribution over the testers.

5. MULTI AGENT TESTING PROTOTYPE

5.1 Overview

The proposed prototype to implement the obtained rules in the centralized approach [20, 27] was the use of expert systems connected to a global KB as described below:

![Fig. 5. Architecture of the Distributed Test System using the centralized approach.](image)

However, by referring the definitions cited in [23], the author argue that the expert systems are disembodied, they do not act on any environment but instead give feedback or advice to a third party. In addition, expert systems are generally not required to be capable of cooperating with other expert systems. To this end, we introduce the notion of the rule based multi agent system.

As explained in [24], MAGS Sy System [25] is an example of rule based MAS. It is
composed by some agents where the kernel is a forward-chaining rule interpreter. Therefore, each agent has the problem solving capacity of an expert system. The knowledge of the agents is structured in an object-oriented knowledge representation scheme. There is a global knowledge base which contains the knowledge that may be accessed by all of the agents. Agents may store their identification in this global knowledge base and thus become known to all agents in the system. Such architecture can be well adapted to our case but, due to the lack of examples and documentation, we will use Jess as a decision component of JADE agent to design our prototype in the distributed context.

5.2 Architecture

As we shown in Fig. 6, the complex tasks of JADE testers’ agents are delegated to well define specialized agent:

- The **Execution Agent** is used for control and observation of events on each PCO. It allows tester to apply input event to IUT and to receive output results from the IUT.
- The **Observation Agent** is used for sending and receiving observation messages.
- The **Synchronization Agent** is used for exchanging synchronization messages.
- The **JESS Agent** interfaces with JESS to see if rule/rules are matched.

The Jess agent uses an expert system to manage internal states. It maps the semantic of FIPA performatives to basic JESS functions. Moreover, the commands communicated in the content of an FIPA ACL are executed by the JESS engine.

As shown in Fig. 7, we define the Jess agent’s behavior of our distributed testing platform. Actually, each jess agent executes the test as follows: The Jess agent has two fundamental parts, the packet sniffer and the rule engine. The packet sniffer captures the packets and parse data to be used as input information for the rule engine. The rule engine carries out matching between the data model and the stored rules. The result of the execution of the selected rules is a set of actions that are sent to the concerned agents.
6. CONCLUSION

Over the last several decades, there have been important developments to overcome problems of coordination in the distributed testing field. In this paper, we proposed our prototype by introducing a rule based system to resolve the coordination/synchronization issues in the distributed testing frameworks. Firstly, in term of flexibility and compared to works cited in [11-15] we notice that such systems permit the implementation of highly flexible systems capable of adapting themselves to different situations. In fact, these systems are able to take decisions concerning possible malfunctions and decided if the process of test returns a failed verdict or an accepted one which increase significantly the flexibility of the distributed test.

On the other hand, we emphasize when comparing this work to the centralized approach [20, 27] that the introduction of the synchronization messages aims to guarantee the ordering of the events over the rules and by the way to ensure the distributions of the rules over the testers which will improve the execution performance. In fact, the results obtained in [27] show that the system control is entirely driven by the knowledge base state and according to experiments when a large number of rules are involved then understanding the interactions between multiple rules affected by the same facts become very difficult and hard to manage which reduce significantly the execution performance.

In addition, we propose in this paper a multi-agent architecture with distributed testing rule generation algorithm that can be used for developing automated testing systems of distributed applications when compared to systems with global rules in the centralized approach. More precisely, expert systems as described in the proposed prototype [27] are generally not required to be capable of cooperating flexibly with other expert systems. To this end, we introduce in this paper multi-agent systems that are ideally suited to represent problems that have multiple problems solving entities. Such systems have the traditional advantages of distributed and concurrent problem solving strategies. Also, by embedding JESS in JADE agents, we aim to improve the ability of the agents in regard to conflict resolution using negotiation skills by incorporating more advanced negotiation rules.
As prospects of our future works, we point out that the study in this paper evaluated only if the output events were observed in the different components of the test system without considering the times at which they were produced. Therefore, it would be interesting to evaluate our approach for implementations with timing constraints and define the temporal conditions to ensure coordination. Thus, the algorithm proposed in this paper could be adapted to generate rules taking into account these new constraints. In this context, we proposed also to extend our study using extended timed automata or timed Petri nets. Finally, our prototype realization of this model is under experimentation in Java environment. We have chosen Jess as a decision component of our Jess agents to implement our prototype testing.

REFERENCES

My El Hassan Charaf received the State Engineer degree in Computer Science from the National Institute of Statistics and Applied Economics (INSEA), Morocco, in 2004 and later, he received the Ph.D. degree in Computer science from the IBN TOFAIL University, Morocco. He is currently a Service Head of Information Systems within the Moroccan Ministry of Equipment and Transport and a member of the Laboratory of Research in Computer Science and Telecommunication (LARIT) at the IBN TOFAIL University. His research interests include: distributed testing, problems of controllability/observability and distributed artificial intelligence.

Mohammed Benattou is a Professor of Computer Science at the IBN TOFAIL University – KÉNITRA where he directs the Computer Science and Telecommunication Laboratory. He has also held several positions in her French academic career: University of PAU, University of ORSAY Paris XI, 3IL and Xlim Laboratory. His research interests include distributed testing, secure testing, and software testing.

Salma Azzouzi received the State Engineer degree in Computer Science from the National Institute of Statistics and Applied Economics (INSEA), Morocco, in 2006 and later, she received the Ph.D. degree in Computer Science from the IBN TOFAIL University, Morocco. She is currently a Professor in the faculty of science and a member of the Laboratory of Research in Computer Science and Telecommunication (LARIT) at the Ibn Tofail University. Her research interests include essentially: distributed testing, problems of controllability/observability and timing constraints in distributed testing.