TEST GENERATION FOR CEFSM COMBINING SPECIFICATION AND FAULT COVERAGE

Sergiy Boroday¹, Alex Petrenko¹, Roland Groz², and Yves-Marie Quemener²
1 – CRIM, Montréal; 2 – France Telecom R&D

Abstract We discuss how specification coverage and fault coverage based test derivation strategies can be combined. In particular, the problem of deriving a test suffix, which raises tester confidence in the configuration of the system, reached after a test derived by a specification coverage criterion, is formalized and solved in the CEFSM setting. The traditional mutant-based test derivation approach is extended with nondeterministic mutants to cover more faults. An experimental tool and case study are reported.

Key words: communicating extended state machines, black box testing, SDL, model based testing, specification coverage, fault models, fault coverage, test derivation.

1. INTRODUCTION

1.1 Specification vs. Fault Coverage

In this section, we contrast two common strategies used for deriving tests from formal specifications in terms of state machines, in particular specification coverage driven approach and fault model/mutation driven approach. We consider here the model of communicating extended FSM (EFSM). There exist a number of tools that support software development activities around specifications based on this model.

Specification coverage: In [GCR96], it is argued that coverage of the extended transitions of an EFSM corresponds well to the preferred coverage implied in the conformance testing standard IS9646. As an example, the commercial tools available on SDL now offer test generation facilities, which were designed based on research for generating conformance tests

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: [10.1007/978-0-387-35497-2_31](https://doi.org/10.1007/978-0-387-35497-2_31)

[EGHJ97], [KJG99]. Such tools may resort to reachability analysis to compute tests that will cover transitions of the EFSM and provide test preambles to reach specific configurations of the EFSM enabling the transitions to be tested. However, they currently do not check the tail state of transitions or the configuration reached after a test. As a result, faults in an implementation (known as Implementation Under Test), viz. faults whereby a transition in the IUT would reach a wrong tail configuration, may easily go undetected. Therefore, the current test generation methods used in tools for EFSM testing have very limited fault detection capabilities.

Fault coverage: Fault models have been used in many testing methods for the classical FSM model [Petr00]. Typical fault models for EFSM can include a wrong tail state and erroneous values of certain variables after a test execution. In mutation-based testing, a faulty implementation is assumed to be derived from a specification, while the difference between them could be explained in terms of the specification's elements or a mutation operator [BDDD91], [YPB94], [ABM98], [FMMD99]. In a more general view, a fault model could be derived from a higher level of testing requirements, than just structural mutations of the specification.

The question arises whether one could merge the two approaches into a test derivation strategy for communicating EFSM to incorporate both concerns, covering interesting portions of the specification and increasing the fault coverage of the resulting tests according to the user-defined fault models. In the next section, we discuss a framework for mutant-based test derivation which embraces both approaches.

1.2 Confirming configuration w.r.t. mutant set

Our idea is to start from an existing test suite as computed by the first approach. Instead of simply adding new tests, we consider that such a test suite exercises well the basic structure of the specification, and that likely residual faults are defined by the user. Then, as in traditional FSM test methods, we derive with a fault-based approach confirming sequences to check the tail configuration (state and data) of each test with respect to the given fault/mutation model. In [PBG99], we investigated such a combined approach for EFSM specifications and now we extend it to cover modular specifications in the form of CEFSM.

Given a (multi-module) specification, a few mutants, and a test sequence, it is required to find a confirming sequence of a given maximal length that distinguishes the specification from mutants when applied to each of the mutants after the test. All the potential implementation faults are modeled by the given mutants themselves that are somehow derived from the specification and their data configurations. Thus, there are two means for

modeling faults, first by introducing some structural changes or mutations into the specification and second, by choosing the initial configuration (configuration reached after the prefix test sequence) of a mutant. Under a competent programmer hypothesis, mutants are usually supposed to differ from the specification only in a marginal way. Therefore, here we assume that the mutants inherit input and output ports of the specification and are defined over the set of inputs of the specification. The problem of building mutants from a given specification in the form of CEFSM is the task of the test expert working from test requirements.

A simple way of building a mutant is to take the CEFSM specification and initialize it into a (suspicious) configuration different from the final configuration reached after the test. An implicit fault model is that an implementation under test reproduces the reference reaction to the test, as it passes the test, but ends up in this erroneous configuration. A confirming sequence has to detect such a residual implementation fault if the fault is an error, in other words, if the mutant is distinguishable from the specification. In this scenario, the test is used only to determine the expected configuration to properly initialize the specification.

In the case of modular specifications, guessing suspicious configurations for each of the modules becomes a daunting task. Moreover, when a mutant structurally differs from the specification, it is not clear how to change the initialization inherited from the specification. We, therefore, assume that the test expert just builds a number of mutants that can pass the given test. This means that the specification and all the mutants are set in the initial configurations. We are required to determine an input sequence that confirms the initial configuration of the specification w.r.t. the initial configurations of mutants taking the given test as a predefined prefix or preamble. Note that this setting does not exclude the above scenario when the test expert just defines a suspicious configuration. In this case, the prefix becomes an empty sequence.

We believe that the process of deriving representative mutants is application-dependant and we do not attempt to give here any generic recipe. What we can suggest here is that to capture more faults with a single mutant the test expert could first mutate the specification according to his/her testing strategy and then choose states and/or variables in a few modules to abstract them. Such abstraction can be used to cover multiple faults instead of only single faults; moreover abstraction may reduce the configuration number and the “size” of configurations. Note that since the process of abstraction induces non-determinism, we allow for such mutants to be non-deterministic even when the specification is perfectly deterministic. The presence of non-deterministic mutants makes our problem statement more general than a classical mutation-based testing problem.

2. CEFSM MODEL

2.1 Extended finite state machine, EFSM

The EFSM model defined in [PBG99] is extended with multiple output signals, as this makes it easier to map languages such as SDL into CEFSM.

Definition 2.1. An *extended* finite state machine (EFSM) M is a pair (S, T) of a finite set of states S and a finite set of transitions T between states from S , such that each transition $t \in T$ is a tuple (s, x, P, op, w, up, s') , where $s, s' \in S$ are the initial and final states of the transition, respectively; $x \in X$ is input, X is a finite set of inputs, and each $x \in X$ is associated with a set of input vectors D_{inp_x} , each component of an input vector corresponds to an input parameter associated with x , the sets of input parameters of different signals are mutually disjoint; $w \in Y^*$ is a sequence of output signals $y_1, y_2, \dots, y_n$, $y_i \in Y$, where Y is a finite set of (elemental) outputs, disjoint from X , each $y \in Y$ is associated with a set of output vectors D_{out_y} , each component of an output vector corresponds to an output parameter associated with y , the sets of output parameters of different signals are mutually disjoint; P , op , and up are functions, defined over input parameters and context variables V , namely:

- $P: D_{inp_x} \times D_V \rightarrow \{\text{True}, \text{False}\}$ is a predicate, where D_V is a set of context vectors $\vec{v}$;
- $op: D_{inp_x} \times D_V \rightarrow D_{out_{y_1}} \times D_{out_{y_2}} \times \dots \times D_{out_{y_n}}$ is an output parameter function that defines values of parameters of each signal in the output word;
- $up: D_{inp_x} \times D_V \rightarrow D_V$ is a context update function.

We normally use $(s \xrightarrow{x, P/op, w, up} s')$ to denote a transition $t \in T$, we will also use the notation $(s \xrightarrow{x, P/y_1(op_1), y_2(op_2), \dots, y_n(op_n), up} s')$. An empty output word will be dropped from a transition to simplify its notation and such a transition is called a *silent* transition.

A *parameterized* signal, input or output, is a pair of the signal itself and a vector of its parameters. It may also be defined as a pair of a signal and a mapping of its parameters into the parameter values. We will use notations for signals in the form of, for example, $y(2, 3)$ meaning the signal y whose two parameters are instantiated to 2 and 3.

Definition 2.2. A context vector $\vec{v} \in D_V$ is called a *context* of M . A *configuration* of M is a pair of state s and context $\vec{v}$.

A context may also be defined as a mapping of variables into variable values.

In the case of empty sets of context variables and parameters, we do not make a difference between configuration and state, in other words, the EFSM model includes the FSM model as a special case.

The EFSM operates as defined in [PBG99]. The machine usually starts from a designated configuration, called the *initial* configuration. A pair of an EFSM and the initial configuration is called an *initialized* EFSM. The *behavior* of an initialized EFSM is the set of all its input-output sequences.

Definition 2.3. An *input-output sequence* of an EFSM (or input-output word) is a sequence of parameterized inputs interleaved by parameterized outputs produced by the EFSM executing a finite sequence of transitions from a certain configuration in response to these inputs.

We target a class of specification EFSMs which are consistent, completely specified, deterministic, and observable [PBG99]. We use the abbreviation REFSM to refer to a machine from this Restricted class of EFSMs. At the same time, in our constructions, we will also use other types of EFSMs, which do not necessarily meet all these requirements.

2.2 Communicating EFSM, CEFSM

Now we define what we mean by a system of communicating EFSMs.

Definition 2.4. A finite set of consistent, completely specified EFSMs along with two disjoint sets of external parameterized input and output signals is called a system of *communicating* EFSMs (CEFMS), if the following conditions are satisfied:

- a) the sets of output signals of any two EFSMs are mutually disjoint;
- b) the external inputs are all inputs of communicating EFSMs that are not outputs of communicating EFSMs;
- c) each external output is an output of an EFSM;
- d) any signal is always associated with the same set of parameters, whenever it appears.

CEFMS functions as follows. Communications are performed via input (FIFO) queues, a single queue is associated with each communicating EFSM (as in SDL). External and internal inputs enter the corresponding queues, while each EFSM consumes signals from its queue. A signal may enter several queues at the same time. A global configuration of the system is defined as a tuple of configurations of component EFSMs and configuration (contents) of their queues. A global configuration of the system with empty queues is called *stable*. When the environment offers an external parameterized input signal to the system, which is in a stable configuration, this input enters queues of all EFSMs to which input alphabets it belongs, and the system goes into an unstable global configuration (a configuration with non-empty queues). In an unstable

configuration of CEFSM, an EFSM (with non-empty queue) takes a signal from its queue and fires an enabled transition. The machine produces a (possibly empty) sequence of outputs, which are consequently put into input queues of corresponding machines, and/or sent to the environment if there are external outputs among these signals. Thus the system moves into a new global configuration, which is either stable or unstable. If a stable configuration is eventually reached the CEFSM can accept another external input, otherwise it falls into livelock.

We make certain assumptions about the environment and behavior of the CEFSM. First of all, we assume that the environment is slow, this means that it submits a next external input to the system only in a stable configuration, when all the queues are empty (this assumption is also known as the synchronization assumption). Test derivation and test execution with a slow testing environment is much simpler compared to an unrestricted one.

Our next assumption concerns only specifications of systems to be tested. We assume that there exists a maximal number of communications that can be triggered by an external input in a specification CEFSM. This implies that:

- the number of external outputs, which may be obtained in response to any external input, never exceeds a certain bound;
- a system always stabilizes in response to external input (it has no livelocks).

Under these assumptions, for a given CEFSM there exists an equivalent EFSM, except for pathologic nondeterministic CEFSMs that allow infinite set of alternative output parameter values for a pair of configuration and input signal. Such CEFSMs have no equivalent EFSMs, because our EFSM model supports only limited nondeterminism (between several transitions), and nondeterminism in output and update function (multifunctions) is not allowed. A system of communicating REFSMs is not necessarily equivalent to an REFSM, because its behavior in terms of external inputs and outputs can be nondeterministic. We rely on the existence of an EFSM equivalent to the given CEFSM in order to formally define input sequences that separate two CEFSM, i.e., that distinguish their external behavior.

2.3 Configuration separating and confirming sequences

We extend the definition of a separating sequence given in [PBG99] to cover multiple outputs.

Definition 2.5. Given two EFSMs M and N , two parameterized signals of M and N are said to be *compatible* if the names of the signals coincide and every common signal parameter has the same value in both parameterized

signals. Two parameterized input-output sequences, $z_1 \dots z_k$ of M and $z'_1 \dots z'_k$ of N , are *compatible* if $k' = k$ and for all $i = 1, \dots, k$, z_i and z'_i are compatible.

Based on this notion, we now define separability of configurations.

Definition 2.6. Given a parameterized input sequence α of EFSMs M and N , configuration c of M and configuration c' of N are *separable* (by α) if every parameterized input-output sequence, produced by (M, c) in response to α , is not compatible with any parameterized input-output sequence that can be produced by (N, c') in response to α . α is said to be a sequence *separating* c from c' .

Configurations of REFSMs with the same sets of inputs, outputs, and their parameters that are not separable are also referred to as *equivalent* configurations. Two REFSMs are *equivalent* if their initial configurations are equivalent. Note that in this study, we assume that a given REFSM may have different configurations that are equivalent.

Definition 2.7. A parameterized input sequence α *confirms* a configuration c w.r.t. a set of configurations C (possibly of different machines) if it separates c from every configuration c' of the set C separable from c . Such a sequence is called a *complete configuration confirming sequence*.

In the above definition, the configuration c is usually assumed to be the one of the specification EFSM M and therefore called the *expected* configuration, while the configurations in the set C are referred to as *suspicious* configurations representing configurations of some faulty implementation machines to which an IUT may erroneously arrive having successfully passed some test case. Following a best-effort approach, as advocated in [PBG99], instead of aiming at finding complete configuration confirming sequence, we target *configuration-confirming sequence*, as a sequence of length l or lesser that confirms c w.r.t. a subset of maximal cardinality of C among all possible parameterized input sequences of that length. Here l is a predefined integer that indicates the acceptable length of the sequence. Whenever complete configuration confirming sequence of length l exists, any configuration-confirming sequence is complete and vice versa. The following proposition gives us sufficient conditions for the existence of configuration-confirming sequence.

Proposition 2.1. Given the expected configuration c and the set of suspicious configurations C , a configuration-confirming sequence exists if the set C , domains of input parameters, or domains of output parameters are finite.

We will restrict ourselves to finite suspicious configuration sets. So, in our problem setting, configuration-confirming sequence always exists. Derivation of a suspicious set (with the corresponding machines) is discussed in [PBG99]. These machines will be referred to as *mutants*. Unlike

the traditional approach, our mutants are allowed to be nondeterministic to model more faults.

Configuration-confirming sequence for CEFSM is defined based on the corresponding equivalent EFSM.

3. TEST DERIVATION

Given an REFSM M , an expected configuration c_0 of M , a set C of k of suspicious configurations of k possibly different mutant (C)EFSMs, c_1 of N_1 , c_2 of N_2 , ..., c_k of N_k , and the maximal length l of sequences, we are required to determine a configuration-confirming sequence. The mutant machines are assumed to be observable and completely defined. The mutants share the same input signal set with the specification. However, mutant's signals could be abstracted from certain parameters of the specification. In this section, we discuss three approaches for deriving confirming sequences for EFSM and CEFSM.

3.1 Distinguishing machine

For the case of single output transitions a solution based on the reachability analysis of the distinguishing machines is proposed in [PBG99]. To determine a configuration confirming sequence, the reachability analysis of this machine should be performed with dedicated tools, e.g., simulators and model checkers. However, the transfer of this method onto real world EFSM languages, which do not follow our EFSM syntax and semantics, may be problematic and faces complexity issues. Therefore, in next section, we consider an alternative method, which reduces derivation of the confirming sequence to reachability analysis of a CEFSM defined from given specification and mutants.

3.2 Distinguishing system for EFSM

We now reduce the testing problem to the problem of analyzing the behavior of a CEFSM. The idea is to build a system of $k + 2$ communicating machines, called a *distinguishing system*. One machine is the original machine modified with the *eot* signal ("end of transition" signal). The *eot* signal is inserted at the end of the output word of each transition to facilitate the comparison of different machines' behavior caused by the same input. There are k machines of the system each of which represents an EFSM initialized in a corresponding suspicious configuration and signals a discrepancy between its behavior and that of the original machine by issuing

a designated signal to the environment. The *Monitor* assists communications between machines. We formally define a distinguishing system as a CEFSM, that includes the following EFSMs: $M_0, M_1, M_2, \dots, M_k, \text{Monitor}$. The set of external inputs of the system is defined as follows. For each input x of M there is an external input x^{ext} of the system with the same sets of input vectors and parameters, as the input x of the M . The system has elemental outputs $\text{fail}^1, \dots, \text{fail}^k$.

The machine M_0 is a slightly modified machine M . It has the same state and input sets as the M . The set of outputs includes all the outputs of the M , as well as a designated output eot . All the parameters and their vectors are defined as in M . Transitions of M_0 are defined as follows. For every transition $(s \xrightarrow{x, P_M / \text{op}_M, y_1 y_2 \dots y_h, \text{up}_M} s')$ of M , there is a transition in M_0 $(s \xrightarrow{x, P_M / \text{op}_M, y_1 y_2 \dots y_h, \text{eot}, \text{up}_M} s')$.

In other words, the only difference between M and M_0 is the eot signal appended to the output word of each transition. The initial configuration of the machine is c_0 .

Next, we define the machines $M_1, \dots, M_k$, corresponding to the mutants $N_1, \dots, N_k$ initialized in suspicious configurations $c_1, \dots, c_k$, respectively. The machines $M_1, \dots, M_k$ are called the inverted mutants, as outputs of mutants become inputs in these machines. The transitions of M_i are also defined in such a way that any discrepancy in parameterized outputs produced by the machine (M_0, c_0) and the (N_i, c_i) in response to inputs is detected and signaled to the environment (the expected configuration c_0 is separated from the c_i). Speaking more formally, the inverted mutants are defined as follows.

Since, by definition, the sets of output signals of two different machines of any CEFSM must be disjoint, and signals are not allowed to have different sets of parameters, we rename signals. For every i , $1 \leq i \leq k$, the notation of signals of the machine M_i is decorated with the superscript i so that it becomes clear to which machine this signal belongs. The signal eot^i along with the sets of parameterized inputs and outputs of N_i (renamed as we described above) constitute the set of inputs of M_i . The output signal fail^i is the only elemental output. For each variable of the machine N_i there is a variable of M_i with the same name and domain and for each input parameter vector inp_x of an input signal x of the machine the auxiliary context variables with the corresponding domains are defined. The projection of the context vector on these variables is denoted $\bar{u}_x$. There is a new state s_{fail} with the transitions $(s_{\text{fail}} \xrightarrow{\text{eot}^i} s_{\text{fail}})$, $(s_{\text{fail}} \xrightarrow{x^i} s_{\text{fail}})$, $(s_{\text{fail}} \xrightarrow{y^i} s_{\text{fail}})$, for all $x^i \in X^i$ and $y^i \in Y^i$.

For each transition $t = (s \xrightarrow{x^i, P_M / \text{op}_M, y_1 y_2 \dots y_h, \text{eot}^i, \text{up}_M} s')$ of the N_i the machine M_i has the transition $(s \xrightarrow{x^i, P_M / (\bar{u}_x := \text{inp}_x) \rightarrow q_{t,1}})$, where inp_x is the input parameter vector of x^i . The state s also has the transitions $(s \xrightarrow{y^i / \text{fail}^i} s_{\text{fail}})$ for each $y^i \in Y^i$, and $(s \xrightarrow{\text{eot}^i / \text{fail}^i} s_{\text{fail}})$.

States $q_{t,j}$, $1 \leq j \leq h$, are new (compared to the states of N_i) and unique. They have the transitions $(q_{t,j} \xrightarrow{y_j^i, \text{inp}_{y_j} \sim \text{op}_j(\bar{v}, \bar{u}_x)} q_{t,j+1})$, $(q_{t,j} \xrightarrow{y_j^i, \neg(\text{inp}_{y_j} \sim \text{op}_j(\bar{v}, \bar{u}_x))} \text{fail}^i \rightarrow s_{\text{fail}})$, where $\text{inp}_{y_j} \sim \text{op}_j(\bar{v}, \bar{u}_x)$ is a predicate that holds True iff the elements of vectors inp_{y_j} and $\text{op}_j(\bar{v}, \bar{u}_x)$ that correspond to same parameters are equal. The outgoing transitions of the states $q_{t,j}$ are $(q_{t,j} \xrightarrow{y^i} \text{fail}^i \rightarrow s_{\text{fail}})$ for each $y^i \neq y_j^i$, $y^i \in Y^i$; and $(q_{t,j} \xrightarrow{eot^i} \text{fail}^i \rightarrow s_{\text{fail}})$. In addition, for all $x \in X$, there are transitions $(q_{t,j} \xrightarrow{x^i} \text{fail}^i \rightarrow s_{\text{fail}})$.

Finally, we define the outgoing transitions of the state $q_{t,h+1}$: $(q_{t,h+1} \xrightarrow{eot^i} \text{up}_M \rightarrow s')$, $(q_{t,h+1} \xrightarrow{y} \text{fail}^i \rightarrow s_{\text{fail}})$ for all $y \in Y$, and $(q_{t,h+1} \xrightarrow{x^i} \text{fail}^i \rightarrow s_{\text{fail}})$ for all $x \in X$.

The initial configuration of M_i is defined by c_i and arbitrary values of the auxiliary context variables.

The machine *Monitor* has the only state s , input signals X , Y , eot , the output signals and their parameters are inputs and parameters of machines $M_1, \dots, M_k$. The *Monitor* supports communications between the components with the transitions $(s \xrightarrow{x^{\text{ext}}/\text{op}_x, xx^1 \dots x^k} s)$, $(s \xrightarrow{y/\text{op}_y, y^1 \dots y^k} s)$, $(s \xrightarrow{eot/\text{eot}^1 \dots \text{eot}^k} s)$, where op_x is a function that takes a parameter vector of x^{ext} and returns the parameter vectors for signals $x, x^1, \dots, x^k$, preserving values of parameters. Similarly, the function op_y deals with the output parameter vector of y .

Proposition 3.1. An input sequence separates the expected configuration c_0 from the initial configuration c_i of the machine N_i iff the distinguishing system produces the external signal fail^i when the environment applies the input sequence.

Consider the distinguishing system that consumes a certain (parameterized) input sequence. As it is clear from the former proposition, the number of outputs fail^i , produced by the distinguishing system in response to the last external input of the sequence is equal to the number of the configurations of C that this sequence separates from c_0 . This leads us to the following corollary.

Corollary 3.2. An input sequence of length l or less is a configuration-confirming sequence of the expected configuration c_0 with respect to the set of suspicious configurations C iff the number of the *fail* signals produced by the distinguishing system in response to the sequence is maximal.

Thus, determining a configuration-confirming sequence amounts to finding a sequence that maximizes the number of *fail* in the distinguishing system.

The distinguishing system may be implemented in an EFSM-based language, such as SDL. A commercial model checker, such as ObjectGeode, can be applied to the distinguishing system to find confirming sequences.

The approach based on distinguishing system has its own limitations. First, mutant machines have to be observable, but transforming a mutant specification (in the form of SDL process) into an equivalent observable EFSM form, (as well as augmenting the specification with the *eot* signal), may be a challenging task. Second, the approach does not extend to treat modular specifications, CEFSM. These limitations are absent in the approach proposed in the next sections.

3.3 Distinguishing system for CEFSM

In order to apply the concept of the distinguishing system to the case of modular specifications, we could represent each mutant CEFSM in a distinguishing system by an EFSM such that has the same external behavior and produces the only output *fail* as an inverted mutant. The length of confirming sequence should never exceed the bound l , therefore, it is sufficient to partially unfold each mutant CEFSM by considering input sequences of the length l . This idea yields the following method:

1. Restrict the set of external parameterized inputs to a finite set.
2. For each mutant CEFSM, build a deterministic (mutant) LTS which represents the mutant's observable external behavior on all the input sequences of length l obtained from the above set of parameterized inputs.
3. For each mutant LTS, construct an inverted mutant machine. Build the distinguishing system as explained in Section 3.2.

We first build the reachability graph for a mutant CEFSM for the restricted input set, and remove in the resulting LTS internal actions, minimize, and determinize it. The obtained mutant LTS contains all the external traces of the corresponding mutant CEFSM that are produced by input sequences of length l from the restricted input set.

The procedure of inverting a mutant LTS is similar to that for the EFSM case, though it slightly differs because of

- a) a transition of LTS is labeled with a single signal while an (E)FSM transition contains a sequence of an input and output signals;
- b) LTS may have a state with outgoing transitions labeled with input and output;
- c) absence of the *eot* signal.

Given a deterministic mutant LTS, the *inverted mutant machine* N^i is an EFSM with the only elemental output, *fail* ^{l} . The input signals of the machine are all external input and output signals of the mutant CEFSM. For each transition $(s \rightarrow x(p) \rightarrow t)$ of the mutant LTS, where $x(p)$ is a parameterized signal, the inverted mutant machine has a transition $(s \rightarrow x^i, P \rightarrow t)$, where x^i is the signal with the same name as $x(p)$, decorated with i , and P is a predicate,

which delivers True only when the parameterized signal is compatible with $x^i(p)$. In each state of the inverted mutant machine, transitions, which make the machine consistent and completely specified for all the signals of the mutant, produce a *fail* output and lead into the state s_{fail} , are defined.

In the distinguishing system, the specification CEFSM is used as it is, without introducing the *eot* signal. Accordingly, the monitor is simplified by removing this signal. The monitor ignores all the input signals x^{ext} that are not in the restricted input set. With these exceptions, the distinguishing system is as defined in the previous section. As in the previous section, the configuration-confirming sequence (for CEFSMs with the restricted input set) is the input sequence, which delivers the maximal number of *fails*; except the special case, when all the *fail* signal are produced on transitions, which are labeled with external inputs of the specification. In this case that input signal does not contribute to the separating sequence. This is the cost we pay for escaping from the *eot* signal.

Next, we describe how the distinguishing system can be implemented in the ObjectGeod environment.

3.4 Implementation with observers

To implement the above method in ObjectGeode we use so-called event observers, which are deterministic extended automata that are directly driven by events of a simulated system. Observers are defined in an SDL like language, GOAL. Observers are used to specify test purposes in the Object Geode TestComposer.

To automate the process of observer generation, we use the Aldebaran tool [CADP] along with a converter of an Object Geode graph into the Aldebaran LTS format GeodeParse. In addition, we have developed a number of shell and awk scripts that call the above programs in a proper order, and build observers. An auxiliary program [Wang01] for variable abstraction was also used in observer construction.

The environment we built allows us to deal with explicitly input complete SDL specifications, which are free from time constraints, timers could, e.g., be replaced by input signals. Additionally, specifications (but not mutants) are required to behave deterministically.

3.5 Extensions and enhancements

3.5.1 Handling multi-mutants

The ObjectGeode simulator can simulate a given SDL specification with several (mutant) observers. The developed script handles just a single mutant observer, but it does not take much time to extend it to process several observers and to eventually determine an input sequence that separates a maximal number of mutants. Since different mutants represent various faults, a goal function, which takes into account user-defined weights of mutants or mutant groups, could be defined. Derivation of a confirming sequence, which maximizes such a goal function, is possible with the Object Geode simulator.

3.5.2 Handling non-separable mutants

The question arises what can be done in the case when for a given specification and a mutant no conforming sequence exists. In terms of the FSM model, this means that the specification machine is a reduction (trace included into) of the mutant machine, hence no separating sequence. We call such a mutant *non-separable*. Note that a non-separable deterministic mutant is observationally equivalent to the specification, thus no useful test can be suggested in this case. However, a nondeterministic non-separable mutant could be used to choose a most “powerful” candidate sequence to be applied as a “quasi-confirming” sequence in this case.

The idea comes from interpreting a nondeterministic mutant as a compact joint representation of a number of potential implementation faults, see, e.g., [PeYe92]. Different faults create thus various executions of the mutant. Therefore, given a candidate input sequence, we may use the number of different executions or output sequences produced by the mutant in response to it to characterize the number of faults “covered” by the sequence. Note that in the case, when a mutant has a chaos state, capable of producing any output sequence in response to any input sequence, a transfer sequence to this state could be most powerful.

The introduced measure can also be used to handle several mutants, by appropriately modifying the goal function to incorporate both, separable and non-separable, mutants.

4. EXPERIMENTS

To validate the approach, experiments on the VTT2 protocol, which is proposed by FranceTelecom R&D, have been performed.

4.1 The VTT2 example

We use a receiving block of the VTT2 protocol, which is a simplified version of a video transmission protocol. The protocol consists of three processes, each of which has two to four states, three to six variables of types boolean, natural, and a set of naturals. Each process has three inputs (including a timer which we replace with an external signal) and two or three outputs. Some signals are parameterized.

4.2 Mutations and tests

We have defined two mutants from our SDL specification of the receiving block of the VTT2 protocol. For those two mutants, we use the same fault model: we suppress one assignment from the original specification. In the first case, it is the assignment of a boolean variable, in the second one, it is a natural variable. In addition, for each mutant, we have abstracted a boolean variable to cover more options for erroneous behavior. This induces nondeterminism in the mutants.

For each of those mutants, we found a test case which does not distinguish between a correct implementation and the mutant. For the first one, we used a test case of length 7, and for the second one of length 18. It is required, for each pair of the mutant and test case, to find a confirming sequence, which being applied to the mutant right after the test case delivers only output sequences different from output sequences of the specification.

4.3 Results

The simulation of the distinguishing systems, built for $l = 3$ (the maximal acceptable length of confirming sequences) is performed. As a result, complete confirming sequences for both test cases are found.

Table 1 provides the characterization of the experiments in terms of states (configurations) and transitions in the constructed objects for each of the two test cases and mutants. As expected, an LTS that describes the observable behavior of a mutant has much fewer states than the unfolded mutant itself. In our example, the number of states reduces by about 30

times. The final number of the configurations traversed during the search of confirming sequences is negligible in our example.

Table 1. Characterization of the experiments in terms of states and transitions

	Test case 1		Test case 2	
	States	Transitions	States	Transitions
OG graph of the mutant	3376	6402	3357	8297
Aldebaran LTS (sliced LTS)	6249	9275	5067	10007
Minimized τ -free LTS	238	1380	173	529
Determinized LTS	226	1308	158	514
OG graph of the distinguishing system (specification + observer)	66	76	59	80

5. CONCLUSION

We addressed the following testing scenario. Assume that some test generation tool has produced from a formal specification in the form of CEFSM/SDL a test case based on some specification coverage concerns. The efforts have to be made to improve the fault detection capability of the test case at a reasonable price by extending the test. The novelty and thus, the complexity of this scenario lie in the fact that fault/mutation coverage-driven testing approaches are traditionally considered incompatible or even orthogonal to specification coverage-driven approaches.

We present a solution to this problem based on which we conclude that fault/mutation and specification coverage approaches for communicating EFSMs are in fact compatible and can seamlessly be integrated into a common test derivation environment.

The specific contributions of our work are as follows.

We cast the problem of extending a given test in a formal setting, using the model of untimed CEFSM. The model is extended with multi-outputs such that a machine produces a sequence of elemental output signals in response to a single input signal. The idea behind this definition is to impose as few restrictions on the system as possible, e.g., multicast is allowed. This reduces the gap between the SDL and EFSM models. The operational semantics is also given based on a slow environment assumption. This implies that confirming sequences are to be applied to a system when it is in a stable state (with empty queues). The notion of configuration-confirming sequence for CEFSM is introduced.

The notion of the distinguishing system is introduced based on the ideas presented in [PBG99]. The distinguishing system allows us to determine a confirming sequence for a specification in the form of a single EFSM. We

also extend the distinguishing system to cover CEFSM specification by replacing each mutant CEFSM with a single EFSM that models the behavior of the mutant CEFSM within a restricted scope. We rely on a model checker to eventually determine a most powerful confirming sequence. Thus, we follow a trend in the area to use a model checker engine to derive tests, see e.g., [WaLi93], [CaSc96], [HLS96], [ABM98], [KJG99], [JM99]. At the same time, the novelty of our approach is that nondeterministic mutants are used to reduce the state space needed to model faults/mutations, while covering more potential faults.

Using the ObjectGeode environment, we demonstrated that the proposed testing methodology is compatible and complementary to the existing test derivation facilities of the industrial SDL tools.

ACKNOWLEDGMENTS

This work was partly supported by France Telecom (research contract 001B341). The work of Yves-Marie Quemener was partly supported by the European Commission (FET project ADVANCE, contract No IST-1999-29082). We would like to thank D. Vincent for his contribution to an experiment on GOAL observers; Alain Kerbrat for the GeodeParse tool, and discussions; Telelogic for the ObjectGeode license given to CRIM, and the Vasy team of INRIA for the CADP tool license.

REFERENCES

- [ABM98] Ammann P., Black P., and Majurski W. Using Model Checking to Generate Tests from Specifications. Proceedings of 2nd IEEE International Conference on Formal Engineering Methods (ICFEM'98); December 1998; Brisbane, Australia. IEEE Computer Society.
- [BDDD91] v. Bochmann G., Das, A. Dssouli R., Dubuc M., Ghedamsi A., Luo G. Fault Model in Testing. Proceedings of the IFIP IV Workshop on Protocol Test Systems; October 1991; Leidschendam, The Netherlands.
- [CADP] CADP (Caesar/Aldebaran Development Package). <http://www.inria.fr/pes/fr/vasy/cadp>
- [CaSc96] Callahan J. and Schneider F. Specification-Based Testing using Model Checking. Proceedings of the 2nd SPIN Workshop; August 1996; Rutgers, NJ.
- [JM99] Jeron T. and Morel P. Test Generation Derived from Model-Checking. In Proceedings of the 11th Conference on Computer-Aided Verification (CAV'99); July 1999; Trento, Italy. LNCS 1633.
- [FMMD99] Fabbri S.C.P.F., Maldonado J.C, Masiero P.C., and Delamaro M.E. Proteum/FSM: A Tool to Support Finite State Machine Validation Based on Mutation Testing. Proceedings of the 19th International Conference of the Chilean Computer Science Society; November 1999; Talca, Chile.

- [EGHJ97] Ek A., Grabowski J., Hogrefe D., Jerome R., Koch B., Schmitt M. Towards the industrial use of validation techniques and automatic test generation methods for SDL specifications. Proceedings of the 8th SDL Forum; September 1997; Evry, France. Elsevier.
- [GCR96] Groz R., Charles O., and Renévo J., Relating Conformance Test Coverage to Formal Specifications. Proceedings of the IFIP TC6/WG6.1 Joint International Conference on Formal Description Techniques (IX) for Distributed Systems and Communication Protocols, and Protocol Specification, Testing, and Verification (XVI); October 1996; Kaiserslautern, Germany. Chapman&Hall.
- [HLS96] Huang S., Lee D., Staskauskas M., Validation-based test sequence generation for networks of extended finite state machines. Proceedings of the IFIP TC6/WG6.1 Joint International Conference on Formal Description Techniques (IX) for Distributed Systems and Communication Protocols, and Protocol Specification, Testing, and Verification (XVI); October 1996; Kaiserslautern, Germany. Chapman&Hall.
- [KJG99] Kerbrat A., Jéron T., Groz R. Automated test generation from SDL specifications. Proceedings of the 9th SDL Forum; June 1999; Montreal, Canada. Elsevier.
- [Petr00] Petrenko A. Fault Model-Driven Test Derivation from Finite State Models: Annotated Bibliography, LNCS 2067, Proceedings of the Summer School MOVEP'2000, *Modeling and Verification of Parallel Processes*; June 2000; Nantes, France.
- [PBG99] Petrenko A., Boroday S., and Groz R. Confirming configurations in EFSM. Proceedings of the IFIP Joint International Conference on Formal Description Techniques for Distributed Systems (FORTE XII) and Communication Protocols, and Protocol Specification, Testing, and Verification (PSTV XIX); October 1999, China. Kluwer.
- [PeYe92] Petrenko A. , Yevtushenko N. Test Suite Generation for a FSM with a Given Type of Implementation Errors. Proceedings of the IFIP 12th International Symposium on Protocol Specification, Testing, and Verification; 1992; USA. North-Holland.
- [WaLi93] Wang C.-J., and Liu M. T., Generating test cases for EFSM with given fault model. Proceedings of the IEEE INFOCOM'93; 1993; San Francisco, USA.
- [Wang01] Wang X. Abstraction of Variables in SDL, Master project in progress, McGill University, 2001.
- [YPB94] Yao M. , Petrenko A., and v. Bochmann G. A Structural Analysis Approach to the Evaluation of Fault Coverage for Protocol Conformance Testing, Proceedings of the 7th IFIP WG 6.1 International Conference on Formal Description Techniques (FORTE'94) October 1994; Berne, Switzerland. Elsevier, 1995.