

# A Mobile Agent Based Synchronization For Distributed Testing

Mohammed Benattou

Institut d'Ingénierie Informatique de Limoges

Laboratoire XLIM - UMR CNRS 6172 -

123, avenue Albert Thomas - 87060 LIMOGES CEDEX

e-mail: benattou@3il.fr

**Abstract** - *Last years, products, models, architecture and frameworks suggest several key issues that will contribute to the success of open distributed systems. However, in practice, the development of distributed systems is more complex. The design process must take into account: the mechanisms and functions required to support interaction, communication and coordination between distributed components. In the context of distributed testing, we illustrate how we can use the concept of mobile agent to reduce synchronized exchanging messages in the distributed testing framework.*

**keywords:** Mobile agent, Distributed testing, Component, Synchronization, Coordination

## 1 Introduction

Currently, new and highly interesting paradigms for distributed computing become the key issue in modern system design. Almost every system providing distributed interfaces for interacting with other interconnecting systems can be seen as distributed system. Last years, products, models, architecture and frameworks suggest several key issues that will contribute to the success of open distributed systems. The most well known of these systems are CORBA [6] from the OMG and J2EE/EJB from Sun Microsystem. Any of these standards integrate the concept of component based systems. The overall idea is the system partition of considered distributed system into disparate components, which use a transparent communication medium to access any kind of resources, and provide new high possibilities for Internet-based applications.

However, in practice, the development of distributed systems is more complex. The design process must taken into account the mechanisms and functions required to support interaction as long as communication and coordination between distributed components. Examples of such applications are systems comprising a set of components that broadcast commands among themselves, multicast controllers that coordinate messages between components, the systems using the alarm signals in non deterministic order, network and application systems, event queues, etc. [9]. The typical reaction of such systems is the generation of errors sets: time-outs, locks, channels and network failures. The most often programming models used to implement these environments rely on an event loop with call-backs, or using the CORBA services.

Our preliminary experience, in the use of the CORBA services is the implementation of the distributed testing application [1] of the broadcast and multicast channels. The basic idea of the proposed work [8] is to coordinate the testers by using a communication service parallel to the IUT through a multicast channel. Each tester interacts with the Implementation Under Test (IUT) only through the attached port and communicates with the other testers through the multicast channel. The implementation of the proposed model has shown that the execution of distributed testers arise many time-outs problems influencing fault detection during the testing process.

Object-oriented based, the development of such applications using the "classical" objects is very difficult, like many possible ways of activating or deactivating event sources and to dispatch the call-backs. Others proposed research work based on the temporal Finite State Machine define the

timing constraint, which the distributed testing application must be respected in real-time execution [5]. However, the implementation of these models generates a great number of synchronized exchange messages.

In this paper, we show how to use the concept of mobile agent to reduce synchronized exchanging messages in the distributed testing framework. A mobile agent can be defined as a computer program that acts autonomously on behalf of person or organisation. It has ability to transport itself from one system to another. A number of advantages, besides using a mobile agent include overcoming network latency, reducing network load, performing autonomous and asynchronous execution, and adapting to dynamic environments [4].

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

## 2 Distributed Testing

The principle of testing is to apply input events to an IUT and to compare the observed output events with the expected results. A set of input events and the expected results is generally called a *test case* and it is generated from the IUT specification. Conformance testing may be seen as a mean to execute an IUT by carrying out test cases, in order to observe whether the behaviour of the implementation is conform to its specification. In the context of open distributed system, the IUT may be viewed as a system providing standardized interfaces for interacting with other systems.

### 2.1 Architecture

The basic idea of the proposed work [8] is to coordinate the testers by using a communication service parallel to the IUT through a multicast channel. Each tester interacts with the IUT only through the attached port and communicates with the other testers through the multicast channel. On a high level abstraction (see figure 1), allowing the organizational requirements of the distributed testing environment to be captured we can distinguish three actors:

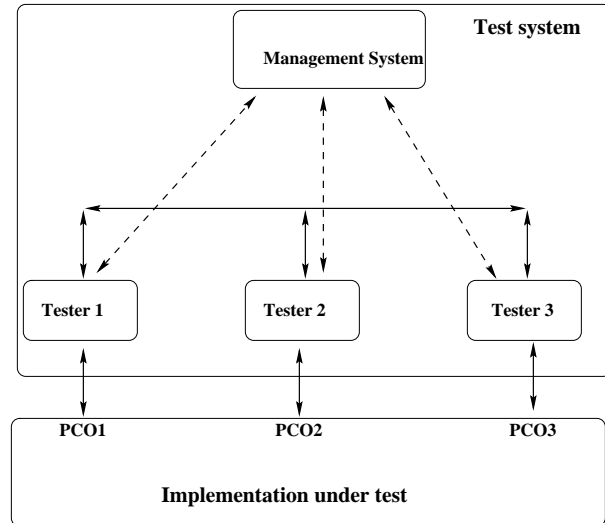


Figure 1: Architecture of distributed testing application

**IUT.** It represents the executable implementation of the distributed software system to be tested. It can be seen as a black-box with points of control and observation (PCO) to which the test system can apply the input events and observe the output results during the testing process.

**Management system.** It generates local test sequences according to the behavior described in [1, 8], dispatches these sequences to related testers, starts and stops the execution of the testers, and provides a global verdict for the test process. The global verdict is deduced from the local test verdicts given by testers. *Management system* has the obligation to stop the execution of the test process if at least one *Tester* object provides a *Fail* verdict.

**Tester.** Each *Tester* object executes its local test sequence. The execution process of a *Tester* object consists of:

- to apply input events to the IUT interface related to it,
- to observe the output results, and
- to provide its local verdict to the *Management system*.

In order to coordinate the test execution process, *Testers* exchange coordination messages dur-

ing their execution. *Tester* objects process together but independently [10].

## 2.2 Modeling concepts

In order to be able to reason about the testing process in a formal setting, the specification and IUT must be modeled by using the same concepts. Therefore, conformance of an IUT to its specification may be defined by means of relations between the IUT model and specification model [3]. I/O FSM (Input/Output Finite State Machines) are widely used in the communication protocol area [3], and may be easily adapted with some extensions for modeling distributed systems. In a communication protocol, a protocol entity communicating with a peer entity, is described by an I/O FSM with one input queue and one output queue. Distributed applications are supposed however to communicate with multiple partners. This leads to the notion of a multi-port FSM [7] which may use several input/output queues called ports.

A *multi-port FSM with  $n$  ports* ( $np$ -FSM)  $\mathcal{A}$  is a 6-tuple  $(Q, \Sigma, \Gamma, \delta, \lambda, q_0)$ , where:  $Q$  is the finite set of *states* of  $\mathcal{A}$ ;  $q_0 \in Q$  is a distinguished state, the *initial state* of  $\mathcal{A}$ ;  $\Sigma$  is a  $n$ -tuple  $(\Sigma_1, \Sigma_2, \dots, \Sigma_n)$  where  $\Sigma_k$  is the *input alphabet of port  $k$* , and  $\Sigma_i \cap \Sigma_j = \emptyset$  for  $i \neq j$ . We write  $\bar{\Sigma}$  for the *input alphabet*  $\Sigma_1 \cup \Sigma_2 \cup \dots \cup \Sigma_n$  of  $\mathcal{A}$ ;  $\Gamma$  is a  $n$ -tuple  $(\Gamma_1, \Gamma_2, \dots, \Gamma_n)$  where  $\Gamma_k$  is the *output alphabet of port  $k$* , and  $\Gamma_i \cap \Gamma_j = \emptyset$  if  $i \neq j$ . We write  $\bar{\Gamma}$  for the *output alphabet*  $(\Gamma_1 \cup \{\}) \times (\Gamma_2 \cup \{\}) \times \dots \times (\Gamma_n \cup \{\})$  of  $\mathcal{A}$ ;  $\delta$  is the *transition function*, it is a partial function  $Q \times \bar{\Sigma} \rightarrow Q$ ;  $\lambda$  is the *output function*, it is a partial function  $Q \times \bar{\Sigma} \rightarrow \bar{\Gamma}$ . Moreover,  $\lambda(q, \alpha)$  is defined if and only if  $\delta(q, \alpha)$  is.

A *transition* of  $np$ -FSM  $\mathcal{A}$  is a 4-tuple  $\mathbf{t} = (q, \alpha, \gamma, q')$  where  $q, q' \in Q$ ,  $\alpha \in \bar{\Sigma}$  and  $\gamma \in \bar{\Gamma}$  are such that  $\delta(q, \alpha) = q'$  and  $\lambda(q, \alpha) = \gamma$ .

Figure 2 gives an example of 3p-FSM with set state  $Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$ ,  $q_0$  being the initial state,  $\Sigma_1 = \{a\}$ ,  $\Sigma_2 = \{b\}$ ,  $\Sigma_3 = \{c\}$ , and  $\Gamma_1 = \{w, x\}$ ,  $\Gamma_2 = \{y\}$ ,  $\Gamma_3 = \{z\}$ . Transitions are represented by arrows.

A *test sequence* of  $np$ -FSM  $\mathcal{A}$  is a sequence in the form:  $!x_1?y_1!x_2?y_2 \dots !x_t?y_t$  where, for  $i = 1, 2, \dots, t$ ,  $x_i \in \bar{\Sigma}$  and  $y_i \subset \bigcup_{k=1}^n \Gamma_k$  is such that, for each port  $k$ ,  $|y_i \cap \Gamma_k| \leq 1$ , i.e.  $y_i$  contains at most one symbol from the output alphabet of each port of  $\mathcal{A}$ .  $!x_i$  means sending message  $x_i$  to the IUT and  $?y_i$  means receiving the messages belonging to

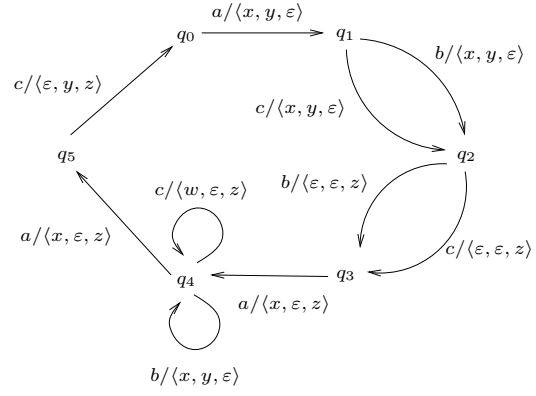


Figure 2: An example of 3p-FSM

$y_i$  from the IUT.

A test sequence for 3p-FSM of the example illustrated in figure 2 is:

$$!a?{x,y}!b?{x,y}!c\{z}!a?{x,z}!b?{x,y}!c?{w,z}!a?{x,z}!c?{y,z} \quad (1)$$

Generally test sequences are generated from the IUT specification and characterized by their fault coverage (output faults and transfer faults [7]). In order to synchronize the test execution, testers exchange coordination messages. These messages encapsulate the information that allows the test system to solve controllability and observability problems. As pointed out in [8] each local test sequence executed by a given tester includes the coordination messages to be sent or to be received from other testers.

In the distributed test method, each tester executes a local test sequence constructed from the complete test sequence of the IUT. A *local test sequence* is in the form  $\alpha_1\alpha_2 \dots \alpha_t$ , where each  $\alpha_i$  is either:

- $!x$ , sending of message  $x \in \Sigma_k$  to the IUT,
- $?y$ , receiving of message  $y \in \Gamma_k$  from the IUT,
- $!c_{h_1, \dots, h_r}$ , sending of coordination message  $c$  to testers  $h_1, \dots, h_r$ ,
- $?c_h$ , receiving of coordination message  $c$  from tester  $h$ .

For each  $\alpha_i$ , if it is a sending, either of a message to the IUT or of a coordination message, the tester sends it. If  $\alpha_i$  is a receiving, either of an output from IUT or of a coordination message, then the tester waits for a message. If no message

is received, or if the received message is not the expected one, the tester gives a fail verdict. If the tester reaches the end of its sequence, then it gives a pass verdict. If all testers give a pass verdict, then the test system ends the test by giving a global pass verdict. If the IUT has  $n$  ports, the algorithm [8] is dedicated to compute the  $n$  related local test sequences from a complete test sequence of IUT.

Applying the proposed algorithm to the test sequence (1), we get the following local test sequences:

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

### 2.3 Synchronization problems

Our preliminary experience, in the use of the event CORBA service for the implementation of the broadcast and multicast channels of distributed testing applications have shown that the parallel testers execution arise many time-outs problems influencing fault detection during the testing process. Lets the execution of the first fragments of the local test sequences  $\omega_{f1}$ ,  $\omega_{f2}$  and  $\omega_{f3}$  selecting from  $\omega_1$ ,  $\omega_2$  and  $\omega_3$  (2).

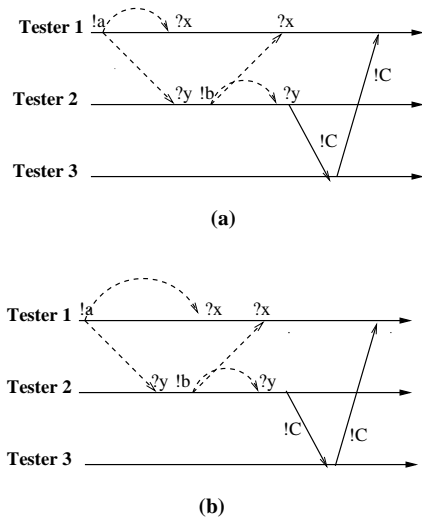


Figure 3: Example 1 of prototype execution

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

The conform execution of local test sequences  $\omega_1$ ,  $\omega_2$  and  $\omega_3$  must give the result shown in figure 3 (a), but the execution of our prototypes gives the failed result shown in figure 3 (b). Indeed, in figure 3 (b) the tester 2 sends the message b to the IUT before the tester 1 receives the output message x from the IUT, and thus the execution of the local sequences is not conform to the  $\omega$  specification (1), where the message b must be sent to the IUT after all output messages caused by the sending of the message a by tester 1 has been received.

The execution example illustrates that the synchronisation between tester objects allows some failed errors, which can be solved by blocking the sending of the message b as long as all outputs, caused by the last message, have been received by all testers. Note that the dashed arrow from !a to ?x corresponds to the sending of the message a to the IUT by the tester 1 which causes the waiting of the messages ?x in tester 1 and ?y in tester 2 sent by the IUT; and that !C is the sending of the coordination message from tester 2 to tester 3.

## 3 Mobile agents

A mobile agent can be defined as a computer program that acts autonomously on behalf of person or organisation. It has ability to transport itself from one system to another. A number of advantages, besides using a mobile agent include overcoming network latency, reducing network load, performing autonomous and asynchronous execution, and adapting to dynamic environments [8]. Our preliminary experience, in using the mobile agent paradigm is modelling and implementing Distributed Intrusion Detection System [2]. It shows that a mobile agent have the analysis capabilities to perform remote query and search actions. In our distributed testing model, agent is used to perform well defined tasks: search the output events occurrence in a given testers.

### 3.1 Generating local test sequences

It should be noticed that, the local test sequences analyse of (2) gives:

- Any message sending by tester to the IUT must be blocked as long as all output events, caused by the last sending message, have been received

by all testers. This behavior is not described in any a local test sequences.

- Difficulties begin with output events identification, and the relating input source causing them.

In order to solve these problems we follow the steps outlined below:

1. Transform the general test sequence generated from the given *multi-port FSM with n ports* by indexing the execution order of input and output events. This order gives the natural execution of each general test sequence transition.

For example, the test sequence given in (1) is transformed to:

$$\begin{aligned} & !a_1?\{x_1,y_1\}!b_2?\{x_2,y_2\}!c_3?\{z_3\}!a_4?\{x_4,z_4\}!b_5?\{x_5,y_5\}!c_6?\{w_6,z_6\} \\ & !a_7?\{x_7,z_7\}!c_8?\{y_8,z_8\} \end{aligned} \quad (4)$$

2. Identify the output events that must be observed before sending any output event to the IUT. In this context, each input event must include the output events caused by the last sending message. We denote by  $!M_i^{\{x_{i-1},y_{i-1}\}}$  the sending of message  $M$  of the transition  $i$  to the IUT after the output events  $\{x_{i-1},y_{i-1}\}$  has been observed in related testers. This new representation is incorporated in the test sequence given in (4):

$$\begin{aligned} & !a_1?\{x_1,y_1\}!b_2^{\{x_1,y_1\}}?\{x_2,y_2\}!c_3^{\{x_2,y_2\}}?\{z_3\}!a_4^{\{z_3\}}?\{x_4,z_4\} \\ & !b_5^{\{x_4,y_4\}}?\{x_5,y_5\}!c_6^{\{x_5,y_5\}}?\{w_6,z_6\}!a_7^{\{w_6,z_6\}}?\{x_7,z_7\} \\ & !c_8^{\{x_7,z_7\}}?\{y_8,z_8\} \end{aligned} \quad (5)$$

3. Apply the algorithm [8] to the test sequence achieved by the last step to generatt local test sequences related to each tester. The applying of the proposed algorithm to the test sequence (5), gets the following local test sequences:

$$\left\{ \begin{aligned} \omega_1 &= !a_1^{\{x_1,y_1\}}?x_1?x_2?C_3?C_3!a_4^{\{z_3\}}?x_4?x_5?w_6!a_7^{\{w_6,z_6\}}?x_7?C_3 \\ \omega_2 &= ?y_1!b_2^{\{x_1,y_1\}}?y_2!C_3?C_3!b_5^{\{x_4,y_4\}}?y_5!C_3?C_3?y_8 \\ \omega_3 &= ?C_2!C_1!c_3^{\{x_2,y_2\}}?z_3!C_1?z_4!C_2?C_2!c_6^{\{x_5,y_5\}}?z_6?z_7 \\ & !C_{\{1,2\}}!c_8^{\{x_7,z_7\}}?z_8 \end{aligned} \right. \quad (6)$$

Notice that, the form of coordination messages has not been changed.

### 3.2 Mobile agent system design

The whole idea behind the use of the mobile agent is to reduce the number of exchanging messages between testers. The message sending

$!M_i^{\{x_{i-1},y_{i-1}\}}$  by some tester to IUT can not be executed only if the messages  $\{x_{i-1},y_{i-1}\}$  have been received by related testers. And then related testers receiving these messages must inform the considered tester. However, none of these messages appeared in local test sequences. In this context, we redefine the behavior of each tester described above only for the sending message to IUT as follow:

For each message sending with the form  $!M_i^{Waiting\_messages}$  each tester execute the following steps:

- Generate a mobile agent
- Detect for each output message belonging to the set of *Waiting\_messages* its related tester
- Send the generated mobile agent to set of detected testers
- Block the message sending  $M_i$  to IUT until receiving the mobile agent that confirms the observation of all considered output events.
- Send the message  $M_i$  to the IUT

## 4 Conclusion and future works

Not only distributed computing become the key issue in modern system design, but it provides new high possibilities for Internet-based applications. However, in practice the development of distributed component systems is more complex. Especially where the implementation must take into account some synchronization rules, and the coordination of distributed components, as it is often the case in complex information systems. In this paper, we have described how we can use the concept of mobile agent to reduce synchronized exchanging messages in the distributed testing framework. Our prototype realisation of this model has been experimented in Java environment We have chosen Voyager [12] as mobile agent environment to implement our prototype testing. Voyager, from recursion software, can be seen as strengthened object request broker. Compared with other mobile agents development platforms, Voyager integrates Java more intensely, and be used easily to develop mobile agent platforms, and also to build up traditional distributed systems. Our work is now oriented toward the development of an environment, which take into consideration the functional requirements of time constraints in this context.

## References

- [1] M. Benattou and Jean-Michel Bruel, *Active Objects for Coordination in Distributed Testing*, Proceedings of the 8th Int. Conf. on Object-Oriented Information Systems OOIS'02, Lecture Notes in Computer Science, Vol 2425, pp 348-357, 2002.
- [2] M. Benattou and K. Tamine, *Mobile Agents Community For Distributed Intrusion Detection System*, Proceeding of International conference on Computing, Communication and Control Technologies, Austin, USA, July 2005 .
- [3] ISO/IEC, *Information retrieval, transfer and management for OSI, Framework: formal methods in conformance testing*, CD 13345-1, 1996.
- [4] W. A. Jansen, *Determining Privileges of Mobile Agents*, 17th Annual Computer Security Applications Conference, pages 149-160, 2001.
- [5] A. Khoumsi, *Testing distributed real-time systems in the presence of inaccurate clock synchronizations*, Information and Software Technology, Volume: 45, p. 853-864, 2003
- [6] OMG:*The common object request broker: architecture and specification*, Version 2.2, 1998.
- [7] A. Petrenko, G. v. Bochmann and M. Yayo, *On fault coverage of tests for finite state specification*, Computer Network and ISDN Systems 29, 1996, pp. 81-106.
- [8] O. Rafiq, L. Cacciari, M. Benattou: *Coordination Issues in Distributed Testing*; Proceeding of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99), Las Vegas (Nevada), USA, 1999, CSREA Press.
- [9] A. J. Restrepo-Zea, C. Petitpierre: *A Simple, Modeling prone CORBA Architecture*
- [10] A. Ulrich, H. Konig:*Architectures For Testing Distributed Systems*; Proceeding of the IFIP International Workshop on Testing of Communicating Systems (IWTCS'99), Hungary, 1999, pp. 93-107.
- [11] T. Vassiliou-Gioles, I. Schieferdeck, M. Born, M. Winkler and M. Li:*Configuration and Execution Support For Distributed Tests*; Proceeding of the IFIP International Workshop on Testing of Communicating Systems (IWTCS'99), Budapest, Hungary, 1999, pp. 61-76.
- [12] T. Wheeler, *Reducing Development Effort Using the Voyager ORB* , Recursion Software, Inc, 2002.