Using status messages in the distributed test architecture

R. M. Hierons [a]

[a] *School of Information Systems, and Computing Mathematics, Brunel University, Uxbridge, Middlesex, UB8 3PH*

Abstract

If the system under test has multiple interfaces/ports and these are physically distributed then in testing we place a tester at each port. If these testers cannot directly communicate with one another and there is no global clock then we are testing in the distributed test architecture. If the distributed test architecture is used then there may be input sequences that cannot be applied in testing without introducing controllability problems. Additionally, observability problems can allow fault masking. In this paper we consider the situation in which the testers can apply a status message: an input that causes the system under test to identify its current state. We show how such a status message can be used in order to overcome controllability and observability problems.

Key words: distributed test architecture, controllability problem, observability problem, status message.

1 Introduction

Many systems are state-based and such systems are typically specified or modelled using finite state machines (FSMs) or state-based languages such as statecharts and SDL that are based on extended finite state machines (EFSMs). Since FSM based test techniques are often used when testing from an EFSM, possibly after some transformations or abstractions have been applied [11, 12, 32], there has been much interest in testing from FSMs. In particular, the problem of testing from an FSM has received much attention in the areas of testing protocols, reactive systems, and object-oriented systems (see, for example, [3, 13, 20]).

In the distributed test architecture there is a remote tester at each port of the system under test (SUT), these testers cannot directly communicate with one another and there is no global clock. The use of the distributed test architecture can lead to additional controllability and observability problems (see, for example, [2, 4, 5, 7, 9, 10, 16, 17, 21, 22, 26, 28, 29, 31, 34]). A controllability problem occurs when a tester does not know when to apply an input since it was not involved in the previous transition. Let us suppose, for example, that there are two ports U and L, the first input x_1 is to be applied at port U and is expected to lead to output at U only and this is to be followed by input x_2 at L. Then the tester at port L does not know when to apply input x_2 since it does not observe the input or output from the previous transition; there is a controllability problem. An observability problem occurs when a tester receives an output but cannot determine which input led to the SUT producing this output. Let us suppose, for example, that the first input x_1 is to be applied at port U and is expected to lead to output y_U at U only and this is to be followed by input x_2 at U, which is expected to lead to output y_L at L only. Then both testers observe the same behaviour if instead the input of x_1 leads to output y_U at U and y_L at L and the input of x_2 leads to no output: two faults mask one another in this sequence but not necessarily in other sequences. Controllability problems can lead to there being input sequences that cannot be applied in the distributed test architecture while observability problems can lead to fault masking.

It is often possible to overcome controllability and observability problems by introducing an external network through which the remote testers can communicate [4, 26]. However, the introduction of such a network can increase the cost of testing. Further, the exchange of messages through the external network introduces delays and these may be problematic if there are real-time constraints [19]. While there are test sequence generation techniques that do not require such an external network, these place restriction on the SUT and so are not generally applicable (see, for example, [5, 7, 16, 17, 27, 29, 31]).

Sometimes an SUT has an input that leads to an output that identifies the current state of the SUT. Such an input is called a *status message* and may have been introduced in order to simplify testing (see, for example, [18]). This paper considers the problem of testing in the distributed test architecture where the SUT has a status message that can be input at any port and it is known that this status message has been correctly implemented. This paper shows how, when testing from an FSM [1], the controllability and observability problems that result from the distributed test architecture can be overcome using status messages. We investigate two types of status message. The first type leads to output being sent to every port and can be used in order to allow the testers at the separate ports to communicate via the SUT. We show that controllability and observability problems can always be overcome when

[1] There are timed extensions to FSMs that allow real-time constraints to be expressed but testing from such models is left as a problem for future work.

there are such status messages that are know to have been implemented correctly. The notion of a status message relates to the concept of a monitor in debugging and there has been much work on the implementation of monitors with the aim of obtaining the global state of a system and achieving this in a non-invasive manner (see, for example, [1, 8, 15, 24, 35]). Here, however, the observation of the state of the SUT is made locally and this corresponds to the second type of status message that we consider, in which the state of the system is output at the port that sent the status message. We show that such status messages cannot be used to overcome controllability and observability problems in arbitrary test sequences but can be used if we carefully choose our test sequences. We give algorithms for generating such a test sequence that achieves full fault coverage for any FSM that does not contain what we call pathological transitions. Note that since we use information about the state of the SUT, the techniques devised in this paper are not black-box and instead are gray-box.

While status messages could be implemented using monitoring systems, the inclusion of status messages can also be seen as a design for test property for distributed systems. Note that previous work such as [4, 19, 26] resolves the controllability and observability problems through the exchange of messages on an external network while we introduce a method that does not require such an external network but does require there to be correctly implemented status messages.

This paper is structured as follows. Finite state machines and the distributed test architecture are described in Section 2. Section 3 shows how controllability and observability problems can be overcome using status messages that lead to output being sent to every port. Section 4 extends this to the use of status messages that lead to output being sent to one port only. Finally, in Section 5, conclusions are drawn and future work is outlined.

2 Preliminaries

2.1 Finite state machines

This paper considers the problem of testing an SUT that has a set of ports denoted $\mathcal{P} = \{p_1, \ldots, p_m\}$. A (deterministic and completely specified) multi-port finite state machine (FSM) is denoted $M = (S, X, Y, \delta, \lambda, s_0)$ in which:

(1) $S = \{s_1, \ldots, s_n\}$ is a finite set of states;
(2) $X = X_1 \cup \ldots \cup X_m$ is the finite input alphabet where X_i is the input alphabet for port p_i and for all $1 \leq i < j \leq m$ we have that $X_i \cap X_j = \emptyset$;

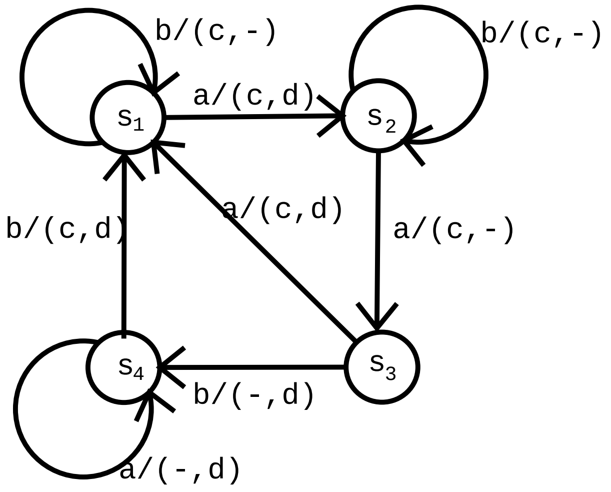

Fig. 1. Multi-port FSM M_0 in which a is an input at port U, b is an input at port L, c is an output at U and d is an output at L

(3) $Y = (Y_1 \cup \{-\}) \times \ldots \times (Y_m \cup \{-\})$ is the finite output alphabet where $-$ denotes null output, Y_i is the output alphabet for port p_i and for all $1 \leq i < j \leq m$ we have that $Y_i \cap Y_j = \emptyset$. If $y_i \in Y_i \cup \{-\}$, $1 \leq i \leq m$ then $(y_1, \ldots, y_m)$ denotes the element of Y with y_i in position i for all $1 \leq i \leq m$;
(4) δ is the state transfer function of type $S \times X \rightarrow S$;
(5) λ is the output function of type $S \times X \rightarrow Y$;
(6) $s_0 \in S$ is the initial state.

In this paper we only consider deterministic completely specified multi-port FSMs and these will simply be called FSMs. Such an FSM M behaves in the following way. If it receives input x when in state s then it moves to state $s' = \delta(s, x)$ and outputs $y = \lambda(s, x)$, defining a *transition* $t = (s, s', x/y)$. States s and s' are said to be the *start* and *end states* of t respectively and x/y is its *label*. The transition t is a *self-loop transition* if its start and end states are the same: $s = s'$.

Consider, for example, the FSM M_0 given in Figure 1. Here there are two ports U and L, two inputs (a at port U and b at L), and two outputs (c at port U and d at port L). The labels on the arcs represent the transitions and so, for example, the arc from s_1 to s_2 with label $a/(c, d)$ denotes the input of a in state s_1 being able to trigger a transition that leads to the output of c at U and d at L and the state becoming s_2. This transition can be represented by the tuple $(s_1, s_2, a/(c, d))$.

Given input $x \in X$, $port(x)$ denotes the port p_i at which x is applied: $port(x) = p_i \Leftrightarrow x \in X_i$. Further, $portno(x)$ denote the port number: $portno(x) = i \Leftrightarrow port(x) = p_i$. Given an output y, $ports(y)$ denotes the set of ports at which y is non-empty: if $y = (y_1, \ldots, y_m)$ then $ports(y) = \{p_i | y_i \neq -\}$. Further, given a transition $t = (s, s', x/y)$, $ports(t) = ports(y) \cup \{port(x)\}$ is the set of ports *involved* in t.

Two transitions t and t' are *consecutive transitions* if the start state of t' is the end state of t. A sequence $\bar{\rho} = t_1 \dots t_k$ of consecutive transitions of M is said to be a *path* of M and the *start state* of $\bar{\rho}$ is the start state of t_1. If the label of t_i is x_i/y_i for all $1 \leq i \leq k$ then the *label* of $\bar{\rho}$ is the input/output sequence $label(\bar{\rho}) = x_1/y_1 \dots x_k/y_k$. Given an input/output sequence $\bar{z} = x_1/y_1 \dots x_k/y_k$, the *input portion* of $\bar{z}$ is the input sequence $x_1 \dots x_k$. Note that throughout this paper, if a variable represents a sequence then it will have a bar over its name.

Two states s and s' of M are *equivalent* if for all $\bar{x} \in X^*$ we have that $\lambda(s, \bar{x}) = \lambda(s', \bar{x})$. Similarly, two FSMs are *equivalent* if their initial states are equivalent and an FSM M is *minimal* if there is no FSM M' that has fewer states than M and is equivalent to M. Since every FSM is equivalent to a minimal FSM we will assume that any FSM considered is minimal. FSM M is *strongly connected* if for every ordered pair of states s, s' of M there is a path from s to s' in M. We only consider minimal strongly connected FSMs in this paper.

In testing we apply an input sequence, called a *test sequence*, to the SUT and compare the resultant input/output sequence with that defined by M. In order to reason about test effectiveness it is normal to state a *fault model*: a set $\Phi(M)$ of models with the property that we believe that the SUT behaves like an (unknown) element of $\Phi(M)$. A test sequence $\bar{x}$ is a *checking sequence* if for every $N \in \Phi(M)$, if N is not equivalent to M then some tester observes a failure if $\bar{x}$ is applied to N when in its initial state. In this paper we use the fault model $\Phi(M)$ that is the set of FSMs with the same input and output alphabets[2] as M. We thus assume that the SUT behaves like an unknown FSM $N = (U, X, Y, \delta_I, \lambda_I, u_0)$.

If an input sequence $\bar{x}$ is applied in a state s of FSM M then there is a resultant input/output sequence that will be denoted $\gamma_M(s, \bar{x})$. For example, the application of aa in state s_1 of M_0 leads to the input/output sequence $a/(c, d)a/(c, -)$. The function γ_M can be defined by the following rules: $\gamma_M(s, \epsilon) = \epsilon$ and $\gamma_M(s, x\bar{x}) = (x/\lambda(s, x))\gamma_M(\delta(s, x), \bar{x})$, where ϵ is the empty sequence. A function γ_N of type $U \times X^* \rightarrow (X/Y)^*$ can be defined in a similar way for the FSM N that models the SUT. Given an input/output sequence $\bar{z} = x_1/y_1 \dots x_k/y_k$ and port $p_i \in \mathcal{P}$, $\pi_i(\bar{z})$ is the projection of $\bar{z}$ produced by removing all elements of $\bar{z}$ that are not observed at port p_i, an example being $\pi_U(a/(c, d)a/(c, -))$ being $acac$ and $\pi_L(a/(c, d)a/(c, -))$ being d. Thus, $\pi_i(\epsilon) = \epsilon$ and for all $x \in X$, $y = (y_1, \dots, y_m) \in Y$ and $\bar{z} \in (X/Y)^*$ we have

[2] If there are potential outputs of the SUT that are not contained in the output alphabet Y of M then these can be added to this output alphabet.

that:

- If $x \in X_i$ and $y_i \neq -$ then $\pi_i(x/y\bar{z}) = xy_i\pi_i(\bar{z})$;
- If $x \in X_i$ and $y_i = -$ then $\pi_i(x/y\bar{z}) = x\pi_i(\bar{z})$;
- If $x \notin X_i$ and $y_i \neq -$ then $\pi_i(x/y\bar{z}) = y_i\pi_i(\bar{z})$; and
- If $x \notin X_i$ and $y_i = -$ then $\pi_i(x/y\bar{z}) = \pi_i(\bar{z})$.

A status message will uniquely identify the current state of the SUT and, where appropriate, give the corresponding state of M. Throughout this paper we let r denote a function that, given a state u of N, returns a label that identifies u. If u corresponds to a state s of M then $r(u) = s$ and otherwise $r(u) \notin S$ is a unique label that represents u.

The presence of a status message assists in the debugging or monitoring of a system and essentially involves an external entity (a monitor) having access to the internal state of the SUT and this is typically achieved through instrumenting the code. However, this is significantly more difficult for distributed systems since each subsystem will have its own state and it is essential to combine the states of these subsystems in a manner that ensures that these local states are sampled at the same time (see, for example, [8, 24]). This problem can sometimes be solved through the use of appropriate hardware in which specialised processers are responsible for reporting on state information [8]. It can also be achieved through middleware or an operating system or through there being a consistent set of clocks [24], the clocks potentially being synchronized through internal communications [15]. While the problem of identifying the internal state has been studied in the context of debugging and monitoring, we show how such information can be used to overcome controllability and observability problems in testing.

2.2 Directed graphs

A directed graph G is defined by a set V of vertices and a set E of directed edges between the vertices and so $E \subseteq V \times V \times L$ for some set L of labels. Given $G = (V, E)$, $(v_i, v_j, l) \in E$ represents an edge from v_i to v_j with label l. An FSM $M = (S, X, Y, \delta, \lambda, s_0)$ with n states can be represented by a directed graph $G = (V, E)$ in which each state s_i is represented by vertex v_i and M has a transition $(s_i, s_j, x/y)$ if and only if E contains the edge $(v_i, v_j, x/y)$.

A sequence $\bar{\rho} = e_1 \dots e_k$ of consecutive edges, $e_i = (v_i, v_{i+1}, l_i)$, is a *path* of G and if $v_1 = v_{k+1}$ then $\bar{\rho}$ is a *tour*. A tour of G is a *Postman Tour* if it contains each edge of G at least once. If each edge is given a cost then the *Chinese Postman Problem (CPP)* is to find a Postman Tour that has minimum cost, where the cost of a tour is the sum of the costs of the edges in the tour (see, for example, [14]). In this paper digraphs are used to represent FSMs and so the cost of an edge will be either 1, if it represents a single transitions, or k if it represents a sequence of k transitions.

2.3 Controllability and observability problems

Let us suppose that we intend to apply the input sequence $x_1 \dots x_k$ to the SUT and M contains a path $\bar{\rho}$ whose starting state is s_0 and whose label is $x_1/y_1 \dots x_k/y_k$. Consider the port p_i at which the input x_j is to be applied, $1 < j \leq k$. Then the tester at p_i can only know when to send x_j if either it applied the previous input x_{j-1} or it received output in the previous transition: $p_i \in ports(y_{j-1})$. If this is not the case then there is a controllability problem.

Definition 1 *If M contains a path $\bar{\rho}$ whose start state is s_0 and whose label is $x_1/y_1 \dots x_k/y_k$ then the input sequence $x_1 \dots x_k$ is* synchronisable *if for all $1 < j \leq k$ we have that $port(x_j) \in \{port(x_{j-1})\} \cup ports(y_{j-1})$. We also say that $\bar{\rho}$ is* synchronisable.

If the distributed test architecture is used and there are no status messages then if $x_1 \dots x_k$ is not synchronisable then it cannot be applied without causing a controllability problem. For example, in M_0 if we apply a in state s_2 then there is output only at U and so this cannot be followed by input b without causing a controllability problem since the tester at L cannot know when to apply b.

Now let us suppose that we wish to execute two consecutive transitions $t = (s, s', x/y)$ and $t' = (s', s'', x'/y')$. There is an *output fault* if one or more of these transitions produces the wrong output in testing. However, such an output fault may go undetected if it is masked by an output fault in the other transition. Let us suppose, for example, that t has output of y_i at port p_i and t' has output $-$ at p_i. Then if instead t produces output $-$ at p_i and t' produces y_i at p_i then the net output at p_i in tt' is correct. Such a pair of output faults will only be detected if either the test sequence also contains at least one of t and t' in a context where such fault masking cannot happen or the input of t' is at p_i and so the tester at p_i knows when to stop waiting for the output y_i. Such problems are captured by the following definitions.

Definition 2 *Let $t = (s, s', x/y)$ and $t' = (s', s'', x'/y')$ be two consecutive transitions of M where y has output $y_i \neq -$ at $p_i \in \mathcal{P}$ and y' has output $-$ at p_i. There is a* forward output shifting fault *if in the SUT t has output $-$ at p_i and t' has output y_i at p_i. This is a* potentially undetectable forward output shifting fault *if $port(x') \neq p_i$.*

In M_0 there is a potentially undetectable forward output shifting fault in the sequence $(s_1, s_2, a/(c, d))(s_2, s_3, a/(c, -))$ of transitions since neither tester would observe a difference if the first transition led to output $(c, -)$ and the second transition led to output (c, d). This is because in each case the tester at U observes *acac* and the tester at L observes d. Of course, if these transitions were contained in different sequences such as $(s_1, s_2, a/(c, d))(s_2, s_2, b/(c, -)))$ in use then the user might observe a failure.

Definition 3 *Let* $t = (s, s', x/y)$ *and* $t' = (s', s'', x'/y')$ *be two consecutive transitions of* M *where* y *has output* $-$ *at* $p_i \in \mathcal{P}$ *and* y' *has output* $y_i \neq -$ *at* p_i. *There is a* backward output shifting fault *if in the SUT* t *has output* y_i *at* p_i *and* t' *has output* $-$ *at* p_i. *This is a* potentially undetectable backward output shifting fault *if* $port(x') \neq p_i$.

In M_0 there is a potentially undetectable backward output shifting fault in the sequence $(s_2, s_3, a/(c, -))(s_3, s_1, a/(c, d))$ of transitions since neither tester would observe a difference if the first transition led to output (c, d) and the second transition led to output $(c, -)$.

These two classes of output shifting faults capture the observability problem when testing in the distributed test architecture. While output can be shifted between two transitions that are not adjacent in a path, it has been proved that such output shifting faults cannot go undetected if there can be no potentially undetectable output shifting faults for adjacent transitions [6].

It is important to note that while an observability problem could lead to an output fault in a transition t being masked in a given test sequence, there may be other sequences in which such a fault is not masked. Thus, an output fault masked in testing might lead to a failure in use.

In this paper we assume that the time between the SUT receiving an input and the testers receiving the resultant outputs is negligible. This avoids issues such as one message overtaking another and corresponds to the notion of a slow environment (see, for example, [23]). See [19] for a discussion of the issues introduced when the time taken to send a message is not negligible. We also assume that the status messages have been implemented correctly.

3 Status messages that send output to all ports

In this section we assume that a tester at any port p_i can send a status message $\mathcal{S}_i$ and that in response the SUT will send the output $status(u) = (r_1(u), \ldots, r_m(u))$ in which u is the current state of the SUT and for each port p_j, $r_j(u)$ denotes $r(u)$ sent to p_j. In this section we first show how such a status message can be used in order to overcome controllability and observability problems. We then explain how, given a test sequence $\bar{x}$, it is possible to add a minimum number of status messages that overcome these problems.

3.1 Overcoming controllability and observability problems

Let $\bar{x} = x_1 \dots x_k$ denote an input sequence that we wish to apply to the SUT. If we follow each input x_i by a status message then all testers know that x_i has been applied and this allows us to overcome controllability and observability problems.

Definition 4 *If $\bar{x} = x_1 \dots x_k$ denotes an input sequence then $f_1(\bar{x}) = \mathcal{S}_{i_1} x_1$ $\mathcal{S}_{i_1} x_2 \mathcal{S}_{i_2} \dots x_{k-1} \mathcal{S}_{i_{k-1}} x_k$ where $i_j = portno(x_j)$.*

Proposition 1 *If $\bar{x} = x_1 \dots x_k$ is an input sequence then there are no controllability or observability problems when applying $f_1(\bar{x})$.*

Proof

There are no controllability problems since the input of each value from $\bar{x}$ is preceded by output to all ports in response to a status message. Each port p_i observes a sequence in the form $\mathcal{S}_i/o_1\bar{z}_1\mathcal{S}_i/o_2\bar{z}_2 \dots \mathcal{S}_i/o_{k-1}\bar{z}_k\mathcal{S}_i/o_k$ in which o_j denotes the response to a status message and $\bar{z}_j$ denotes any input and output observed at p_i as a result of the input of x_j after $x_1 \dots x_{j-1}$. Thus, each tester can determine which input led to a particular observed output and so there are no observability problems. □

We can thus produce a checking sequence if we add a final status message to check the end state of the last transition.

Proposition 2 *Let us suppose that we have status messages that have been correctly implemented. If $\bar{x} = x_1 \dots x_k$ is the input portion of the label of a path of M with starting state s_0 that contains every transition of M then if we follow $f_1(\bar{x})$ with a status message then we obtain a checking sequence.*

Proof

We assume that the application of $f_1(\bar{x})$ followed by a status message does not lead to a failure being observed and so the expected sequence of responses to status messages and inputs is observed. It is now sufficient to prove that M and the FSM $N = (U, X, Y, \delta_I, \lambda_I, u_0)$ that models the SUT must be equivalent. Recall that for state u of N, the response $r(u)$ to a status message uniquely identifies the state of N and is s if u corresponds to state s of M.

Since $f_1(\bar{x})$ starts with a status message, we know that $r(u_0) = s_0$. Since the application of $\bar{x}$ to the completely specified M leads to each input from X being applied in each state in S, for every state s and input x, if $r(u) = s$, $\delta(s, x) = s'$, $r(u') = s'$ then $\delta_I(u, x) = u'$. From this we can deduce that N has n states $u_1, \ldots, u_n$ such that for all $1 \leq i \leq n$ we have that $r(u_i) = s_i$ and that r is a bijection from U to S. It is now sufficient to prove that for every state $u_i \in U$ and input $x \in X$, $\lambda_I(u_i, x) = \lambda(s_i, x)$. This follows from observing that for each u_i and x, x is applied in state u_i in $f_1(\bar{x})$, no failures are observed, and by Proposition 1 there are no observability problems. □

The problem of finding a checking sequence with fewest inputs is an instance of the Chinese postman problem (CPP). The CPP can be solved in low order polynomial time (see, for example, [30]).

3.2 Optimization

We have shown how status messages can be used to overcome controllability and observability problems. We now show how a minimum number of status messages can be added to $\bar{x}$ in order to achieve this.

Given consecutive transitions t_1 and t_2 of M we can identify the conditions under which there can be controllability or observability problems in $t_1 t_2$.

Definition 5 *Let $t_1 = (s, s', x_1/y_1)$ and $t_2 = (s', s'', x_2/y_2)$ denote consecutive transitions of M. Then $pr(t_1, t_2)$ is true if and only if one or more of the following occurs:*

- *There is a controllability problem: $port(x_2) \notin ports(t_1)$.*
- *There is a potentially undetectable forward output-shifting fault: $ports(y_1) \not\subseteq ports(y_2) \cup \{port(x_2)\}$.*
- *There is a potentially undetectable backward output-shifting fault: $ports(y_2) \not\subseteq ports(y_1) \cup \{port(x_2)\}$.*

Based on this we can identify the locations where status messages are required.

Definition 6 *Let $\bar{x} = x_1 \ldots x_k$ denote an input sequence and let $\bar{\rho} = t_1 \ldots t_k$ denote the sequence of transitions of M whose label has input portion $\bar{x}$. Then $f_2(\bar{\rho})$ is defined recursively as follows:*

- *If $|\bar{x}| \leq 1$ then $f_2(\bar{\rho}) = \bar{x}$*
- *If $|\bar{x}| > 1$ and $pr(t_1, t_2)$ then $f_2(\bar{\rho}) = x_1 \mathcal{S}_i f_2(t_2 \ldots t_k)$ where $i = portno(x_1)$*
- *If $|\bar{x}| > 1$ and $\neg pr(t_1, t_2)$ then $f_2(\bar{\rho}) = x_1 f_2(t_2 \ldots t_k)$*

The use of f_2 allows us to overcome controllability and observability problems and does so while adding a minimum number of status messages.

Proposition 3 *Let $\bar{\rho}$ denote a path of M that starts at s_0 and whose label has input portion $\bar{x}$. There are no controllability or observability problems when applying $f_2(\bar{\rho})$.*

Proposition 4 *Let $\bar{\rho}$ denote a path of M that starts at s_0 and whose label has input portion $\bar{x}$. If $\bar{x}'$ is an input sequence whose application to M causes no observability or controllability problems and $\bar{x}'$ can be formed from $\bar{x}$ by adding status messages then $|\bar{x}'| \geq |f_2(\bar{\rho})|$.*

Proof

Let $\bar{\rho} = t_1 \dots t_k$. Proof by contradiction: assume that $|\bar{x}'| < |f_2(\bar{\rho})|$. Thus, there exists $1 < j \leq k$ such that in $f_2(\bar{\rho})$ there is a status message between x_{j-1} and x_j and in $\bar{x}'$ there is no status message between x_{j-1} and x_j. Then, since there is a status message in $f_2(\bar{\rho})$ between x_{j-1} and x_j we have that $pr(t_{j-1}, t_j)$ is true. But by the definition of $pr(t_{j-1}, t_j)$, there must be a controllability problem or an observability problem between t_{j-1} and t_j, providing a contradiction as required. □

We can thus augment a given test sequence with a minimum number of status messages in order to overcome any controllability or observability problems. Note, however, that in order to produce a checking sequence we will also want to check the initial state of the SUT and in order to do so we may have to add an initial status message. In addition, if we want to produce a checking sequence then we have to be careful that we have checked the final state of every transition[3].

Consider the FSM M_0 and for state s_i let t_{ia} denote the transition from s_i with input a and let t_{ib} denote the transition from s_i with input b. Then the following sequence contains every transition of M_0.

$$t_{1b}t_{1a}t_{2b}t_{2a}t_{3a}t_{1a}t_{2a}t_{3b}t_{4a}t_{4b}$$

Let us suppose that we wish to apply function f_2 to this sequence and also precede it by a status message in order to obtain a test sequence. We add status messages in order to overcome the following potential problems.

(1) A potentially undetectable backward output shifting fault for $t_{1b}t_{1a}$;
(2) A potentially undetectable backward output shifting fault for $t_{2a}t_{3a}$;

[3] It is sufficient for every transition to be preceded and followed by status messages at least once in the sequence but this is not a necessary condition.

(3) A potentially undetectable forward output shifting fault for $t_{1a}t_{2a}$;
(4) A controllability problems and a potentially undetectable forward output shifting fault for $t_{2a}t_{3b}$;
(5) A controllability problem for $t_{3b}t_{4a}$;
(6) A potentially undetectable backward output shifting fault for $t_{4a}t_{4b}$.

This leads to the following test sequence.

$$\mathcal{S}t_{1b}\mathcal{S}t_{1a}t_{2b}t_{2a}\mathcal{S}t_{3a}t_{1a}\mathcal{S}t_{2a}\mathcal{S}t_{3b}\mathcal{S}t_{4a}\mathcal{S}t_{4b}$$

This leaves two main open problems. First, there is the question of how we can produce an optimal test sequence for a given test criterion. Here, there are alternative notions of optimal such as minimizing the number of status messages used or the total length, including status messages, being minimal. A second question relates to the following observation: If we wish to apply an input sequence $\bar{x}$ that is the input portion of the label of path $\bar{\rho}$ then $\bar{\rho}$ may contain repeated transitions. Where this is the case, we may be able to eliminate some status messages from $f_2(\bar{x})$ if we require only that the test sequence has no controllability problems and that every transition is tested in at least one context in which there can be no observability problems. This raises the question of how we can add a minimal number of status messages in order to ensure that every transition is tested.

4 Status messages that send output to a single port

While status messages are often used in the testing of state-based system, a status message sent to a port p_i will normally only lead to output at p_i; such messages will be called *local status messages*. In addition, while it should be possible to adapt monitors used in debugging to send status messages to all ports, this is not how they usually operate. Thus in this section we assume that if the status message $\mathcal{S}_i$ is sent then the SUT will reply with the output $status_i(u) = (-, \ldots, -, r_i(u), -, \ldots, -)$ in which u is the current state of the SUT and $r_i(u)$ denotes $r(u)$ sent to p_i. In this section we explain how such status messages can be used in order to overcome controllability and observability problems.

4.1 Overcoming controllability and observability problems

In contrast to the status messages discussed in Section 3, a tester cannot indirectly communicate with another tester through the use of a local status message. However, a tester might use local status messages in order to observe the state of the SUT and through this it can observe a change of state that resulted from a transition in which it was not involved. In this section we show how such an approach can be used in order to overcome controllability and observability problems. We assume that a tester at port p_i sends local status messages sufficiently frequently so that it is guaranteed that the SUT receives a status message from p_i between any two consecutive transitions[4]. Thus, if a transition occurs and this leads to a change of state then the change of state is observed by each tester, through the response to local status messages, and this is observed *after* the output is received from the SUT. One way of implementing this approach is to generate status reports periodically (see, for example, [24] for a discussion of such schemes) and to choose the period to be sufficiently small.

While it is well known that the presence of status messages simplifies the testing of state based systems, the type considered in this section do not necessarily overcome controllability and observability problems. Consider, for example, the FSM M_0 and the input of ba in s_1. The first transition should lead to no change of state and so even if we use status messages the most we can observe is cac at U, bd at L and a change in state to s_2 after this. We make the same observations if instead the SUT produces $b/(c,d)$ and then $a/(c,-)$ and has the same state changes and so the use of status messages does not overcome an observability problem in this test sequence. In addition, consider the FSM M_1 in Figure 2. Here the path from the initial state s_1 with label $a/(c,-)b/(c,d)$ contains a controllability problem since the tester at L does not know when to apply b. However, since the first transition should lead to no state change, the ability of the tester at L to observe the state of the SUT cannot be used in order to overcome this controllability problem. Thus, since status sequences cannot be used to overcome controllability and observability problems in arbitrary test sequences we will have to carefully choose our test sequences in addition to using status messages.

Let us suppose that the test sequence includes the input x that should execute the transition $t = (s, s', x/y)$ with $s' \neq s$ and local status messages are sent before and after the transition. If the correct state transfer happens then every tester is aware that this transition has occurred and otherwise a failure is noted. If no failure is observed then the tester that is to apply the next input knows when to do this even if it was not involved in t. In addition, if the previous transition also involved a state change then for each port p_i the tester at p_i can determine the output at p_i in response to x since this output occurs between two state changes. Thus, there are no controllability or observability problems. However, this observation relies on t and the previous transition

[4] Recall that we assume that the time between the SUT receiving an input and the testers receiving the resultant outputs is negligible.

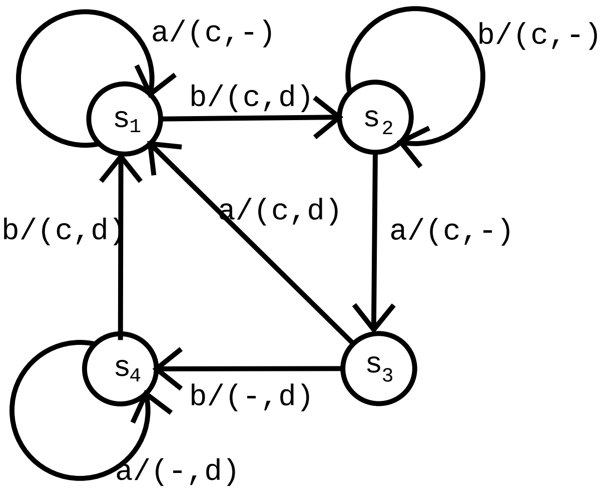

Fig. 2. Multi-port FSM M_1 in which a is an input at port U, b is an input at port L, c is an output at U and d is an output at L

each leading to a change in state in M and thus does not allow us to include self-loop transitions in such sequences. We now consider such transitions.

Definition 7 *A transition $t = (s, s, x/y)$ is said to be* pathological *if there is no path $\bar{\rho}$ with starting state s and an ending state $s' \neq s$ such that $t\bar{\rho}$ is synchronisable.*

If M contains a pathological transition t and this is taken then we are in the situation in which no tester (or user/subsystem) is aware of the opportunity to change state even if such an opportunity exits. This is like a sink state and so we assume that there are no pathological transitions. It is straightforward to show that M_0 contains no pathological transitions.

Definition 8 *A path $\bar{\rho} = t_1 \ldots t_k$ of M is* weakly synchronisable *if for all $1 \leq i < k$ we have that either $t_i t_{i+1}$ is synchronisable or the start state and end state of t_i are different.*

If $\bar{\rho}$ is weakly synchronisable then local status messages allow us to overcome controllability problems: The tester that is to apply the input from t_{i+1} precedes and follows the input of x in transition t_i by a local status message and if $t_i t_{i+1}$ is not synchronisable then the state change allows it to determine when t_i has occurred.

Proposition 5 *If M contains no pathological transitions then there exists a weakly synchronisable path $\bar{\rho}$ of M that contains every transition of M.*

Proof

Since there are no pathological transitions each self-loop transition $t = (s, s, x/y)$ can be followed by a path $\bar{\rho}_t$ with ending state $s' \neq s$ such that $t\bar{\rho}_t$ is synchronisable. Thus, since M is strongly connected, there exists a path $\bar{\rho}$ that contains every transition and where every self-loop transition t, for state s, is in a synchronisable subpath $t\bar{\rho}_t$ of $\bar{\rho}$ whose ending state is not s. The result thus follows. □

Since we can include every transition of M in a weakly synchronisable path, we can execute every transition of M without encountering controllability problems. There remains the problem of producing such a path that does not allow the possibility of fault masking leading to output faults not being observed. This will be achieved by including the following subpaths for a self-loop transition $t = (s, s, x/y)$ of M.

Definition 9 *Given a self-loop transition $t = (s, s, x/y)$ of M, the following define paths $\mathcal{T}_1(t)$ and $\mathcal{T}_2(t)$.*

(1) Let $t\bar{\rho}_t$ be a minimal synchronisable path whose ending state is not s and $\bar{\rho}_t = \bar{\rho}'_t t'$ for some transition t'. Then we include the path $\mathcal{T}_1(t) = t_1 t\bar{\rho}_t$ for some transition t_1 that is not a self-loop transition. There is a state change in t_1 and the last transition of $\bar{\rho}_t$ and this allows us to conclude that any output between these state changes is in response to the input portion of the label of $t\bar{\rho}'_t$.

(2) A subsequence $\mathcal{T}_2(t) = t_2\bar{\rho}_t$ for a transition t_2 that is not a self-loop transition (t_2 can be the same as t_1). Since t_2 and the last transition of $\bar{\rho}_t$ include a state change, this allows us to check the output of $\bar{\rho}'_t$ in N.

If we consider the self-loop transition of M_0 we can obtain the following.

- $\mathcal{T}_1(t_{1b}) = t_{4b}t_{1b}t_{1a}$. This sequence starts at s_4 and ends at s_2.
- $\mathcal{T}_2(t_{1b}) = t_{4b}t_{1a}$. This sequence starts at s_4 and ends at s_2.
- $\mathcal{T}_1(t_{2b}) = t_{1a}t_{2b}t_{2a}$. This sequence starts at s_1 and ends at s_3.
- $\mathcal{T}_2(t_{2b}) = t_{1a}t_{2a}$. This sequence starts at s_1 and ends at s_3.
- $\mathcal{T}_1(t_{4a}) = t_{3b}t_{4a}t_{4b}$. This sequence starts at s_3 and ends at s_1.
- $\mathcal{T}_2(t_{4a}) = t_{3b}t_{4b}$. This sequence starts at s_3 and ends at s_1.

Proposition 6 *Let $t_1, \ldots, t_k$ denote the self-loop transitions of M, $T = \{\mathcal{T}_1(t_1), \mathcal{T}_2(t_1), \ldots, \mathcal{T}_1(t_k), \mathcal{T}_2(t_k)\}$ and let $T_1, \ldots, T_{2k}$ denote a permutation of $\mathcal{T}_1(t_1), \mathcal{T}_2(t_1), \ldots, \mathcal{T}_1(t_k), \mathcal{T}_2(t_k)$. Let $\bar{\rho} = \bar{\rho}_0 T_1 \bar{\rho}_{11} T_2 \bar{\rho}_{12} \ldots T_{2k-1} \bar{\rho}_{k1} T_{2k} \bar{\rho}_{k2}$ for weakly synchronisable subpaths $\bar{\rho}_0, \bar{\rho}_{11}, \bar{\rho}_{12}, \ldots, \bar{\rho}_{k1}, \bar{\rho}_{k2}$ be a path of M such that every transition of M that is not a self-loop transition is contained in some $\bar{\rho}_{ij}$ in a context in which it is preceded by a transition that is not a self-loop transition, $1 \leq i \leq k$, $1 \leq j \leq 2$. If local status messages are used and $\bar{\rho}_0$ is preceded by a local status message then the input portion of the label of $\bar{\rho}$ is a checking sequence.*

Proof

First, observe that the sequence is weakly synchronisable and so, since local status messages are being used, it can be applied without causing controllability problems.

If a transition leads to a change of state then this change of state is observed by the testers and thus an output fault in such a transition cannot be masked if it is preceded by another transition that involves a change in state.

Consider a self-loop transition $t = (s, s, x/y)$ of M. The path $\bar{\rho}$ includes the subsequences $\mathcal{T}_1(t) = t_1 t \bar{\rho}_t$ and $\mathcal{T}_2(t) = t_2 \bar{\rho}_t$, $\bar{\rho}_t = \bar{\rho}'_t t'$ for a transition t' that is not a self-loop transition. Let u denote the state of N with $r(u) = s$ and let $\bar{x}'$ denote the input portion of the label of $\bar{\rho}'_t$. From no tester observing a failure in response to $t_2 \bar{\rho}_t$, since t_2 involves a state change we can conclude that for all $1 \leq i \leq m$ we have that $\pi_i(\gamma_M(s, \bar{x}')) = \pi_i(\gamma_N(u, \bar{x}'))$. From no tester observing a failure in response to $t_1 t \bar{\rho}_t$ we can conclude that for all $1 \leq i \leq m$ we have that $\pi_i(\gamma_M(s, x\bar{x}')) = \pi_i(\gamma_N(u, x\bar{x}'))$. Further, the input of x when N is in state u leads to no change in state. Thus, since $\pi_i(\gamma_M(s, x\bar{x}')) = \pi_i(x/\lambda(s, x))\pi_i(\gamma_M(s, \bar{x}'))$ and $\pi_i(\gamma_N(u, x\bar{x}')) = \pi_i(x/\lambda_I(u, x))\pi_i(\gamma_N(u, \bar{x}'))$, we must have that for all $1 \leq i \leq m$, $\pi_i(x/\lambda(s, x)) = \pi_i(x/\lambda_I(u, x))$. Thus, from no tester observing a failure in response to $\bar{\rho}$ we can conclude that the output of t is correct.

If the label of $\bar{\rho}$ labels a path in N from u_0 then we can conclude the following:

(1) the self-loop transition are correctly implemented since for each self-loop transition t, $\bar{\rho}$ contains $\mathcal{T}_1(t)$ and $\mathcal{T}_2(t)$.
(2) the transitions of M that are not self-loops are correctly implemented since each such transition t is contained in a subsequence $t't$ of $\bar{\rho}$ for a transition t' that is not a self-loop transition: the observation of a state change in t' immediately precedes the output of t and the observation of a state change in t immediately follows the output of t.

The result thus follows. □

Proposition 7 *If M contains no pathological transitions then there exists a weakly synchronisable path $\bar{\rho}$ of M with input portion $\bar{x}$ such that $\bar{x}$ is a checking sequence.*

Proof

Since M has no pathological transitions, we can define subsequences $\mathcal{T}_1(t)$ and $\mathcal{T}_2(t)$ for each self-loop transition. The result thus follows from M being strongly connected and Proposition 6.

□

4.2 Checking sequence generation

We have seen that, if M has no pathological transitions and we have local status messages then there exists a checking sequence. In this section we show how such checking sequences can be produced.

Let $t_1, \ldots, t_k$ denote the self-loop transitions of M and assume that for all t_i, $1 \leq i \leq k$, a path $\bar{\rho}_{t_i}$ has been defined and thus for all $1 \leq i \leq k$ we have that $t_i\bar{\rho}_{t_i}$ is synchronisable and $\bar{\rho}_{t_i}$ ends in a transition that is not a self-loop. Then we produce a digraph $G' = (V, E')$ in which $V = \{v_1, \ldots, v_n\}$ and $E' = E_1 \cup E_2$ where:

(1) E_1 is the set of edges that represent transitions that are not self-loops. Thus, $E_1 = \{(v_i, v_j, x/y) | s_j = \delta(s_i, x) \neq s_i \wedge y = \lambda(s_i, x)\}$.
(2) E_2 is the set of edges that represent the $\bar{\rho}_{t_i}$ and the $t_i\bar{\rho}_{t_i}$ for the self-loop transitions $t_1, \ldots, t_k$. For each t_i, $1 \leq i \leq k$, let $s_{st(i)}$ denote the start state of t_i and let $s_{f(i)}$ denote the end state of $\bar{\rho}_{t_i}$. Thus $E_2 = \{(v_{st(i)}, v_{f(i)}, label(t_i\bar{\rho}_{t_i})) | 1 \leq i \leq k\} \cup \{(v_{st(i)}, v_{f(i)}, label(\bar{\rho}_{t_i})) | 1 \leq i \leq k\}$.

It is sufficient to find a tour of G' that includes every edge.

Proposition 8 *If local status messages are used, tour Γ of G' contains every edge of E' and $\bar{\rho}$ denotes a path produced by starting Γ at the vertex corresponding to s_0 then the input portion of the label of $\bar{\rho}$, preceded by a status message, is a checking sequence.*

Proof

First note that the structure of G' ensures that if $\bar{\rho}$ is a path of G' that includes an edge representing a transition t then either $\bar{\rho}$ starts with t or in $\bar{\rho}$ the transition t is preceded by a transition that is not a self-loop. Thus, since Γ includes all of the edges of G', the path $\bar{\rho}$ satisfies the conditions of Proposition 6 and so is a checking sequence. □

For M_0, using the sequences given earlier, we can obtain the directed graph shown in Figure 3 in which solid lines represent edges from E_1 and each dotted line represents two edges from E_2.

The following is one possible resultant checking sequence.

$$t_{1a}t_{2a}\mathcal{T}_1(t_{4a})\mathcal{T}_1(t_{2b})\mathcal{T}_2(t_{4a})\mathcal{T}_2(t_{2b})t_{3b}\mathcal{T}_1(t_{1b})t_{2a}t_{3b}\mathcal{T}_2(t_{1b})t_{2a}t_{3a}t_{1a}t_{2a}t_{3b}t_{4b}$$

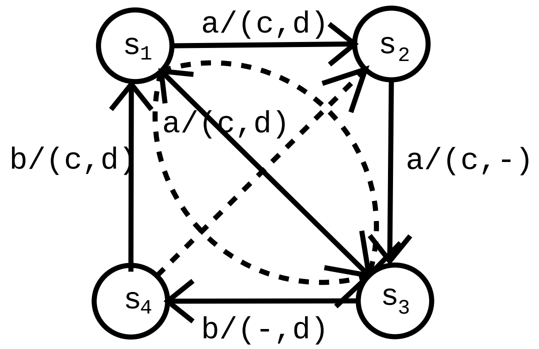

Fig. 3. The digraph for M_0

The problem of finding a lowest cost tour of G', where the cost is the length of the corresponding test sequence, is an instance of the Chinese postman problem (CPP). As noted earlier, the CPP can be solved in low order polynomial time. However, this approach does not utilize potential overlap, between the subsequences used, that has been used when testing from a single-port FSM (see, for example, [25, 33]).

5 Conclusions

If the system under test has multiple ports that are physically distributed then we place a tester at each port in testing. If these testers cannot directly communicate and there is no global clock then we are testing in the distributed test architecture and this introduces controllability and observability problems that cannot, in general, be overcome.

Some systems have a status message: an input that leads to the SUT replying with a message that identifies its current state. This paper has shown how a status message can be used to overcome controllability and observability problems. We have considered two types of status message. The first type can be input at any port and leads to output being sent to all ports. Such status messages can effectively be used in order for one tester to send a message to the other testers via the SUT and so it is possible for any test sequence to be augmented with such status messages in order to overcome controllability and observability problems. The second type of status message, called a local status message, can be input at any port but if it is input at a port p then the response is sent to p only. This type of status message is similar to facilities provided by monitoring systems but cannot be used to overcome controllability and observability problems in arbitrary test sequences. We have shown how a checking sequence, that is guaranteed to lead to a failure if the SUT is faulty, can be produced using such status messages.

There are a number of lines of future work. A distributed system with ports

$p_1, \ldots, p_m$ might be implemented through a set of subsystems with FSM models $M_1, \ldots, M_m$ where the subsystem represented by M_i interacts with the environment at port p_i, $1 \leq i \leq m$. The use of a set of communicating FSMs that run in parallel avoids the state space explosion that occurs when such models are combined. In this context, in order for a subsystem to report the global state it must know the current state of every other subsystem. It may thus be simpler to have a separate status message $\mathcal{S}_i$ for each port p_i where the message $\mathcal{S}_i$ can be input at p_i and the output produced in response to $\mathcal{S}_i$ is the current state of M_i and this is similar to the notion of compositional monitoring [35]. It is relatively straightforward to adapt the notion of weakly synchronized to this case: we require that for any two consecutive transitions tt' in which the input from t' is applied at port p_i, either tt' is synchronisable or t leads to a change in state of the component M_i. It should thus be possible to adapt the approach given in this paper, that uses local status messages in checking sequence generation, to use such status messages. In addition, previous work has looked at timing issues when testing using multiple testers that can communicate directly through an external network [19]. It would be interesting to investigate timing issues when testing using status messages since many systems have real-time constraints. However, it seems likely that similar approaches can be used in order to overcome controllability and observability problems when testing systems with real-time constraints although issues arise if there are timing constraints between events at different ports.

References

[1] Andreas Bauer, Martin Leucker, and Christian Schallhart. Model-based runtime analysis of distributed reactive systems. In *17th Australian Software Engineering Conference (ASWEC 2006)*, pages 243–252. IEEE Computer Society, 2006.

[2] S. Boyd and H. Ural. The synchronization problem in protocol testing and its complexity. *Information Processing Letters*, 40(3):131–136, 1991.

[3] B. Broekman and E. Notenboom. *Testing Embedded Software.* Addison-Wesley, London, 2003.

[4] L. Cacciari and O. Rafiq. Controllability and observability in distributed testing. *Information and Software Technology*, 41(11–12):767–780, 1999.

[5] J. Chen, R. M. Hierons, and H. Ural. Conditions for resolving observability problems in distributed testing. In *24rd IFIP International Conference on Formal Techniques for Networked and Distributed Systems (FORTE 2004)*, volume 3235 of *Lecture Notes in Computer Science*, pages 229–242. Springer-Verlag, 2004.

[6] Jessica Chen, Robert M. Hierons, and Hasan Ural. Resolving observability problems in distributed test architectures. In *Formal Techniques for Networked and Distributed Systems (FORTE 2005)*, volume 3731 of *Lecture Notes in Computer Science*, pages 219–232. Springer, 2005.

[7] W. Chen and H. Ural. Synchronizable checking sequences based on multiple UIO sequences. *IEEE/ACM Transactions on Networking*, 3:152–157, 1995.

[8] Paul S. Dodd and Chinya V. Ravishankar. Monitoring and debugging distributed real-time programs. *Software, Practice and Experience*, 22(10):863–877, 1992.

[9] R. Dssouli and G. von Bochmann. Error detection with multiple observers. In *Protocol Specification, Testing and Verification V*, pages 483–494. Elsevier Science (North Holland), 1985.

[10] R. Dssouli and G. von Bochmann. Conformance testing with multiple observers. In *Protocol Specification, Testing and Verification VI*, pages 217–229. Elsevier Science (North Holland), 1986.

[11] A. Y. Duale and M. U. Uyar. A method enabling feasible conformance test sequence generation for EFSM models. *IEEE Transactions on Computers*, 53(5):614–627, 2004.

[12] M. A. Fecko, M. U. Uyar, A. Y. Duale, and P. D. Amer. A technique to generate feasible tests for communications systems with multiple timers. *IEEE/ACM Transactions on Networking*, 11:796–809, 2003.

[13] Mario Friske and Bernd-Holger Schlingloff. Improving test coverage for UML state machines using transition instrumentation. In *26th International Conference on Computer Safety, Reliability, and Security (SAFECOMP)*, volume 4680 of *Lecture Notes in Computer Science*, pages 301–314. Springer, 2007.

[14] A. Gibbons. *Algorithmic Graph Theory*. Cambridge University Press, 1985.

[15] Dan Gunter, Brian Tierney, Brian Crowley, Mason Holding, and Jason Lee. Netlogger: A toolkit for distributed system performance analysis. In *Proceedings of the 8th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 267–273. IEEE Computer Society, 2000.

[16] S. Guyot and H. Ural. Synchronizable checking sequences based on UIO sequences. In *Protocol Test Systems, VIII*, pages 385–397, Evry, France, September 1995. Chapman and Hall.

[17] R. M. Hierons and H. Ural. Synchronized checking sequences based on UIO sequences. *Information and Software Technology*, 45(12):793–803, 2003.

[18] C.-M. Huang, Y.-I. Chang, and M. T. Liu. A computer-aided incremental protocol test sequence generation: the production system approach. In *IEEE Annual Phoenix Conference on Computers and Communications*, pages 608–614, 1991.

[19] Ahmed Khoumsi. A temporal approach for testing distributed systems. *IEEE Transactions on Software Engineering*, 28(11):1085–1103, 2002.

[20] D. Lee and M. Yannakakis. Principles and methods of testing finite-state machines - a survey. *Proceedings of the IEEE*, 84(8):1089–1123, 1996.

[21] G. Luo, R. Dssouli, and G. v. Bochmann. Generating synchronizable test sequences based on finite state machine with distributed ports. In *The 6th IFIP Workshop on Protocol Test Systems*, pages 139–153. Elsevier (North-Holland), 1993.

[22] G. Luo, R. Dssouli, G. v. Bochmann, P. Venkataram, and A. Ghedamsi. Test generation with respect to distributed interfaces. *Computer Standards and Interfaces*, 16:119–132, 1994.

[23] G. L. Luo, G. v. Bochmann, and A. Petrenko. Test selection based on communicating nondeterministic finite-state machines using a generalized Wp-method. *IEEE Transactions on Software Engineering*, 20(2):149–161, 1994.

[24] Masoud Mansorui-Samani and Morris Sloman. Monitoring distributed systems. *IEEE Network*, pages 20–30, 1993.

[25] R. E. Miller and S. Paul. On the generation of minimal length conformance tests for communications protocols. *IEEE/ACM Transactions on Networking*, 1(1):116–129, 1993.

[26] Omar Rafiq and Leo Cacciari. Coordination algorithm for distributed testing. *The Journal of Supercomputing*, 24(2):203–211, 2003.

[27] A. Rezaki and H. Ural. Construction of checking sequences based on characterization sets. *Computer Communications*, 18(12):911–920, 1995.

[28] B. Sarikaya and G. v. Bochmann. Synchronization and specification issues in protocol testing. *IEEE Transactions on Communications*, 32:389–395, April 1984.

[29] K.-C. Tai and Y.-C. Young. Synchronizable test sequences of finite state machines. *Computer Networks and ISDN Systems*, 30(12):1111–1134, 1998.

[30] Harold W. Thimbleby. The directed chinese postman problem. *Software, Practice and Experience*, 33(11):1081–1096, 2003.

[31] H. Ural and Z. Wang. Synchronizable test sequence generation using UIO sequences. *Computer Communications*, 16(10):653–661, 1993.

[32] M. U. Uyar and A. Y. Duale. Resolving inconsistencies in EFSM modeled specifications. In *IEEE Military Communications Conf. (MILCOM)*, Atlantic City, NJ, October 1999.

[33] B. Yang and H. Ural. Protocol conformance test generation using multiple UIO sequences with overlapping. In *ACM SIGCOMM 90: Communications, Architectures, and Protocols*, pages 118–125, Twente, The Netherlands, September 24-27 1990.

[34] Y. C. Young and K. C. Tai. Observational inaccuracy in conformance testing with multiple testers. In *IEEE 1st workshop on application-specific software engineering and technology*, pages 80–85, 1998.

[35] Mohammad Zulkernine and Rudolph E. Seviora. A compositional approach to monitoring distributed systems. In *International Conference on Dependable Systems and Networks (DSN 2002)*, pages 763–772. IEEE Computer Society, 2002.