FSM Based Interoperability Testing Methods for Multi Stimuli Model

Khaled El-Fakih[1], Vadim Trenkaev[2], Natalia Spitsyna[2], Nina Yevtushenko[2]
[1] American University of Sharjah, PO Box 26666, Sharjah, United Arab Emirates

kelfakih@aus.ac.ae
[2] Tomsk State University, 36 Lenin str., Tomsk, 634050, Russia
snv@kitidis.tsu.ru, {vad, yevtushenko}@elefot.tsu.ru

Abstract. In this paper, we propose two fault models and methods for the derivation of interoperability test suites when the system implementation is given in the form of two deterministic communicating finite state machines. A test suite returned by the first method enables us to determine if the implementation is free of livelocks. If the implementation is free of livelocks, the second method returns a test suite that checks if the implementation conforms to the specification. Application examples are used to illustrate the methods.

1. Introduction

The objective of interoperability testing is to assure that two or more protocol implementations can interact and if so whether they behave together as expected [KSK00, VBT01]. As usual, to guarantee the fault coverage we need a formal model of protocol specifications and implementations as well as a formal model of possible faults. One of the widely used formal models for protocol specification and testing is the Finite State Machine (FSM) model. Then, two communicating protocol implementations can be considered as a system of two communicating FSMs (SCFSM). The FSMs communicate asynchronously via bounded internal queues where messages are stored. We consider the case of *multiple stimuli* [SKC02] where two external messages (multiple stimuli input) from the environment can be sent simultaneously to both protocol implementations. We study some properties of a multiple stimuli SCFSM and we use reachability analysis [Wes78, Boc80, BrZa83] to derive the joint behavior of two communicating FSMs.

Often protocol specifications contain optional commands or options that are not specified or parameters that have no restrictions on their implementations. As a corollary, such specifications are not complete and are described by partial FSMs. On the other hand, the implementations of these machines are complete and usually tested in isolation using the quasi-equivalence conformance relation. According to this relation, for each defined behavior of a protocol specification the corresponding implementation has to have the same behavior. However, the undefined transitions of the protocol specifications can be completed in different ways by different vendors.

This can cause a livelock when an input sequence that traverses undefined transitions is applied to the system implementation. In the first part of the paper, we present a fault model and a method for interoperability testing for livelocks when the system implementation is given in the form of two deterministic communicating finite state machines. A complete test suite detects livelocks (if exist) in any possible system implementation. A livelock is detected by means of a time-out period when traversing a transition that leads to a livelock i.e. a complete test suite has to traverse, for each two possible protocol implementations a transition that can lead to a livelock. Thus, in this case, the considered fault model [STEY03] is different than that usually used in conformance testing. The fault model does not include the specification of the whole system; it only contains the fault domain, i.e. the set of possible protocol implementations. For the compact representation of the fault we use a mutation machine [KPY99].

We note that when an implementation at hand has no livelocks, we are still required to test if it satisfies its specification. Accordingly, in the second part of the paper, we present a related test derivation method. Assuming that the protocol implementations are tested in isolation and found quasi-equivalent to their specification, the test derivation method uses the incremental test derivation methods presented in [EYB02, Elf02] in order to generate tests only for the untested parts of the system implementation. The performed experiments clearly show significant gains in using incremental testing when the tested part of the system implementation consists of up to 80% of the whole implementation.

This paper is organized as follows. Section 2 includes necessary definitions and Section 3 introduces a multiple stimuli model for a system of communicating finite state machines. Section 4 includes a livelock testing method, and Section 5 contains a test derivation method w.r.t. a given specification. Both methods are illustrated using simple application examples. Section 6 concludes the paper

2. Preliminaries

A *finite state machine* (*FSM*) A is a 5-tuple $\langle S,I,O,h,s_0\rangle$, where S is a finite nonempty set with s_0 as the initial state; I and O are input and output alphabets; and $h\subseteq S\times I\times O\times S$ is a behavior relation. The behavior relation defines all possible transitions of the machine. Given a current state s_j and input symbol i, a 4-tuple $(s_j,i,o,s_k)\in h$ represents a transition from state s_j under the input i to the next state s_k with the output o, usually written as $s_j \xrightarrow{i/o} s_k$.

We assume that a FSM A has a *reset capability*, i.e. there is a special reset input "r" that takes the FSM from any state to the initial state. As usually, we assume that each transition with the reset input is correctly implemented, i.e. we do not include the reset input into the input alphabet I.

A transition from a state s_j under input symbol i is called *deterministic* if there exists the only pair (o,s_k) such that $(s_j,i,o,s_k)\in h$. If FSM A has only deterministic transitions then FSM A is said to be *deterministic*; otherwise, A is *non-deterministic*. In the deterministic FSM A instead of behavior relation h we use two functions: *transition function* $\psi: D_A\subseteq S\times I\rightarrow S$ and *output function* $\varphi: D_A\subseteq S\times I\rightarrow O$ where D_A

is called the *specification domain* of the FSM. Therefore, in general, a deterministic FSM is a 7-tuple $\langle S,I,O,\psi,\varphi,D_A,s_0\rangle$. An FSM is called *Chaos* if it has only chaos transitions, i.e. if $h=S\times I\times O\times S$. When at least one of the sets S, I and O is not a singleton a chaos FSM is non-deterministic.

If for each pair $(s,i)\in S\times I$ there exists $(o,s')\in O\times S$ such that $(s,i,o,s')\in h$ then FSM A is said to be *complete*; otherwise, A is *partial*. For a complete deterministic FSM, the specification domain D_A coincides with the Cartesian product $S\times I$, i.e. a complete deterministic FSM is a 6-tuple $\langle S,I,O,\psi,\varphi,s_0\rangle$.

FSM $B=\langle S',I,O,g,s_0\rangle$, $S'\subseteq S$, is a *submachine* of FSM $A=\langle S,I,O,h,s_0\rangle$ if $S'\subseteq S$ and $g\subseteq h$, i.e. if each transition of FSM B is obtained by fixing an appropriate transition of the FSM A. Given a complete FSM A, we let $Sub(A)$ denote the set of all complete deterministic submachines of A.

In usual way, the behavior relation is extended to input and output sequences. Given state $s\in S$, input sequence $\alpha=i_1i_2\ldots i_k\in I^*$ and output sequence $\beta=o_1o_2\ldots o_k\in O^*$, the input-output sequence $i_1o_1i_2o_2\ldots i_ko_k$ is called a *trace of A at state s* if there exists state s' such that $(s,i_1i_2\ldots i_k,o_1o_2\ldots o_k,s')\in h$, i.e. there exist states $s_1=s$, s_2, $\ldots$, s_k, $s_{k+1}=s'$ such that $(s_i,i_i,o_i,s_{i+1})\in h$, $i=1$, $\ldots$, k. A trace at the initial state is simply called a *trace of A*.

Given deterministic FSMs B and A and states t of FSM B and s of FSM A, state t is *quasi-equivalent* to s, written $t\approx_{quasi}s$, if the set of traces of FSM B at state t contains that of FSM A at state s. If the sets of traces at states t and s coincide, then states t and s are *equivalent*, written $s\cong t$. FSM B is *quasi-equivalent* to A, written $B\approx_{quasi}A$, if the set of traces of FSM B contains that of A. FSMs A and B are *equivalent*, written $A\cong B$, if their sets of traces coincide.

3. Multi Stimuli Model of a System of Communicating Finite State Machines

3.1. A system of communicating FSMs

Many complex systems are typically specified as a collection of communicating components. We consider here a system that consists of two communicating FSMs (SCFSM) (Fig.1). We let the alphabets $I_1\cup I_2$ and $O_1\cup O_2$ represent the externally observable input/output actions (or messages) of the system, while the alphabets E_1 and E_2 represent the internal (hidden) input/output interactions between the two component FSMs. The FSMs communicate asynchronously via bounded internal queues where messages are stored. We consider the case of *multiple stimuli* [KSK02] where simultaneously two external inputs (multiple stimuli input) from the environment can be sent to both component machines. Moreover, in response to an input each component machine can produce a pair of outputs, one to the environment and one to other component machine [TKS03]. We also assume that the system works in a *slow environment* [PYBD96]. This means that the next external input is applied only when the processing of previous external input by the system has been completed, i.e. when the internal queues become empty. Due to this assumption, if the system queues are empty and a multiple stimuli input is applied to the system, each internal queue can get a message. After the processing an internal message by one of the component machines one of the queues will become empty while another message can be added to the input queue of the other component machine. In this case the component machine that has two messages in its input queue processes one of these messages and as a corollary it can produce an input message to the other component machine. Thus, at any time, the length of the input queues does not exceed two.

Under the above assumptions, the collective behavior of the two communicating FSMs can be described by a finite *composed machine* that describes the observable behavior of the system. The composed machine is obtained from a reachability graph [Wes78, BoSu80, BrZa83] that described the collective behavior of the system components in terms of internal and external actions of the system. In the following subsection we give the details of building a reachability graph and a composed machine.

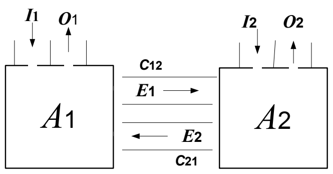


Fig.1 A system of two communicating finite state machines

We note that after submitting an appropriate external input to the system, i.e. when the input queues are empty, the two component machines can carry on an infinite internal dialogue. In this case we say that the system falls into a livelock. Here, as in the single stimuli mode, a livelock of the system can result in the absence of an external output at least at one external port. Moreover, differently from the single stimuli mode [PYBD96], we also have another type of livelocks that occurs when one of the system components produces an infinite external sequence.

If the system can fall into livelock under an appropriate input sequence then the composed machine enters the designated *Livelock* state with the designated *livelock* output [STEY03]. In this case, the corresponding transition of the composed machine is called *suspicious* and takes the machine to the designated *Livelock* state.

3.2. Reachability graph and composed FSM

Formally, we consider a system of two communicating FSMs $A1=\langle Q,I_1\cup E_2,O_1\times E_1,h_1,q_0\rangle$ and $A2=\langle T,I_2\cup E_1,O_2\times E_2,h_2,t_0\rangle$ (Fig. 1) where the channel C12 (C21) is a FIFO queue linking the FSM $A1$ ($A2$) to the FSM $A2$ ($A1$). Thus, the FSM $A1$ has $I_1\cup E_2$ as the set of inputs and $O_1\times E_1$ as the set of outputs and the FSM $A2$ has $I_2\cup E_1$ as the set of inputs and $O_2\times E_2$ as the set of outputs. The alphabets I_1, I_2 and O_1, O_2 represent the *externally* observable input/output actions of the system, while the alphabets E_1 and E_2 represent the *internal* input/output interactions between the two component machines that are non-observable (hidden). As in [PYBD96] we assume that all the alphabets are pair-wise disjoint.

In order to deal with the situation where a component FSM in response to an input produces only an internal or an external output, we assume that the alphabets O_1, E_1, O_2 and E_2 include the silent message ε. Thus, the output pair $(o,\varepsilon)\in O_1\times E_1$ corresponds to the situation where $A1$ produces only the external output o to the environment.

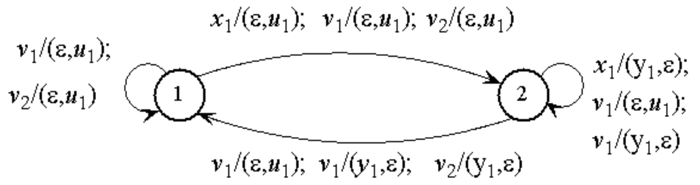
To describe the joint behavior of a SCFSM we build a reachability graph G [Wes78, BoSu80, BrZa83]. The reachability graph G is a pair (V,E), where the set V of vertices represents the set of so-called *global states* of the system. The set E of edges represents transitions between global states. A global state of a SCFSM is a 4-tuple (q,t,c_{12},c_{21}) where $q\in Q$, $t\in T$, $c_{12}\in E_1^2$ and $c_{21}\in E_2^2$ are the contents of the internal queues C12 and C21, respectively, where E^2 is the set of all sequences over the alphabet E of length at most two. A global state is called *stable* if all internal queues are empty. Otherwise, it is called *transient*.

Under the above assumptions, a component machine of SCFSM can produce a pair of outputs in response to an input. By this reason, given a stable state and an external input, the system can produce a pair of external output sequences. In case of finite dialogue, the length of these sequences cannot exceed an appropriate integer k. In case of infinite dialogue, the system falls into a livelock, i.e. the system enters the designated *Livelock* state. This happens when at least one component machine of the system does not produce an external output or produces an infinite sequence of external outputs. As usual we assume that a livelock can be detected by means of a timer. In other words, if after an appropriate period of time the system does not produce any external output sequence in at least one of its external ports or it continues producing output actions, then we conclude that the system falls into a livelock.

Given a SCFSM of $A1$ and $A2$, in order to derive the composed machine we construct a reachability graph G that describes the joint behavior of $A1$ and $A2$ under single inputs of the sets I_1 and I_2 and under multi stimuli inputs of the set $I_1\times I_2$. The externally observable behavior of the SCFSMs, i.e. the composed machine $A1\Diamond A2$, can be obtained from the reachability graph by hiding all internal actions and pairing inputs with corresponding output sequences of length up to k similar to the single stimuli model [PYBD96]. Each transition of the FSM $A1\Diamond A2$ has $i_1i_2\in I_1\times I_2$, $i_1\in I_1$, or $i_2\in I_2$ as an input label and as an output label it has the designated *livelock* output, in case the transition leads to the designated *Livelock* state, or a pair of finite output sequences (β,γ) of length at most k, where β is defined over the external alphabet O_1 and γ is defined over the external alphabet O_2.

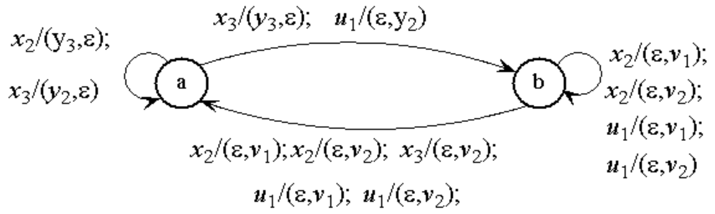
Given a state of the composed machine $A1\Diamond A2$ and an external (single or multiple stimuli) input, if there exists a path in the reachability graph that starts at the state and includes a cycle with only transient states, then the system falls into *livelock* at the state when the input that labels the head transition of the path is applied. In this case, the composed machine includes a corresponding suspicious transition to the designated *livelock* state labeled with the given input and the designated *livelock* output. Thus, the composed FSM $A1\Diamond A2$ under a given input either transits to the livelock state producing the livelock output or it transits to another global state producing a pair of finite output sequences.

As an example, consider the FSMs *MM*1 and *MM*2 shown in Figures 2 and 3 below. The sets of external inputs and outputs of *MM*1 are $\{x1, \varepsilon\}$ and $\{y_1,\varepsilon\}$, and the sets of external inputs and outputs of *MM*2 are $\{x_2, x_3, \varepsilon\}$ and $\{y_2,y_3,\varepsilon\}$. The set of internal inputs of *MM*1 (internal outputs of *MM*2) is $\{v_1,v_2,\varepsilon\}$, and the set of internal inputs of *MM*2 (internal outputs of *MM*1) is $\{u_1,\varepsilon\}$. Figure 4 shows a part of the reachability graph of FSMs *MM*1 and *MM*2. For example, from state 1*a* under the input (x_1x_3) the system can reach the stable state 2*b* and produce the output pair $(\varepsilon,y_2\ y_2)$ or it can fall into livelock. Thus, in the corresponding composed machine *MM*1◊*MM*2, we add an outgoing suspicious transition from state 1*a* to the livelock state labeled with the input/output (x_1x_3)/*Livelock*. Similarly, we include the following suspicious transitions in *MM*1◊*MM*2. From state 1*b*, transitions x_1/*Livelock* and x_2/*Livelock*. From state 2*b*, transitions x_2/*Livelock* and (x_1x_2)/*Livelock*. The composed machine *MM*1◊*MM*2 is shown in Figure 5.



$x_1/(\varepsilon,u_1)$; $v_1/(\varepsilon,u_1)$; $v_2/(\varepsilon,u_1)$
$v_1/(\varepsilon,u_1)$;
$v_2/(\varepsilon,u_1)$
1
2
$x_1/(y_1,\varepsilon)$;
$v_1/(\varepsilon,u_1)$;
$v_1/(y_1,\varepsilon)$
$v_1/(\varepsilon,u_1)$; $v_1/(y_1,\varepsilon)$; $v_2/(y_1,\varepsilon)$

Fig. 2. FSM *MM*1



$x_3/(y_3,\varepsilon)$; $u_1/(\varepsilon,y_2)$
$x_2/(y_3,\varepsilon)$;
$x_3/(y_2,\varepsilon)$
a
b
$x_2/(\varepsilon,v_1)$;
$x_2/(\varepsilon,v_2)$;
$u_1/(\varepsilon,v_1)$;
$u_1/(\varepsilon,v_2)$
$x_2/(\varepsilon,v_1)$;$x_2/(\varepsilon,v_2)$; $x_3/(\varepsilon,v_2)$;
$u_1/(\varepsilon,v_1)$; $u_1/(\varepsilon,v_2)$;

Fig. 3. FSM *MM*2

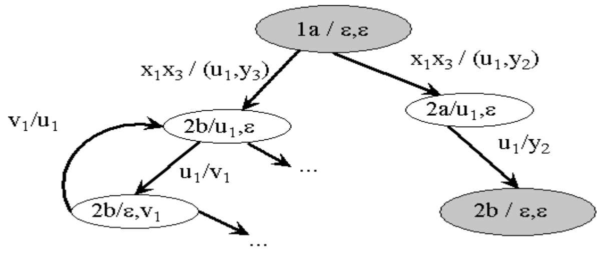
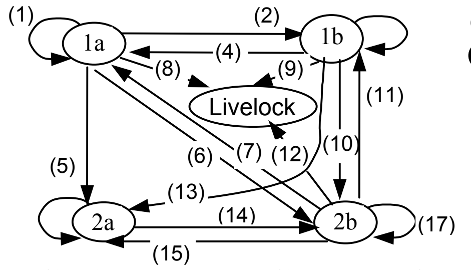


Fig. 4. Part of reachability graph



(1) $x_2/(\varepsilon,y_3)$; $x_3/(\varepsilon,y_2)$; $x_1x_3/(y_1,y_3)$
(2) $x3/(\varepsilon,y_3)$; $x_1x_3/(y_1,y_3)$;
$x_1x3/(\varepsilon,y_2y_3)$
(3) $x_1/(y_1,\varepsilon),(\varepsilon,y_2)$; $x_2/(y_1,\varepsilon),(\varepsilon,y_2)$;
$x_3/(\varepsilon,y_2)$; $x_1x_2/(y_1y_1, \varepsilon)$;
$x_1x_2/(y_1,y_2)$;
$x_1x_2/(\varepsilon,y_2y_2)$; $x_1x_3/(y_1,y_2)$

(4) $x_2/(y_1,\varepsilon)$; $x_1/(y_1,\varepsilon)$; $x_1x_2/(y_1y_1,\varepsilon)$; $x_1x_2/(y_1,y_2)$; **(5)** $x_1x_3/(y_1,y_3)$
(6) $x_1/(\varepsilon,y_2)$; $x_1x_2/(\varepsilon,y_2y_3)$; $x_1x_3/(\varepsilon,y_2y_3)$; $x_1x_3/(\varepsilon,y_2y_2)$; $x_1x_3/(y_1,y_3)$
(7) $x_2/(y_1,\varepsilon)$; $x_3/(y_1,\varepsilon)$; $x_1x_2/(y_1y_1,\varepsilon)$; $x_1x_3/(y_1y_1,\varepsilon)$
(8) $x_1x_3/Livelock$ **(9)** $x_1x_2/Livelock$; $x_1/Livelock$; $x_2/Livelock$
(10) $x_1/(y_1,\varepsilon),(\varepsilon,y_2)$; $x_2/(\varepsilon,y_2)$; $x_3/(\varepsilon,y_2)$; $x_1x_2/(y_1y_1,\varepsilon)$; $x_1x_2/(y_1,y_2)$; $x_1x_2/(\varepsilon,y_2y_2)$
(11) $x_2/(y_1,\varepsilon),(\varepsilon,y_2)$; $x_1x_2/(y_1y_1,\varepsilon)$; $x_1x_2/(y_1,y_2)$ **(12)** $x_1x_2/Livelock$; $x_2/Livelock$
(13) $x_1/(y_1,\varepsilon)$; $x2/(y_1,\varepsilon)$ **(14)** $x_3/(\varepsilon,y_3)$; $x_1x_3/(y_1,y_3)$ **(15)** $x_2/(y_1,\varepsilon)$; $x_1x_2/(y_1y_1,\varepsilon)$
(16) $x_1/(y_1,\varepsilon)$; $x_2/(\varepsilon,y_3)$; $x_3/(\varepsilon,y_2)$; $x_1x_2/(y_1,y_3)$; $x_1x_3/(y_1,y_2)$
(17) $x_1/(y_1,\varepsilon)$; $x_2/(y_1,\varepsilon),(\varepsilon,y_2)$; $x_1x_2/(y_1y_1,\varepsilon)$; $x_1x_2/(y_1,y_2)$

Fig. 5. The composed FSM $MM1\Diamond MM2$

The composition of two component machines can be partial or complete, deterministic or non-deterministic depending on these machines. Here we note that differently from the single stimuli mode in the multi stimuli mode at each transient state one of the component machines can be faster than the other in producing a response to an applied input or both component machines can produce simultaneously their outputs. However, according to the following proposition, if the component machines are deterministic then their composed machine is also deterministic.

Proposition 1. If the component machines $A1$ and $A2$ of a SCFSM are deterministic and the system does not fall into a livelock, then the composed FSM $A1\Diamond A2$ is deterministic.

Proof. Let $(q,t,\varepsilon,\varepsilon)$ be a stable state of the system. Consider the mode "multiple external input", i.e. the case when the composed FSM has an input $i_1i_2\in I_1\times I_2$. The cases with a single stimulus from the environment can be proved in the same way.

Let under the input i_1 and i_2 the FSMs $A1$ and $A2$ at a states q and t produce the output pairs (o_1,e_1) and (o_2,e_2) and enter states q' and t'. Then in the reachability graph there is the transition labeled with $i_1i_2/(o_1,e_1)\&(o_2,e_2)$ from the node $(q,t, \varepsilon,\varepsilon)$ to the transient node (q',t',e_1,e_2). Since the FSM $A1$ and the FSM $A2$ communicate asynchronously then there are three cases: one of the FSMs $A1$ or $A2$ starts to work first, or the two FSMs work simultaneously. To prove the statement it is enough to show that in all cases the system enters one and the same state with the same pair of external outputs. Let under the input e_2 the FSM $A1$ at state q' produce the output pair (o_3,e_3) and enter the state q''. Let also under the input e_1 the FSM $A2$ at a state t' produce the output pair (o_4,e_4) and enter the state t''. If FSM $A1$ starts to work first then in the reachability graph there is the transition labeled with $e_2/(o_3,e_3)$ from the state (q',t',e_1,e_2) to the state $(q'', t', e_1e_3, \varepsilon)$. Now, since the channel queue C21 is empty, the stimulus e_1 is taken from the queue of the channel C12, and FSM $A2$ starts to work, i.e., in the reachability graph there is the transition labeled with $e_1/(o_4,e_4)$ from the node $(q'', t', e_3e_1, \varepsilon)$ to the node (q'', t'', e_3,e_4). Thus, the system enters the transient state (q'', t'', e_3, e_4) with the external output (o_3,o_4). By direct inspection, one can assure that we have the same next state and the same external output when the FSM $A2$ starts to work first. Let now FSMs $A1$ and $A2$ work simultaneously. Then in the reachability graph there is the transition labeled $e_1e_2/(o_4,e_4)$ & (o_3,e_3) from the node (q',t',e_1,e_2) to the node (q'', t'', e_3, e_4) and the external output (o_3,o_4).

Thus the reachability graph has three different paths from the node (q',t',e_1,e_2) to the node (q'', t'', e_3, e_4) with the same external output (o_3,o_4), i.e. the composed machine is deterministic.

❑

4. Testing Livelocks

We recall that one of the purposes of interoperability testing is to test if the joint behavior of two component implementations has no livelocks. As usual, to guarantee complete fault coverage we need a formal model of possible faults. In general the traditional fault model <specification, conformance relation, fault domain> [PYB96] is used. However, in order to test for livelocks, we do not need the specification of the whole system. Accordingly, in the following we introduce a more general fault model and a method for complete test derivation w.r.t. this model.

4.1 A Fault Model for Livelock Testing

Often protocol specifications contain optional commands or options that are not specified or parameters that have no restrictions on their implementations. As a corollary, such specifications are not complete and are described by partial FSMs. Thus, hereafter, we consider two partial deterministic component specifications $A1$ and $A2$. The implementations of $A1$ and $A2$ are usually tested in isolation using the quasi-equivalence conformance relation. According to this relation, for each defined behavior of $A1$ (or $A2$) the corresponding implementation has to have the same behavior. However, the undefined transitions of $A1$ and $A2$ can be completed in different ways by different vendors. When there are no restrictions imposed, the designers can complete the undefined transitions according to their preferences. This can cause a livelock when an input sequence that traverses undefined transitions is applied to the system implementation.

Formally, we consider two deterministic partial component specifications $A1$ and $A2$ and we assume that their implementations are complete and deterministic. We let $\Re1$ and $\Re2$ denote the sets of all possible complete deterministic implementations of $A1$ and $A2$. We assume that each machine of the sets $\Re1$ ($\Re2$) is quasi-equivalent to the corresponding partial specification $A1$ ($A2$). An implementation system is the composition of two complete deterministic FSMs of the sets $\Re1$ and $\Re2$, i.e.

$\Re$ = {$Imp1 \Diamond Imp2$ | $Imp1 \in \Re1$, $Imp2 \in \Re2$}. Thus, the set $\Re$ is the set of all possible system implementations.

We say that *a test suite is complete* w.r.t. the fault model <$\Re$, livelock-free> if the test suite detects each system implementation that falls into a livelock under some input sequence. Usually a livelock is detected by means of a timer. Therefore, in order to detect a livelock, it is sufficient to traverse a transition of an implementation system that leads to a livelock. In other words, a test suite is complete w.r.t. the fault model <$\Re$, livelock-free> if for each possible system implementation *Imp* of the set $\Re$ that has transitions leading to a livelock, the test suite traverses at least one of these transitions.

A straightforward approach for deriving a complete test suite w.r.t. the fault model <$\Re$, livelock-free> is to explicitly enumerate all possible system implementations and for each implementation with at least one transition leading to a livelock to derive an input sequence that traverses one of these transitions. However, in order to avoid the explicit enumeration of all possible implementation machines, a mutation machine [KPY99] can be used.

The fault domain $\Re j$ of a component machine *Aj*, *j* = 1,2, can be described by a complete mutation machine *MMj*. This mutation machine is obtained from *Aj* by completing its undefined transitions in all possible ways, i.e. for each undefined transition of *Aj* we add new transitions to all possible states with all possible outputs, or due to the imposed restrictions.

Given a mutation machine MM_j, *j*=1,2, the set of all deterministic submachines of *MMj* coincides with the set $\Re j$. That is the set $\Re1$ = *Sub*(*MM*1) and the set $\Re2$ = *Sub*(*MM*2). Thus, each possible implementation system is a submachine of the machine $MM \cong MM1 \Diamond MM2$. Accordingly, we can use the fault model <*Sub*(*MM*), livelock-free> [STEY03] for livelock testing. In the following subsection we present a method for deriving a complete test suite w.r.t. the model <*Sub*(*MM*), livelock-free> without enumerating submachines of the mutation machine *MM*.

4.2 Test Suite Derivation Method

State *s* of an FSM *A* is *reachable* if there exists an input sequence that takes the FSM from the initial state to *s*. If state *s* of FSM *A* is reachable while traversing only deterministic transitions then *s* is said to be *deterministically reachable*. In this case, we call an input sequence that takes *A* from the initial state to *s* while traversing only deterministic transitions a *deterministic transfer* sequence for state *s*. The set of deterministic transfer sequences for all deterministically reachable states of *A* is called a *deterministic cover set* of *A*. Moreover, given the FSM $MM \cong MM1 \Diamond MM2$, we let MM_{NoLTr} denote the submachine obtained from *MM* by deleting all transitions leading to the *Livelock* state.

In order to derive a complete test suite w.r.t. the fault model <*Sub*(*MM*), livelock-free>, we consider the following cases.

Extreme Case 1. There are no suspicious transitions in *MM*. That is, the composition of any two complete sub-machines of *MM*1 and *MM*2 does not fall into livelock. In this case, we do not need to test for livelocks.

Extreme Case 2. Each state of the submachine MM_{NoLTr} is reachable via deterministic transitions. In this case, for each outgoing suspicious transition of the state pair *st* of *MM* labeled with the input *x*, we include in the test suite the input sequence $r.\alpha x$ where α is an input sequence that deterministically takes the submachine MM_{NoLTr} from its initial state to the state *st*.

General Case. Generally, not each suspicious transition of the submachine *MM* is deterministically reachable. In this case, we derive a complete test suite as follows.

Algorithm 1. Test suite derivation algorithm
Input: A non-deterministic mutation machine $MM \cong MM1 \Diamond MM2$
Output: A complete test suite w.r.t. the fault model <*Sub*(*MM*), livelock-free>
Step 1. Determine the minimal length deterministic cover set *D* of the submachine MM_{NoLTr}, let $m=|D|$. Moreover, let $\alpha_j \in D$ be a deterministic transfer sequence for state s_j.
Step 2. For each deterministically reachable state s_j of MM_{NoLTr} we derive a traversal set $Tr(s_j)$ in the following way. Let α be an input sequence such that the length of α is not greater than n-m+1, where *n* is number of states of the FSM *MM*. We include α into $Tr(s_j)$ if there exists an output sequence β of *MM* to α such that the following conditions hold:

- the trace α/β does not traverse twice a state of *MM*,
- α/β does not traverse a deterministically reachable state of MM_{NoLTr},
- the last transition traversed by α/β is suspicious.

Step 3. For each deterministically reachable state s_j of the submachine MM_{NoLTr} we derive the set $E_j = r.\alpha_j.Tr(s_j)$. The test suite is the union of the sets E_j over all deterministically reachable states.

Proposition 2. The test suite returned by Algorithm 1 is complete w.r.t. the fault model <*Sub*(*MM*), livelock-free>.

As an example, consider the mutation machine $MM = MM1 \Diamond MM2$ shown in Figure 5. States 1*a* and 2*b* are deterministically reachable from the initial state through the inputs ε and x_1. Thus, we derive the traversal sets for the states 1*a* and 2*b*. The input sequences in the traversal set for state 1*a* have to traverse the outgoing suspicious transitions of state 1*b*. These are transitions labeled with the inputs x_1, x_2 and (x_1x_2). State 1*b* can be reached from state 1*a* under the inputs x_3 and (x_1x_3). Therefore, $Tr(1a)=\{x_3.x_1,\ x_3.x_2,\ x_3.(x_1x_2),\ (x_1x_3).x_1,\ (x_1x_3).x_2,\ (x_1x_3).(x_1x_2)\}$. Moreover, the input sequences of the traversal set for state 2*b* have to traverse the outgoing suspicious transitions of 1*b*. In addition, they have to traverse the outgoing suspicious transitions of state 2*b*. These are the transitions labeled with the inputs x_2 and (x_1x_2). State 1*b* can be reached from state 2*b* under the inputs x_2 and x_1x_2. Thus, $Tr(2b)=\{x_1, x_2, (x_1x_2)\} \cup \{x_2.x_1,\ x_2.x_2,\ x_2.(x_1x_2),\ (x_1x_2).x_1,\ (x_1x_2).x_2,\ (x_1x_2).(x_1x_2)\}$. Therefore, Algorithm 1 returns the complete test suite, $\{r.x_3.x_1,\ r.x_3.x_2,\ r.x_3.(x_1x_2),\ r.(x_1x_3).x_1,\ r.(x_1x_3).x_2,\ r.(x_1x_3).(x_1x_2)\} \cup \{r.x_1.x_2,\ r.x_1.(x_1x_2),\ r.x_1.x_2.x_1,\ r.x_1.x_2.x_2,\ r.x_1.x_2.(x_1x_2),\ r.x_1.(x_1x_2).x_1,\ r.x_1.\ (x_1x_2).x_2,\ r.x_1.(x_1x_2).(x_1x_2)\}$.

5. Testing w.r.t. Specification

If a system implementation is free of livelocks, we are still required to test if it satisfies the specification. Given the partial specifications *A*1 and *A*2 of the communicating protocol entities and their corresponding implementations *Imp*1 and *Imp*2, we assume that *Imp*1 and *Imp*2 are deterministic, complete, tested in isolation and found quasi-equivalent to *A*1 and *A*2. Thus, we assume that the joint behavior of the complete protocol implementations (i.e. the system implementation) is checked w.r.t. the defined behavior of the partial specifications. Given the specification *Spec* of the whole system, we are required to determine if $Imp1\Diamond Imp2 \cong Spec$.

Here we note that *Spec* of a given SCFSM can be obtained in various ways. For example, *Spec* can be derived based on our knowledge how the whole SCFSM has to work. In this paper *Spec* is assumed to be deterministic and complete[1]. However, the components implementations can be completed in different ways by different vendors. Since *Imp*1 and *Imp*2 were tested in isolation and found quasi-equivalent to *A*1 and *A*2, we assume that $Imp1\Diamond Imp2$ is quasi-equivalent to *Spec*. However, the behavior of the complete implementation machine $Imp1\Diamond Imp2$ has also to be tested w.r.t. the specification under undefined input sequences. In this case, the incremental testing methods [EYB02, Elf02] are known to return shorter test suites than the W[Chow78], Wp[Fuj91], or HIS[PYLD93] methods. If the fault domain is represented as the set of deterministic submachines of an appropriate mutation machine, then the length of a test suite returned by incremental testing methods is known to essentially depend on the number of deterministic transitions in the mutation machine. By this reason, in this paper, we divide the fault domain into three parts assuming that the implementation of at most one component machine can be faulty or that both component implementations can be faulty. To do this we augment the given partial specification machines *A*1 and *A*2 according to our preference and we obtain *CompA*1 and *CompA*2 as the complete forms of *A*1 and *A*2. In the following two subsections we present a fault model and a test derivation method based on the above assumptions.

5.1 A Fault Model for Testing w.r.t. Specification

Let *Imp*1 and *Imp*2 be two deterministic complete implementations of the partial deterministic protocol specifications *A*1 and *A*2. We recall that *Imp*1 and *Imp*2 are submachines of the mutation machines *MM*1 and *MM*2. In order to determine if the joint behavior of *Imp*1 and *Imp*2, i.e. $Imp1\Diamond Imp2$, is equivalent to the reference specification *Spec*, we use a traditional fault model $<Spec, \cong, Sub(MM)>$, where the fault domain is the set *Sub*(*MM*) of all deterministic submachines of the mutation machine $MM \cong MM1\Diamond MM2$.

Here we reasonably assume that both implementations can be faulty. A *test suite is complete* w.r.t. the fault model $<Spec, \cong, Sub(MM)>$ if the test suite detects each

[1] In the general case an implementation system can be tested w.r.t. the reduction relation since there can occur several options of the behavior under undefined input sequences.

system implementation that is not equivalent to *Spec*. In the following subsection we derive a complete test suite w.r.t. to this fault model.

5.2 Test Derivation Method

In order to generate a complete test suite for the fault model is <*Spec*, ≅, *Sub*(*MM*)>, one can use the known W[Chow78], Wp[Fuj91], or HIS[PYLD93] test derivation methods assuming an upper bound *m* on the number of states of the implementation system is given. This bound can be calculated as the number of states in the composed system *A*1◊*A*2. However, these methods generate tests not only for *Sub*(*MM*) but also for every possible implementation with up to *m* states. Thus, we need a more appropriate approach that generate tests for the domain fault domain *Sub*(*MM*) taking into account the fact that *Imp*1 and *Imp*2 are tested in isolation and found quasi-equivalent to *A*1 and *A*2, i.e. the machine *Sub*(*MM*) has many deterministic transitions. In other words, an approach based on the incremental testing methods presented in [EYB02,Elf02] can be effectively used. These methods generate tests that check the untested parts of an implementation utilizing some information from the tested parts. However, since the lengths of the test suites generated using the incremental methods significantly depend on the number of nondeterministic transitions of *MM*, which can be too many, we consider three subdomains of the fault domain *Sub*(*MM*). Then, we generate tests, using the incremental testing methods, for one subdomain and we reduce, using the reduction algorithm presented in [EPYB03], the other domains based on the expected behavior of the implementation system (or System Under Test (SUT)) to these tests. In other words, we delete from other subdomains nonconforming submachines that are detected with the derived part of a test suite. Particularly, we consider the fault subdomains *Sub*(*MM*1◊*CompA*2) where *Imp*2 is assumed to be fault free, i.e. *Imp*2≅*CompA*2, and the subdomain *Sub*(*CompA*1◊*MM*2), where *Imp*1 is assumed to be fault free. We generate incremental tests for the subdomain *Sub*(*MM*1◊*CompA*2) and we use these tests to reduce the mutation machines *MM* and *CompA*1◊*MM*2 [EPYB03]. Then, we derive tests for the reduced subdomain of *Sub*(*CompA*1◊*MM*2) and we use these tests to reduce *MM*. Finally, we generate tests for the fault domain *Sub*(*MM*′), where *MM*′ is a reduced submachine of the initial mutation machine *MM*. The details of the method are presented in the algorithm given below.

Here we note that in order to assess the gains of using incremental testing v.s. complete testing of the whole system implementation, we have implemented and experimented with the methods presented in [EYB02]. The experiments show that when the tested part is up to 95% of the whole implementation, on average, the HIS based test suites are 36 times bigger than the corresponding incremental test suites. Moreover, these test suites are on average 11.3, 6.1, and 4.0 times bigger when the tested parts are up to 90%, 85%, and 80% respectively. Moreover, the experiments showed that the ratios of the lengths of the test suites do not significantly depend on the size of specifications.

Algorithm 2. Test suite derivation algorithm
Input: A specification of the whole system *Spec*, partial deterministic components *A*1 and *A*2, and their completed forms *CompA*1 and *CompA*2.
Output: A complete test suite *TS* w.r.t. the fault model <*Spec*, ≅, *Sub*(*MM*).

Step 1. Derive *MM*1 and *MM*2 by completing in all possible ways (or due to some preferences) all the undefined transitions of *A*1 and *A*2. Then, derive the mutation machines *MM* ≅ *MM*1◊*MM*2, *MM*1◊*CompA*2, and *CompA*1◊*MM*2.

Step 2. Use an incremental test derivation method for deriving the complete test suite TS_1 w.r.t. the fault model <*Spec*, ≅, *Sub*(*MM*1◊*CompA*2)>.
Reduce *MM* and *CompA*1◊*MM*2 and obtain *MM*′ and *F*2 using TS_1 and the expected output behavior of the SUT to TS_1.

Step 3. Use an incremental test derivation method for deriving the complete test suite TS_2 w.r.t. the fault model <*Spec*, ≅, *Sub*(*F*2)>.
Reduce *MM*′ and obtain *MM*″ using TS_2 and the expected output behavior of the SUT to TS_2.

Step 4. Use an incremental test derivation method for deriving the tests suite TS_3 w.r.t. the fault model <*Spec*, ≅, *Sub*(*MM*″))>.
Output $TS = TS_1 \cup TS_2 \cup TS_3$

Proposition 3. The test suite *TS* generated using Algorithm 2 is complete w.r.t. the fault model <*Spec*, ≅, *Sub*(*MM*)>.

As an application example, consider the partial deterministic component machines *A*1 and *A*2, showı Figures 6.1 and 6.2, respectively. The set of external inputs and outputs of *A*1 are $X=\{x_1,\varepsilon\}$ and $Y=\{y_1,\varepsilon\}$. The set of external inputs and outputs of *A*2 are $I=\{i, \varepsilon\}$ and $O=\{o,\varepsilon\}$, the set of internal inputs of *A*1 (internal outputs of *A*2) is $V=\{v_1,v_2,\varepsilon\}$, and the set of internal inputs of *A*2 (internal outputs of *A*1) is $U=\{u,\varepsilon\}$. The initial state of *A*1 is the state labeled by "1".

	1	2
x	1/(ε, u)	2/(y, ε)
v_1	2/(ε, u)	
v_2	1/(y, ε)/	2/(y, ε)

Fig. 6.1 Machine *A*1

	a
i	
u	a/(o, v_2)

Fig. 6.2 Machine *A*2

	1a	2a
x	1a/(y, o)	2a/(y,ε)
i	2a(y, o)	1a/(yy, o)
xi	2a/(yy, oo)	1a/(yyy, o)

Fig. 6.3 The machine *Spec*

	1	2
x	1/(ε, u)	2/(y, ε)
v_1	2/(ε, u)	**1/(y, u)**
v_2	1/(y, ε)	2/(y, ε)

Fig. 7.1 Machine *CompA*1

	a
i	**a/(ε, v_1)**
u	a/(o, v_2)

Fig. 7.2 Machine *CompA*2

The specification of the given specification *Spec* is shown in Fig. 6.3 and it has the sequence x as a distinguishing sequence, i.e. the outputs at the states 1*a* and 2*a* of *Spec* to x are different output pairs (y, o) and (y, ε). We assume that the implementations *Imp*1 and *Imp*2 of the *A*1 and *A*2 were tested in isolation and found quasi-equivalent to *A*1 and *A*2, respectively.

	1a	2a
x	1a/(*y*,*o*)	2a/(*y*,*ε*)
i	2a/(*y*,*o*)	1a/(*y*,*ε*); 2a/(*y*,*ε*); 2a/(*y*,*o*); 1a/(*yy*,*o*); 2a/(*yy*,*o*)
xi	2a/(*yy*,*oo*)	1a/(*yy*,*ε*); 2a/(*yy*,*ε*); 1a/(*yyy*,*o*); 2a/(*yyy*,*o*); 2a/(*yy*,*o*)

Fig. 8.1. Mutation machine *MM*1 ◊*CompA*2

	1a	2a
x	1a/(*y*,*o*)	2a/(*y*,*ε*)
i	2a/(*y*,*o*); 2a/(*y*,*oo*); 1a/(*y*,*o*); 1a/(*y*,*ε*)	1a/(*yy*,*o*); 1a/(*yy*,*oo*); 2a/(*y*,*ε*); 2a/(*y*,*o*)
xi	1a/(*y*,*oo*); 2a/(*yy*,*oo*); 1a/(*yy*,*o*); 2a/(*yy*,*ooo*); 1a/(*yy*,*oo*);	1a/(*yyy*,*o*); 1a/(*yyy*,*oo*); 2a/(*yy*,*ε*); 2a/(*yy*,*o*)

Fig. 8.2. Mutation machine *CompA*1◊*MM*2

	1a	2a
x	1a/(*y*,*o*)	2a/(*y*,*ε*)
i	2a/(*y*,*o*); 2a/(*y*,*oo*); 1a/(*y*,*o*); 1a/(*y*,*ε*); 1a/(*ε*,*o*);	1a/(*yy*,*o*); 1a/(*y*,*oo*); 1a/(*yy*,*oo*); 1a/(*y*,*o*); 1a/(*y*,*ε*); 2a/(*yy*,*oo*); 2a/(*yy*,*o*); 2a/(*y*,*oo*); 2a/(*y*,*ε*); 2a/(*ε*,*o*); 2a/(*y*,*o*);
xi	1a/(*y*,*oo*); 2a/(*yy*,*oo*); 1a/(*yy*,*o*); 2a/(*yy*,*ooo*); 1a/(*yy*,*oo*);	1a/(*yyy*,*o*); 1a/(*yyy*,*oo*); 1a/(*yy*,*oo*); 1a/(*yy*,*o*); 1a/(*yy*,*ε*); 2a/(*yyy*,*oo*); 2a/(*yyy*,*o*); 2a/(*yy*,*oo*); 2a/(*yy*,*o*); 2a/(*yy*,*ε*); 2a/(*y*,*o*)

Fig. 8.3. Mutation machine *MM* when both implementations can be faulty

In order to test if *Imp*1◊*Imp*2 ≅ *Spec*, we first complete the undefined transitions of *A*1 and *A*2 in all possible ways and we obtain the machines *MM*1 and *MM*2. Then, we derive the mutation machine *MM* ≅ *MM*1◊*MM*2 shown in Fig. 8.3. By direct inspection one can observe that *MM* has no livelocks. We assume that the designers complete the partial specifications *A*1 and *A*2 according to their preferences and obtain the complete deterministic FSMs *CompA*1 and *CompA*2 shown in Figures 7.1 and 7.2, where the added transitions are shown in bold. Moreover, we derive the mutation machines *MM*1◊*CompA*2 and *CompA*1◊*MM*2 shown in Figure 8.1 and 8.2, respectively. All nondeterministic transitions of these mutation machines have to be tested. Afterwards, in **Step 2** of Algorithm 2, we derive the input sequences TS_1 = {*riix*, *ri*(*x*,*i*)*x*} using the incremental methods presented in [EYB02]. Particularly, for the non-deterministic transitions of *MM*1◊*CompA*2, we determine the corresponding transitions in *Spec*. These are transitions, (2a)-*i*/(*yy*, *o*)->(1a) and (2a)- (*x*, *i*)/(*yyy*, *o*)-> (1a). The characterization set *W* = {*x*} of *Spec* does not traverse these transitions, accordingly, according to the so-called Case-1 of [EYB02], we derive the incremental tests *riix*, *ri*(*x*,*i*)*x* for testing these transitions. If the SUT is equivalent to *Spec*, the expected behavior of the SUT to the input sequences of TS_1 is *ri*/(*y*, *o*)*i*/(*yy*,*o*)*x*/(*y*,*o*) and *ri/*(*y*, *o*) (*xi*)/(*yyy*,*o*)*x/*(*y*,*o*). Afterwards, using these sequences, we reduce *CompA*1◊*MM*2 of Fig. 8.2. The reduced machine is that in Fig. 8.2 without underlined transitions. Afterwards, for the untested (i.e. non-deterministic) transitions of *CompA*1◊*MM*2, we determine the corresponding transitions in *Spec*. This is the transition (1a)-(*x*, *i*)/(*yy*, *oo*)->(2a). In order to test this transition, we apply again the so-called Case-1 of [EYB02] that returns the input sequence *r*(*x*,*i*)*x*, i.e., TS_2 = {*r*(*x*,*i*)*x*}. The expected output of a fault-free SUT to *r*(*x*,*i*)*x* is *r*.*x*,*i*/(*yy*, *oo*).*x*/(*y*, ε). Finally, in **Step 3**, using TS_1 and TS_2 and their expected outputs, we reduce the mutation machine *MM*. In this example, TS_1 and TS_2 completely reduce *MM*, i.e., all transitions of *MM* become deterministic and *MM* is equivalent to *Spec* in Fig. 6.3. Thus, we skip **Step 4** and the test suite $TS_1 \cup TS_2$ completely checks if *Imp*1◊ *Imp*2 ≅ *Spec*. The total length of the union of the test suites is 11, while the length of the test suite derived using the W method for the whole specification *Spec* is 18.

6. Conclusion

In this paper, we have proposed two fault models and methods for the derivation of interoperability test suites when the system implementation is given in the form of two deterministic communicating finite state machines. A test suite returned by the first method determines if the implementation is free of livelocks. If the implementation is free of livelocks, the second method returns a test suite that checks if the implementation conforms to the specification.

Acknowledgements

The authors would like to thank Rita Dorofeeva at Tomsk State University for implementing and experimenting with the incremental testing methods. Moreover, the authors from Tomsk State University also acknowledge a partial support from the program “Russian Universities”.

References

[BrZa83] D.Brand and P.Zafiropulo, On communicating finite state machines, J. ACM 30(2), (1983) 323-342.

[BoSu80] G. v. Bochmann, and C. A. Sunshine, “Formal methods in communication protocol design”, *IEEE Trans. on Comm*., Vol 28, 1980, pp 624-631.

[Chow78] T. S. Chow, “Test Design Modeled by Finite-State Machines,” *IEEE Trans. SE*, vol. 4, no.3, 1978, pp. 178-187.

[Elf02] K. El-Fakih, Protocol retesting and diagnostic testing methods, Ph.D. Thesis, University of Ottawa, 2002.

[EYB02] K. El-Fakih, N. Yevtushenko and G.Bochmann, Protocol re-testing methods, Proc. of the IFIP 14th International Conference on Testing of Communicating Systems, 2002, Berlin, Germany, 19-22.

[EPYB03] K. El-Fakih, S. Prokopenko, N.Yevtushenko, G. Bochmann, Fault diagnosis in extended finite state machines. Proc. of the IFIP 15th International Conference on Testing of Communicating Systems. Lecture Notes in Computer Science 2644, pp. 197-210, 2003.

[Fuj91] S. Fujiwara, G. v. Bochmann, F. Khendek, M. Amalou, and A. Ghedamsi, "Test Selection Based on Finite State Models," *IEEE Trans. SE*, vol. 17, no. 6, 1991, pp. 591-603.

[KSK00] S.Kang,, J.Shin, M.Kim, Interoperability Test Suite Derivation for Communication Protocols. Computer Networks, 32 (2000) 347-364.

[KPY99] I. Koufareva, A. Petrenko, N. Yevtushenko, Test generation driven by user-defined fault models, Proceedings of IFIP TC6 12th International Workkshop on Testing of Communicating Systems, Hungary, 1999. – pp. 215-233.

[PYB96] A. Petrenko, N. Yevtushenko, G. v. Bochmann. Fault models for testing in context. FORTE'96.

[PYBD96] A. Petrenko, N. Yevtushenko, G. v. Bochmann, and R. Dssouli, Testing in context: framework and test derivation, Computer communications, Vol. 19, pp. 1236-1249, 1996.

[PYLD93] A. Petrenko, N. Yevtushenko, A. Lebedev, and A. Das, "Nondeterministic State Machines in Protocol Conformance Testing," *Proc. of the IFIP 6th* IWPTS, France, 1993, pp. 363-378.

[SKC02] S.Seol, M.Kim, and S.T.Chanson, Interoperability Test Generation for Communication Protocols based on Multiple Stimuli Principle, Proceedings of the IFIP 14th Inter. Conf. TestCom2002, Berlin, pp.151-169.

[STEY03] N. Spitsyna , V. Trenkaev, K. El-Fakih, and N. Yevtushenko, FSM based interoperability testing-work in progress, presented as work in progress at FORTE 03, Berlin, Germany, Sept. 2003.

[TKS03] Trenkaev V., Kim M., and Seol S. Interoperability Testing Based on a Fault Model for a System of Communicating FSMs // Lecture Notes in Computer Science, Vol. 2644: D.Hogrefe, A.Wiles (Eds.), Testing of Communicating Systems, Proceedings, 2003, pp. 226-241

[VBT01] C. Viho, S.Barbin and L. Tanguy, Towards a formal framework for interoperability testing, Proceedings of the 21st Inter. Conf. FORTE 2001, Korea, pp.51-68.

[West78] C.H. West, An automated technique of communication protocols validation, IEEE Trans. Comm., 26 (1978) 1271-1275.