2011 Fourth International Conference on Software Testing, Verification and Validation Workshops

MOST: a Multi-Objective Search-Based Testing from EFSM

Thaise Yano, Eliane Martins
Institute of Computing
State University of Campinas, UNICAMP
Campinas, SP, Brazil
tyano,eliane@ic.unicamp.br

Fabiano L. de Sousa
Space Mechanics and Control Division
National Institute for Space Research, INPE
São José dos Campos, SP, Brazil
fabiano@dem.inpe.br

***Abstract*—This paper introduces a multi-objective evolutionary approach to test case generation from extended finite state machines (EFSM), named MOST. Testing from an (E)FSM generally involves executing various transition paths, until a given coverage criterion (e.g. cover all transitions) is met. As traditional test generation methods from FSM only consider the control aspects, they can produce many infeasible paths when applied to EFSMs, due to conflicts in guard conditions along a path. In order to avoid the infeasible path generation, we propose an approach that obtains feasible paths dynamically, instead of performing static reachability analysis as usual for FSM-based methods. Previous works have treated EFSM test case generation as a mono-objective optimization problem. Our approach takes two objectives into account that are the coverage criterion and the solution length. In this way, it is not necessary to establish in advance the test case size as earlier approaches. MOST constructs a Pareto set approximation, i.e., a group of optimal solutions, which allows the test team to select the solutions that represent a good trade-off between both objectives. The paper shows empirical studies to illustrate the benefits of the approach and comparing the results with the ones obtained in a related work.**

***Keywords*-model-based testing; feasible path; search-based testing; EFSM**

I. Introduction

Test generation is a time-consuming activity, and is still predominantly manual. Model-based testing (MBT) is aimed at the automatic test case generation from system behavior models. State models are commonly used to represent system behavior and have been used for test case generation for a long time [1]. In particular, extended finite state machines (EFSM) allows the representation of control as well as data aspects of a system behavior, and can be used to represent a large variety of systems. Test case generation from EFSM, however, is still challenging. Several researchers have considered the control and data flow separately [2], [3], [4]. For the data part, the test selection is based on data-flow criteria developed for structural testing, since the notation used for the description of predicates and actions of an EFSM is similar to high-level programming languages. To test the control flow, the traditional methods for test case generation from finite state machine (FSM) have been applied. The basic idea of these methods is to systematically traverse the model based on a coverage criterion (e.g. all states or all transitions) in this way generating transition paths. In other words, to generate these paths, it is common to use reachability analysis that performs breadth-first or depth-first search of the model. One limitation is that the reachability graph can be infinite, and it is not easy to determine when it is safe to stop the search. Besides, generated paths can be infeasible for an EFSM because of conflicts in conditions in the path.

Instead of exhaustive enumeration of the state space, search-based software testing (SBST) uses meta-heuristics to find test data. The input domain is usually too large for exhaustive testing to be practical. Moreover determining the test data is an undecidable problem and, then, SBST is one way to deal with this problem. Early SBST approaches focused mainly on test data generation for code-based testing [5]. Recently, there are initiatives for model-based testing, in special, from EFSM [6], [7], [8]. A crucial difference in this context is the number of design variables, i.e., elements to be optimized. In code-based testing, the design variables are defined according to the arguments of the program under test, while in MBT it is necessary to generate a test sequence. Thus, the length of a test sequence is also of concern to test case generation.

In this paper we propose the MOST approach, a Multi-Objective Search-based Testing from EFSM. To cope with the problem of infeasible paths generation, we use model execution to obtain the transition paths, instead of reachability analysis. Executable modeling for requirements specification is used to produce a prototype of a system in its initial development phase. One of the benefits of using such models is that system validation can be accomplished earlier in the development cycle. MOST uses an executable version of an EFSM model to evolve the solutions obtained with the use of a meta-heuristic algorithm. The goal is to cover a given test purpose, which, in our case, represents one target transition. By instrumenting the executable model, it is possible to observe the executed transition path for a given input sequence. The input sequence consists of the input interactions and the data to trigger a transition path. In case the observed transition path does not satisfy the test purpose, the input interactions and the data are modified and the process is iterated.

978-0-7695-4345-1/11 $26.00 © 2011 IEEE
DOI 10.1109/ICSTW.2011.37

MOST uses a multi-objective optimization approach, which in particular searches a balance between the minimum length of the input sequence and the test purpose coverage. There are many situations for which the cost of executing a test case is high, in special, when using hardware-in-the-loop or when accessing remote infrastructure (e.g., database). Therefore, our goal is not only to satisfy a given test criterion, but also to allow users to select shorter test cases, when they intend to reduce execution cost.

To guide the search algorithm, we propose the use of model slicing in a previous work [8], while in this paper we rely solely on dependence analysis. As we are only interested in identifying the parts of the EFSM that affect the test purpose, the dependence analysis is enough for this intent and we do not need to perform all steps to obtain the model slice. The objective functions were modified to take into account this change and also to reward better a solution that achieves the test purpose. The dependence analysis is used to deal with the loss of information problem [9], due to the lack of controllability of the internal variables (or context variables) of the EFSM by the search algorithm. An experimental analysis is presented to evaluate the efficiency of MOST and the results were compared to test cases obtained by Kalaji et al. [7]. The main contributions of this paper are as follows:

- The paper introduces the MOST approach, which integrates the control as well as the data aspects of an EFSM for test case generation. Control and data aspects are treated in one step, that is, MOST obtains not only the transition path that constitutes a test case, but also the data that triggers this path. In general, the data is random and obtained after the path generation.
- The use of dependence analysis to guide the search for a solution.
- An executable model is used for the test case generation. A first advantage of this approach is that users can simply execute the model in order to validate it before test case generation. Secondly, since the model is an abstraction of the real implementation, it is more attractive for dynamic test case generation than the system implementation. A third point is the possibility to generate test cases even when no source code is available, such as third-party components or services. Finally, the use of an executable model is also aimed at coping with the infeasible path generation problem: only paths triggered during model execution are considered as candidate solutions.
- The multi-objective approach allows the automatic adjustment of solution length to cover the test purpose. In some approaches [9], [6], [7], this length must be given in advance by the user, and sometimes the algorithm is not able to find a solution for the given length.
- The paper also presents the results from an empirical study using, as subjects, the models used by Kalaji et al. [7]. The results show that MOST is capable of producing test cases for the same models as in the existing approach. The solutions obtained in both approaches are also compared, and a discussion of this comparison is presented.

The paper is organized as follows. Section II presents the EFSM model, the dependence analysis definitions and also some related works. Section III describes the test case generation approach, MOST. Section IV describes the experiments. Section V remarks some conclusions and future works.

II. Background

A. The model

The system behavior is represented by an EFSM, defined as a tuple (S, s_0, I, O, T, V, P) in which [10]: S is a finite set of states; $s_0 \in S$ is the initial state; I is a set of input events; O is a set of output events; T is a finite set of transitions; V is a set of variables; and P is a set of input parameters. Each transition $t \in T$ is given by a source state $source(t) \in S$, a target state $target(t) \in S$ and a label of the form $i(t)[g(t)]/a(t)$ where: $i(t) \in I$, $g(t)$ is a logical expression called guard and $a(t)$ is the action executed when the transition is activated. Input events can contain one or more parameters belonging to P. The parts g and a of the label are optional. A transition is triggered when the corresponding input event occurs and the guard associated with the transition is satisfied. The guard g can contain logical and comparative operators. When the transition is triggered, the corresponding action a is executed, which may change values of variables and parameters in assignment statements, or also produce output events. It is assumed that the machine remains in its current state upon reception of an unexpected input (i.e., inputs not specified in a given state). In the presence of such input, the machine generates a null output as response. In order to illustrate the concepts presented, we use as example the EFSM M_1 of a simple ATM (Automated Teller Machine) [7] (Figure 1).

B. The dependence information

Many approaches for program dependence analysis are based on control-flow graphs that satisfy the unique endpoint requirement, which is not applicable to EFSM in general, as they can have multiple exit nodes or no exit nodes at all. We select the dependence analysis for state models defined by Androutsopoulos et al. [10], [11], since they encompass the main results of previous works and also cope with the non-termination possibility in EFSM. The main concepts used in this research are presented in the sequel.

Androutsopoulos et al. define the dependence analysis according to the type of path. A path is a sequence of nodes such that for every consecutive pair of nodes (n_j, n_{j+1}) in the path there is an edge from n_j to n_{j+1}. A transition

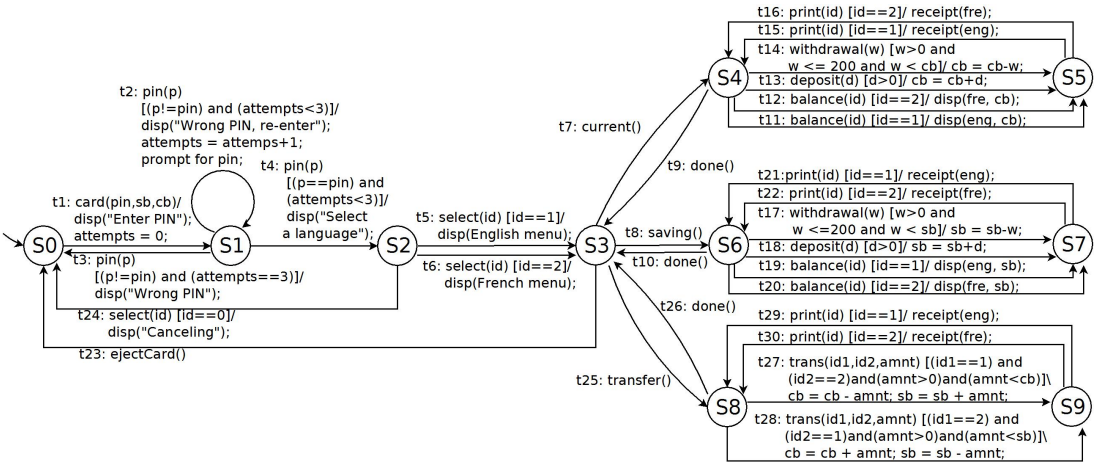

Figure 1. EFSM for an ATM system [7]

$t_j \in T$ is in a path if (n_j, n_{j+1}) is in the path and $n_j = source(t_j)$ and $n_{j+1} = target(t_j)$. A path may be infinite, since it is possible to have infinite loops.

A generic notion of control dependence between transitions is given in terms of a function $PATHs$.

Definition 1. *A transition t_j is **control dependent** on a transition t_i ($t_i \xrightarrow{CD} t_j$) iff t_i has at least one sibling t_k such that: i. for all paths $\pi \in PATHs(target(t_i))$, the $source(t_j)$ belongs to π; ii. there exists a path $\pi \in PATHs(source(t_k))$ such that the $source(t_j)$ does not belong to π. Two transitions t_i and t_k are siblings if $source(t_i) = source(t_k)$.*

The control dependence called NTSCD (Non-Termination Sensitive Control Dependence) is defined considering $PATHs$ as the maximal path function.

Definition 2. *A **maximal path** π is any path that terminates in a final transition, or is infinite. A final transition is one whose target is an exit state that has no outgoing transitions.*

The data dependence (DD) relates transitions according to definitions and uses of variables:

Definition 3. *A transition t_j is **data dependent** on a transition t_i with respect to a variable v ($t_i \xrightarrow{DD} t_j$) if: i. $v \in D(t_i)$, where $D(t_i)$ is a set of variables defined by transition t_i, i.e. variables defined by actions and by the event of t_i; ii. $v \in U(t_j)$, where $U(t_j)$ is a set of variables used in a condition and actions of transition t_j; iii. there exists a path in an EFSM from the $source(t_i)$ to the $target(t_j)$ whereby v is not modified by any of the intermediate transitions.*

In our approach, we define two sets of transitions using the information obtained from the dependence analysis: $T_{affecting}$ and $T_{critical}$. For each transition, we can identify its affecting transitions, $T_{affecting}$, based on the dependence graph (DG) that shows the control and/or data interdependences among the transitions.

Definition 4. *For a given transition t, $T_{affecting}(t)$ contains t and the transitions obtained by a backward traversal of DG starting at t.*

$T_{affecting}(t)$ contains all transitions upon which t is directly or indirectly control and/or data dependent. For the transition t_{13} in the ATM model, $T_{affecting}(t_{13}) = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_{10}, t_{13}, t_{14}, t_{17}, t_{18}, t_{23}, t_{24}, t_{25}, t_{26}, t_{27}, t_{28}\}$ considering the dependence types DD and NTSCD in the DG. The test purpose t is always included in $T_{affecting}$. In our approach, a test path (path starting at the initial state) to reach t needs to contain transitions in $T_{affecting}(t)$. However, not all transitions in $T_{affecting}(t)$ must be part of the test path. For example, a test path that reaches t_{13} can be: $t_1 t_4 t_6 t_7 t_9 t_7 t_{13}$.

Also important to guide our test case generation is to determine the notion for EFSM that corresponds to critical branching nodes in structural testing [12]. This is a branching node with an exit that, if taken, the test path misses the target. In order to extend this concept for an EFSM transition, we use the definitions 1 and 3, according to which a transition t may be dependent on a transition t_a, which can have a sibling t_c, and there exists a path $\pi \in PATHs(source(t_c))$ such that the $source(t)$ does not belong to π. We define a critical transition as:

Definition 5. *A transition t_c is **critical** with respect to a target transition t if $t_c \notin T_{affecting}(t)$ and $\exists t_a \in T_{affecting}(t) | source(t_c) = source(t_a)$.*

Using Definition 5 it is possible to define $T_{critical}(t)$, the set of all critical transitions for t. Considering t_{13} of the ATM example as test purpose, $T_{critical}(t_{13}) = \{t_{11}, t_{12}, t_{19}, t_{20}\}$

C. Related works

In the context of model-based testing, Kalaji et al. [6] proposed the generation of feasible paths from an EFSM, using a genetic algorithm (GA). They proposed a fitness metric based on the data dependence analysis of a path. The testability of the model is improved in order to search feasible paths and a penalty value is assigned to a path according to the assignment type, guard type and its operator of the transitions. In [7], the metric was extended to deal with counter variables (i.e., variables that counts how many times a transition is repeated). The test case generation of Kalaji et al. requires two steps: one to generate the transition paths, and another one to generate test data to sensitize the paths. The MOST approach requires only one step for the generation of a test case. The problem of infeasible paths generation is considered in another way in our work: we use an executable model to obtain only feasible paths. Our evolutionary algorithm generates an input events sequence and the values of their parameters and evaluates the transition path triggered during model execution. A last difference is that in MOST the path length is not defined a priori: it is determined during test case generation.

There are works that investigate a search-based testing to generate a sequence of function calls [9], [13]. The objective function also uses the concept of approach level and branch distance. In [9], the sequence with their parameters is obtained during the search for a solution, as in our work. However, they use the chaining approach to represent the sequence and predefine the sequence length. In relation to [13] instead of a target path, we focus on the transition coverage. Then we do not have available a given path to guide the search and, for this reason, dependence analysis is used in our approach.

There are few evolutionary multi-objective methods applied to software testing. But all of them focus on different purposes: branch coverage and dynamic memory consumption of programs [14], coverage, cost and fault history for regression testing [15]. We are concerned with test purpose coverage and solution length.

The MOST approach is based on a previous work [8]. The main difference is the model analyzer component that uses dependence analysis, instead of slicing of the model. The objective functions are adapted to take into account this modification and also are improved to reward better a solution that covers the test purpose.

III. Test generation approach

The proposed approach, MOST, consists of the following steps:

1) develop the model M:
   a) elaborate an EFSM M;
   b) obtain an executable version of M;
   c) validate M;
2) analyze the dependences of M to obtain the transitions sets $T_{affecting}$ and $T_{critical}$;
3) generate the test cases.

The system under test is modeled according to the EFSM presented in Section II-A. Once created, the executable version of this model should be obtained (Section III-A). For the second step, the dependence information is obtained, as described in Section II-B. The evolutionary approach used for test case generation (third step) is explained in Sections III-B, III-C and III-D.

A. Construction of the executable model

The EFSM M is manually obtained from the system requirements. The model should be expressed in a way that allows its executable version to be created. Its executable version implements the behavior in a programming language. Since we use a model that is UML compliant, any tool supporting executable UML can be used to generate the model code. We use the SMC tool[1] (State Machine Compiler) that takes as input an EFSM and returns the source code of the model in different languages. We use the Java version of the source code generated by the tool, in order to keep the language compatibility used in the test case generator prototype. In the following, we show how the EFSM is associated with the Java code.

Definition 6. *A state $s \in S$ is associated with a Java class. A transition $t \in T$ with $source(t) = s$, where $s \in S$, is associated with a method of the class corresponding to s.*

When the method corresponding to a transition t is executed, the input event $i(t)$ is received and the guard $g(t)$ is verified, considering the current state s and $source(t) = s$. The action $a(t)$ is executed, whether $g(t)$ is true; otherwise, a null output is produced. In case an unexpected event is received, the machine remains in the same state and a null output is produced (completeness assumption).

As MOST is a model based testing technique, the test cases are derived from the model. Therefore, the validation of the model is important since the testing activity depends on the model. Using an executable model makes it easy for a non-expert user to validate the model: the user just runs the model using the input sequences to determine whether the behavior is as expected. The executable model is instrumented to monitor the triggered transitions, producing the path traversed by an input sequence during execution. Once validated, the model can be used for test generation.

B. Multi-objective evolutionary approach

Multi-objective optimization is the process of simultaneously optimizing two or more objective functions. This section presents why and how we use a multi-objective optimization for our model-based testing approach.

[1]Available in http://smc.sourceforge.net.

Multi-objective evolutionary algorithms intend to solve problems when the solutions need to meet several conflicting objectives simultaneously and no single optimal solution exists. In other words, there is no single solution that simultaneously optimizes each objective; the solution must adopt a trade-off among the objectives. This trade-off is known as Pareto optimal set and the corresponding objective functions values form the Pareto front. Each solution is non-dominated which means that it cannot be improved in any objective without causing degradation in at least one other objective. Without loss of generality, a multi-objective problem can be the minimization of $F(X) = [F_1(X), F_2(X), \ldots, F_{nof}(X)]$, where X represents the design variables and nof the number of objectives. A solution X_1 is said to dominate a solution X_2 if and only if

$$F_i(X_1) \leq F_i(X_2) \quad \forall i \in \{1, \ldots, nof\} \quad \wedge$$
$$F_j(X_1) < F_j(X_2) \quad \exists j \in \{1, \ldots, nof\}.$$

MOST forms the Pareto optimal set based on two criteria: coverage of a test purpose and solution length. As far as we know, MOST is the first model-based testing evolutionary approach that uses Pareto optimality for test case generation. The multi-objective approach was chosen due to the search space in MBT. Differently from code-based test generation approaches, in which a solution consists of the arguments values of the program under test and the number of the arguments is known, we generate a sequence of input events with their parameters. The length of the input sequence is, a priori, not limited, as a reactive system can never terminate. Thus in our approach, the sequence length is automatically determined during test case generation. A simple approach consists in letting the user define the length of the input sequence before starting the search [16], [17], [6]. However, in case the user chooses a length value that is too small for the algorithm to find a solution, it is necessary to give another value and start test case generation all over again. Furthermore, the aim is also to give the user the opportunity to choose a minimal length test case that covers the test purpose, when the test case execution takes too long.

As evolutionary algorithm, we use the M-GEO$_{vsl}$ (Multi-Objective Generalized Extreme Optimization with variable string length), presented in previous work [8]. As stated before, MOST uses two criteria for test case generation that are represented by two objective functions: the test purpose coverage (F_1) and the minimum length of the input sequence (F_2). The latter intends to minimize the input sequence length. In other words, we search for a minimum length of the input sequence but long enough to cover the test purpose. F_2 is given by (2) below, in which $|seq|$ represents the input sequence length. A value into the range [0,1] is added to the sequence size in the sense of penalizing unexpected inputs, since these inputs do not append transitions to the path.

$$Minimize: F_1 = AL + ND \quad (1)$$
$$F_2 = |seq| + 1 - 1.001^{-unexpectedInputs} \quad (2)$$

where

$$AL = 2 * |T_{affecting}| - RW$$
$$ND = 1 - 1.001^{-d}$$
$$RW = \begin{cases} |T'_{affecting}| & \text{, if not cover } t; \\ |T'_{affecting}| + |T_{affecting}| & \text{, if cover } t. \end{cases}$$

The objective function F_1 is calculated in terms of the approach level AL and normalized distance ND [5], as in structural testing, but with some slight modifications to consider EFSM features for the calculation of these terms. AL measures how close an input sequence is to reach a path that traverses the test purpose t. AL is calculated based on the dependence information using $T_{affecting}$. The transitions $t_a \in T_{affecting}(t)$ are used to guide the search toward t. The value of AL is minimized with relation to the number of transitions of $T_{affecting}$ that were triggered ($|T'_{affecting}|$) during model execution. If the sequence produces a path that traverses t (i.e., t is covered), the fitness value is rewarded with $|T_{affecting}|$. The normalized distance ND [18] is calculated at the point where the control flow takes a critical transition $t_c \in T_{critical}(t)$, that diverges away from a transition $t_a \in T_{affecting}(t)$. Thus the siblings transitions of t_a need to be analyzed. It is worth noting that more than two transitions can have the same source state in an EFSM, in contrast to code-based testing that only the true and false branch of a node are verified. Then to penalize the solution that takes a critical transition, two situations need to be considered for the calculation of the term d in ND: i. the input event of t_c and t_a are the same but the guards are different and ii. the input event of t_c and t_a are different. In the first case, the distance d is computed using the functions defined by Tracey et al. [19] (Table I). For example, if a guard transition $(x == y)$ needs to be evaluated as true to reach the test purpose, the distance function is defined as $abs(x - y) + K$ for transitions with the same input event. The value K is a non-zero positive constant which is always added if the term is not true. In the second case, taking t_c receives a penalty γ in order to distinguish solutions with different input events. It is important to note that $T_{affecting}$ and $T_{critical}$ are obtained in an early step of the evolution process. $T_{affecting}$ is obtained using the dependence graph, as described in Section II-B. Figure 2 shows how to determine $T_{critical}$ for a given test purpose t. When a transition is added to $T_{critical}$, we also inform which kind of penalty will be used in d calculation.

C. Input sequence encoding

The population of M-GEO$_{vsl}$ represents the input sequence. M-GEO$_{vsl}$ uses discrete encoding of the design variables. Each design variable of the population corresponds to a species. For each species is associated a fitness value, in contrast to GA, in which the fitness evaluation

input: test purpose t
$T_a = getTaffecting(t)$
for $\forall t_i \in T_a$ **do**
  $S = getSiblings(t_i)$
  for $\forall t_j \in S$ **do**
    if $t_j \notin T_a$ **then**
      if $event(t_j) == event(t_i)$ **then**
        $addTcritical(t_j, distanceFunction(t_i))$
      else
        $addTcritical(t_j, \gamma)$
      end if
    end if
  end for
end for
output: $T_{critical}(t)$

Figure 2. Algorithm to determine $T_{critical}$

Table I
DISTANCE FUNCTIONS

Operator	Distance function d
Boolean	if $TRUE$ then 0 else K
$a = b$	if $abs(a-b) = 0$ then 0 else $abs(a-b) + K$
$a \neq b$	if $abs(a-b) \neq 0$ then 0 else K
$a < b$	if $a - b < 0$ then 0 else $(a-b) + K$
$a \leq b$	if $a - b \leq 0$ then 0 else $(a-b) + K$
$a > b$	if $b - a < 0$ then 0 else $(b-a) + K$
$a \geq b$	if $b - a \leq 0$ then 0 else $(b-a) + K$
$\neg a$	Negation is moved inward and propagated over a
$a \wedge b$	$d(a) + d(b)$
$a \vee b$	$min(d(a), d(b))$
a xor b	$d((a \wedge \neg b) \vee (\neg a \wedge b))$ $\equiv min((d(a) + d(\neg b)), (d(\neg a) + d(b)))$

is related to a configuration of the design variables (an individual). The population consists of three parts (Figure 3): i. input sequence size, ii. sequence of input events, and iii. parameters of all input events. Thus all these items are generated during the evolution process. The population size is variable in relation to the input sequence size. If the sequence size is 10, the population will have 10 species representing input events, for instance. We assume that the parameters of the input events with identical name are the same, in this way the number of parameters is constant. For this reason, when the parameters do not represent the same information, they should have different names.

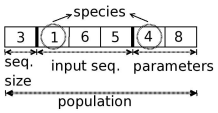

Figure 3. Population in M-GEO$_{vsl}$

For example, a population for M_1 (Figure 1) can be:

|seq|	input sequence	parameters
9	0 11 1 2 10 5 4 5 9	344 344 703 203 922 831 2 1 1 227

that represents the sequence $seq = \{card(344, 703, 203), print(2), pin(344), select(2), withdrawal(922), current(), done(), current(), deposit(831)\}$. The last part of the population represents the following parameters: $pin, p, sb, cb, w, d, id, id1, id2$ and $amnt$.

D. Evolution process

The evolution process of M-GEO$_{vsl}$ is illustrated in Figure 4. In first step, the population is initialized randomly with uniform distribution. All species are mutated temporarily, one at a time, to obtain the fitness of the species. The Pareto front and Pareto set are updated according to domination concept with the solutions that cover the test purpose. The value of each species is mutated to another in the corresponding domain. For example, a species that represents an input event will be mutated to another input event. Each new population is evaluated by F_1 and F_2. After this evaluation, the mutated value returns to its original one.

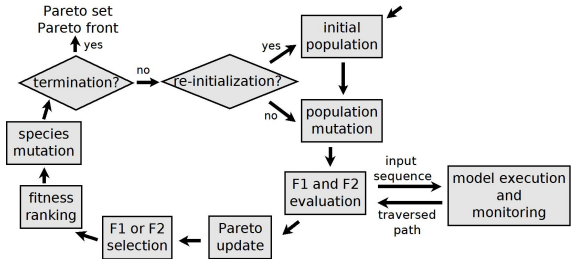

Figure 4. Evolution process of M-GEO$_{vsl}$

In next step, one objective function F_i is randomly chosen. All species are ranked by their fitness value in relation to F_i. According to this ranking, one species v is selected to be mutated using the probability distribution $P \sim k^{-\tau}$, where k is the position rank of v and τ an adjustable parameter of M-GEO$_{vsl}$. The algorithm has only the mutation operator. As the sequence size is a species, the algorithm can generate sequences with different numbers of input events, mutating the sequence size. In this way, the algorithm should deal with two situations when this sequence size is mutated: the new value of sequence length increases or decreases the current value. When the new value increases the current one, the sequence is completed with random values. In the other case, the extra input events in the sequence are ignored. Figure 5 shows different mutations. In the first case, an input event is mutated to a value in its domain and, in the other case, the sequence size is mutated: from the value 3 to 4, then one input event is also added into the input sequence, and from the value 3 to 2, then the last input event is ignored.

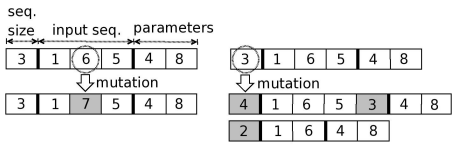

Figure 5. Mutations in M-GEO$_{vsl}$

The stop condition is the maximum number of evaluations of all objective functions. If this condition is not achieved, the algorithm verifies whether a re-initialization should be

started. A new point in the search space is found in a re-initialization, maintaining the solutions of the Pareto front.

To evaluate each solution, M-GEO$_{vsl}$ interacts with an executable model of the EFSM M under test. M-GEO$_{vsl}$ generates an input sequence and the executable model receives it as input. The executable model is instrumented to monitor the transitions triggered during execution. A transition is triggered when the corresponding input event is received and the associated guard is satisfied, considering the input parameters and/or variables involved in the condition.

It is worth noting that the terms input sequence and test case are distinct. An **input sequence** consists of a sequence of input events and their parameters. Some of the events in this sequence, although they are part of the EFSM input alphabet, may not correspond to a specified transition, as the machine may not be complete. A **test case**, on the other hand, is the **path** triggered during model execution according to the given input sequence. As the input sequence can contain nominal as well as unexpected events, the path length is not necessarily equal to the sequence size. For instance, the transition path triggered by seq (Section III-C) is $path = \{t_1 t_4 t_6 t_7 t_9 t_7 t_{13}\}$. Observe that $|path|$ is 5 whereas seq has 8 input events including 3 unexpected inputs. M-GEO$_{vsl}$ evaluates the input sequence using F_1 and F_2 according to the transition path produced during model execution.

IV. Experiments

This section describes the experiments conducted using MOST for test case generation. The experiments were aimed at answering the research questions below:

- Q_1: What is the cost of using the MOST approach?
- Q_2: How does the Pareto front guide a user of the MOST approach to select the test cases to be used?
- Q_3: How well do the test cases generated using the MOST approach compared to others generated with existing approaches?

In order to answer Q_1, we measured MOST performance in terms of the cost of test case generation: transition coverage, number of objective function evaluations, test case length and execution time. For question Q_2, we analyzed the size of the Pareto front; the wider this front, the larger the number of alternative solutions offered to the user. Moreover, the Pareto front gives the users information about the relationship between $T_{affecting}$ coverage and the input sequence length. The Pareto front is also used to answer Q_3, by analyzing the solutions which are dominated or non-dominated by the solutions obtained by Kalaji et al.'s approach [7].

A. Case studies

The case studies presented here are two EFSMs that were used to validate another evolutionary approach [7]. Table II presents the subject models. Each model is described in terms of number of states, transitions and variables. The last column indicates the CCS (Cyclomatic Complexity of State machine) [20] of the models. CCS is an adaption of metrics to measure design complexity of state models and is given by: $CCS = |T| + |I| + |A_G| + 2$, where T is the set of transitions, I is the set of input events and A_G is the set of atomic expressions in the guards. The ATM model (M_1) is an extension of the EFSM described in [21]. The model represents three services: deposit, withdrawal and transfer between two accounts. The second model (M_2) shows the initiator of the INRES protocol [22]. The protocol is a connection-oriented composed by the initiator and the responder. The initiator of the protocol establishes a connection and sends data. The responder receives data and terminates connections.

Table II
Experimental Models

Models	#States	#Transitions	#Events	#Parameters	CCS
M_1 : ATM	10	30	13	10	77
M_2 : INRES	4	16	7	2	34

B. Experimental set up

In order to apply MOST to the case studies, the steps presented in Section III were followed. Two tools were used in the experiments: SMC to obtain the executable models, presented in Section III-A, and SLIM (SLIcing state based Model) tool [10] for the dependence information. To adapt M-GEO$_{vsl}$ for the problem being tackled, it is necessary to tune the control parameter τ before starting test case generation. For each transition, we performed 5 runs with 10^5 function evaluations as stopping criterion and 50 re-initializations, with value of τ in the range [1,5] incremented by 0.25. We consider $K = 100$ and $\gamma = 1000$ for the calculation of the term d of F_1. The performance of M-GEO$_{vsl}$ presented the best results with $\tau = 3.75$. It was used a Pentium 4 with 3.00 GHz and 1 GB of RAM memory.

Once obtained the value of the sole adjustable parameter of M-GEO$_{vsl}$, the next step is to configure the algorithm execution by setting the values that follow. The input sequence length could be 500 at maximum. This limit avoids the generation of sequence length with the largest possible integer, but long enough for the subjects. Each input event in the sequence may range from 1 to $|I|$, where I is the input alphabet of the model. The number of parameters in a sequence is constant, as mentioned in Section III-C. Table II presents the number of parameters for each model. The input parameters are also considered as positive integer values.

Each transition of the subject model was taken as test purpose. A total of 10 runs per test purpose were performed in M-GEO$_{vsl}$. For each run, the stopping criterion was 10^6 objective function evaluations and the number of re-initializations was 100. The value of the parameter $\tau = 3.75$ was used for the mutation operation, considering the results

of the tuning phase. For the calculation of F_1, we considered $K = 100$ and $\gamma = 1000$.

C. Results

Q_1 : Using MOST approach, we obtained 100% coverage of the transitions of both models M_1 and M_2, meaning that at least one path was generated to cover each transition of the models. As a multi-objective approach, M-GEO$_{vsl}$ can produce more than one successful solution that achieves the test purpose. Due to space limitation, Tables III and IV show only the shortest path obtained for each transition of ATM and INRES, respectively. It also presents the path length, the corresponding input sequence size and the number of objective function evaluations to find the solution. It is worth noting that although the stopping condition for M-GEO$_{vsl}$ was 10^6 evaluations, less than half of the number of objective function evaluations is necessary to obtain a solution, on average. Each run of M-GEO$_{vsl}$ took approximately 22.28 seconds.

Table III
ATM: SHORTEST PATH FOUND FOR EACH TRANSITION

t_i	Path	|Path|	|Seq|	#Eval
t_1	t_1	1	1	437560
t_2	t_1t_2	2	2	478392
t_3	$t_1t_2t_2t_2t_3t_1$	6	7	666060
t_4	t_1t_4	2	4	329642
t_5	$t_1t_4t_5t_{23}$	4	17	8934
t_6	$t_1t_4t_6t_7t_9$	5	11	397514
t_7	$t_1t_4t_5t_8t_{10}t_7t_9$	7	22	588480
t_8	$t_1t_4t_6t_8t_{20}t_{22}$	6	15	644208
t_9	$t_1t_4t_5t_7t_9$	5	13	423358
t_{10}	$t_1t_4t_5t_8t_{10}$	5	17	258314
t_{11}	$t_1t_4t_5t_7t_{11}$	5	25	66796
t_{12}	$t_1t_4t_6t_{23}t_1t_4t_6t_7t_{12}$	9	23	48324
t_{13}	$t_1t_4t_6t_7t_9t_7t_{13}$	7	28	732390
t_{14}	$t_1t_4t_5t_7t_{14}t_{15}t_{14}$	7	27	269752
t_{15}	$t_1t_4t_5t_7t_{13}t_{15}$	6	21	835058
t_{16}	$t_1t_4t_6t_{25}t_{26}t_7t_{12}t_{16}t_{13}t_{16}t_9t_8t_{10}$	13	45	879412
t_{17}	$t_1t_4t_6t_8t_{17}t_{22}t_{10}t_7t_9$	9	46	889210
t_{18}	$t_1t_4t_6t_7t_9t_7t_9t_8t_{17}t_{22}t_{18}$	11	32	209604
t_{19}	$t_1t_4t_5t_{25}t_{26}t_8t_{19}$	7	18	883308
t_{20}	$t_1t_4t_6t_{25}t_{26}t_8t_{20}t_{22}$	8	28	409840
t_{21}	$t_1t_4t_5t_7t_9t_8t_{19}t_{21}t_{18}t_{21}t_{19}$	11	37	826070
t_{22}	$t_1t_4t_6t_7t_9t_8t_{20}t_{22}t_{18}$	9	39	239506
t_{23}	$t_1t_4t_5t_7t_9t_{23}$	6	19	976300
t_{24}	$t_1t_4t_{24}$	3	7	868388
t_{25}	$t_1t_4t_5t_8t_{10}t_{25}$	6	20	772902
t_{26}	$t_1t_4t_5t_8t_{10}t_{25}t_{26}$	7	17	662680
t_{27}	$t_1t_4t_5t_8t_{18}t_{21}t_{10}t_{25}t_{27}t_{30}t_{26}t_{23}$	12	39	602256
t_{28}	$t_1t_4t_5t_{25}t_{28}$	5	39	458002
t_{29}	$t_1t_4t_6t_8t_{10}t_{25}t_{28}t_{29}t_{28}$	9	41	77010
t_{30}	$t_1t_4t_5t_{25}t_{28}t_{30}$	6	15	289372
	Average:	6.63	22.50	507621.40

Q_2 : Each solution found by M-GEO$_{vsl}$ represents a point in the Pareto front. The Pareto front shows the trade-off between the objective functions F_1 and F_2. To illustrate, we present the Pareto front for transition t_4 of M_1 (Figure 6) and for transition t_{12} of M_2 (Figure 7). When the solutions traverse more transitions of $T_{affecting}$ (i.e, minimizing F_1), the input sequence is the longest one. On the other hand, for the shorter input sequence (i.e., minimizing F_2), we have the worst coverage of the transitions of $T_{affecting}$.

Table IV
INRES: SHORTER PATH FOUND FOR EACH TRANSITION

t_i	Path	|Path|	|Seq|	#Eval
t_0	t_0	1	1	7944
t_1	t_0t_1	2	2	585602
t_2	$t_0t_1t_2$	3	4	102036
t_3	$t_0t_1t_3t_{13}$	4	4	83638
t_4	$t_0t_{12}t_{12}t_1t_3t_3t_3t_3t_4$	9	14	95694
t_5	$t_0t_1t_2t_5t_8$	5	5	117010
t_6	$t_0t_1t_3t_3t_2t_5t_7t_5t_6$	9	12	699532
t_7	$t_0t_1t_2t_5t_8t_7$	6	8	586992
t_8	$t_0t_1t_{13}t_1t_2t_5t_8t_8$	8	8	649950
t_9	$t_0t_1t_2t_5t_8t_{10}t_8t_8t_9$	9	11	882778
t_{10}	$t_0t_1t_2t_5t_8t_{10}t_7t_5$	8	9	146990
t_{11}	$t_0t_1t_2t_5t_8t_{10}t_8t_8t_{11}$	9	19	808730
t_{12}	$t_0t_{12}t_1$	3	3	784220
t_{13}	$t_0t_1t_{13}$	3	4	716142
t_{14}	$t_0t_1t_2t_{14}$	4	5	848320
t_{15}	$t_0t_1t_2t_5t_{10}t_8t_{15}t_1$	8	10	477806
	Average:	5.69	7.44	474586.50

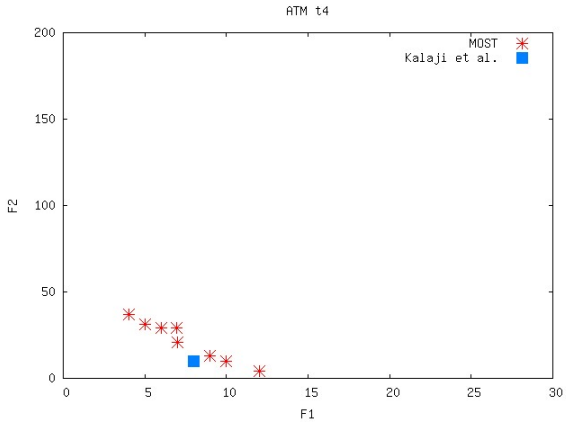

Figure 6. ATM: Pareto front for t_4

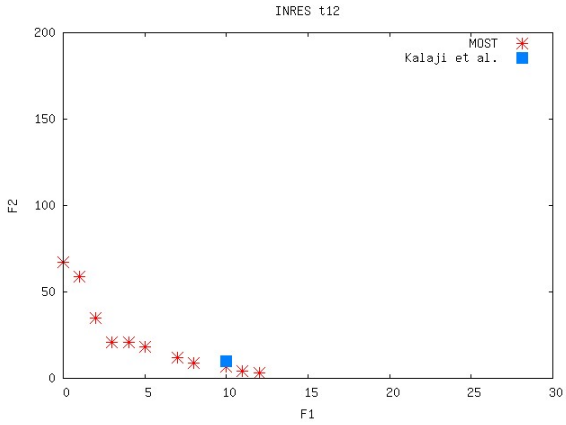

Figure 7. INRES: Pareto front for t_{12}

Figure 8 and 9 show the number of points in the Pareto front for each transition of M_1 and M_2, respectively. It can be noted that, in general, the Pareto front has more than one point, which means that M-GEO$_{vsl}$ obtains more than one path, that can be used to traverse the test purpose. The

more solutions in the Pareto front, the more alternatives the test team have. For example, when test case execution time is high, shorter test cases can be selected. However, if the goal is to reduce the number of test cases, the option may be the use of larger test cases, covering more transitions of $T_{affecting}$. Thus, it is possible to cover several transitions at once, generating a smaller number of test cases and obtaining also a reduction in the cost of test case generation.

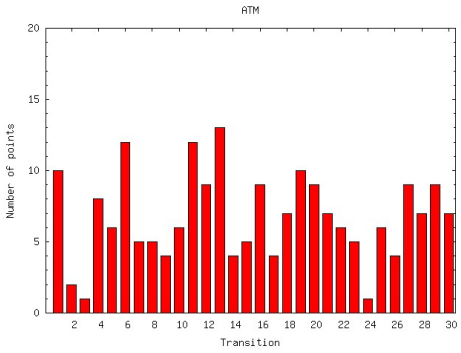

Figure 8. ATM: Number of points in the Pareto front

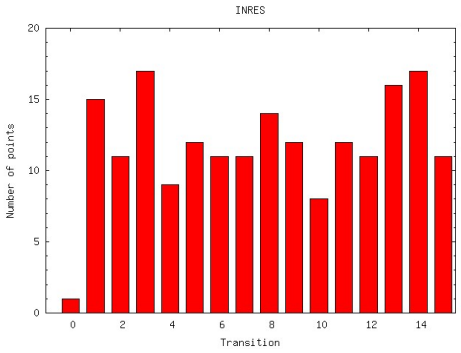

Figure 9. INRES: Number of points in the Pareto front

When the Pareto front has one point, only one solution was generated. Then, this solution dominated all others generated by M-GEO$_{vsl}$ in relation to both objective functions F_1 and F_2. This means that this solution presented better values considering F_1 and F_2 than other generated solutions. For instance, there is only one path for the transition t_{24} of M_1. In this case, M-GEO$_{vsl}$ found the shortest path from the initial state to t_{24} ($path_{24} = \{t_1 t_4 t_{24}\}$). To trigger t_{24}, the transitions t_1 and t_4 should be traversed first and the guards $g(t_4)$ and $g(t_{24})$ should be satisfied. t_4 is triggered as long as the provided pin is the one expected and the number of attempts to enter the correct pin is less than 3; while $g(t_{24})$ is only satisfied when the select language id is equal to zero. M-GEO$_{vsl}$ generated the parameters values to allow $path_{24}$ to be traversed: the parameters p and pin received the same value, then the context variable $attempts$ remained zero, and id received value zero. Observing Table III, the length of the input sequence generated for t_{24} is 7, then the sequence contains 4 unexpected input events. Furthermore, from Table III and IV, it is worth noting that the test case length vary according to the test purpose.

Q_3 : For the comparison with existing works, the closest we could find of our approach was the works of Kalaji et al. [6], [7]. Although a mono-objective approach is applied, they use only the model for test generation, that is, the system source code is not necessary, such as in [16]. We compare the results of MOST to the test cases obtained in their recent work [7] that also considers M_1 and M_2. A genetic algorithm was used to find a path to cover each transition of the models. They fixed the length of all paths in 10 and performed 1000 generations with a population of 25 individuals. Each path was evaluated using F_1 and F_2, obtaining the point p_k in the objective function space. For all transitions, the Pareto front points p_p obtained by MOST were 70.77% non-dominated by p_k for M_1 and 96.20% for M_2, on average. Then, in general, p_k is dominated by the Pareto front, that means the Pareto front has better results considering F_1 and F_2 than p_k. For instance, Figures 6 and 7 show the comparison between the Pareto front and p_k (indicated by ■). There are situations in which p_k dominates some points of the Pareto front p_p. In this case, considering F_1, we analyzed that the path length of these points p_p is smaller than 10. As p_p presents smaller path than p_k, p_p does not contain as many transitions of $T_{affecting}$ as p_k. It is worth remembering that F_2 represents the size of the input sequence that can have unexpected inputs, and not the path length. When p_k dominates the points p_p in relation to F_2, although p_p is longer solution, p_p has more transitions of $T_{affecting}$. Note that in Figure 6 the solution p_k for M_1 is not dominated by any other solution found in MOST. Therefore, in this case, the Pareto front can include the point p_k. For M_2, Figure 7 shows that the point p_k is clearly dominated by p_p and can not be part of the Pareto front.

V. Conclusions

This paper presents the MOST approach, a search-based testing technique for test case generation from EFSM. MOST uses a multi-objective evolutionary algorithm to allow the generation of test cases to cover a given transition (test purpose). Then more than one successful path can be found to cover the test purpose. Using MOST, we can obtain the transition paths (test cases) as well as the data to trigger them. In the evolution process, the control and data aspects of the model are considered in one step. The value of the input sequence length can be found automatically, then it is not necessary to establish a priori the test case size as other approaches. In order to guide the search for the test purpose, we proposed an objective function that uses information gathered from a dependence analysis of the model, so that a solution is the one that covers most of the transitions on

which the test purpose depends. We consider both control and data dependence analysis.

The proposed approach uses an executable model for test case generation, to avoid infeasible path generation, which is a problem when considering control and data aspects. By executing the model in the search for the solution, we can be sure that the solution is feasible, at least at model level. Moreover, the model execution can be more attractive than executing the system implementation, since SBST requires many executions of the system under test. We can also use the executable model to validate the EFSM, an important issue as the test cases in model-based testing are derived from the model. Furthermore, model-based evolutionary testing is useful when the source is not available.

The results of MOST were compared to another search-based testing approach for EFSM. Considering F_1 and F_2, we obtained better or similar results, in general.

Further experiments are being performed, taking into account other models, not only the ones used as benchmark, but also from real-world applications. A future work is to consider only the parameters involved in the generated input sequence, instead of optimizing all parameters of the model.

ACKNOWLEDGMENT

This research is supported by CAPES, CNPq and Serasa Experian. The authors also wish to thank Zheng Li for providing the dependence analysis of the models.

REFERENCES

[1] G. Bochmann and A. Petrenko, "Protocol testing: Review of methods and relevance for software testing," in *International Symposium on Software Testing and Analysis (ISSTA'94)*, 1994, pp. 109–124.

[2] B. Sarikaya, G. Bochmann, and E. Cerny, "A test design methodology for protocol testing," *IEEE Transactions on Software Engineering*, vol. 13, no. 5, pp. 518–531, May 1987.

[3] C. Bourhfir, R. Dssouli, and E. M. Aboulhamid, "Automatic test generation for EFSM-based systems," University of Montreal, Canada, Tech. Rep. IRO 1043, Aug. 1996.

[4] A. Y. Duale and M. U. Uyar, "A method enabling feasible conformance test sequence generation for EFSM models," *IEEE Trans. on Computers*, vol. 53, no. 5, pp. 614–627, 2004.

[5] P. McMinn, "Search-based software test data generation: a survey," *Software Testing, Verification and Reliability*, vol. 14, no. 2, pp. 105–156, 2004.

[6] A. S. Kalaji, R. M. Hierons, and S. Swift, "Generating feasible transition paths for testing from an extended finite state machine," in *Proc. ICST'09*, 2009, pp. 230–239.

[7] A. S. Kalaji, R. Hierons, and S. Swift, "Generating feasible transition paths for from an extended finite state machine (EFSM) with the counter problem," in *Proc. SBST'10: 3rd Int. Workshop on Search-Based Software Testing*, 2010, pp. 232–235.

[8] T. Yano, E. Martins, and F. L. De Sousa, "Generating feasible test paths from an executable model using a multi-objective approach," in *Proc. SBST'10: 3rd Int. Workshop on Search-Based Software Testing*, 2010, pp. 236–239.

[9] P. McMinn and M. Holcombe, "Evolutionary testing of state-based programs," in *Proc. GECCO'05*, 2005, pp. 1013–1020.

[10] K. Androutsopoulos, N. Gold, M. Harman, Z. Li, and L. Tratt, "A theoretical and empirical study of EFSM dependence," in *Proc. ICSM'09: 25th IEEE Int. Conf. on Software Maintenance*, Sep. 2009, pp. 287–296.

[11] K. Androutsopoulos, D. Clark, M. Harman, Z. Li, and L. Tratt, "Control dependence for extended finite state machines," in *Proc. FASE '09: 12th Int. Conf. on Fundamental Approaches to Software Engineering*, 2009, pp. 216–230.

[12] P. McMinn, M. Harman, D. Binkley, and P. Tonella, "The species per path approach to searchbased test data generation," in *Proc. ISSTA'06*, 2006, pp. 13–24.

[13] R. Lefticaru and F. Ipate, "Functional search-based testing from state machines," in *Proc. ICST '08*, 2008, pp. 525–528.

[14] K. Lakhotia, M. Harman, and P. McMinn, "A multi-objective approach to search-based test data generation," in *Proc. GECCO'07*, 2007, pp. 1098–1105.

[15] S. Yoo and M. Harman, "Pareto efficient multi-objective test case selection," in *Proc. ISSTA '07*, 2007, pp. 140–150.

[16] A. Baresel, H. Pohlheim, and S. Sadeghipour, "Structural and functional sequence test of dynamic and state-based software with evolutionary algorithms," in *Proc. GECCO'03*, 2003, pp. 2428–2441.

[17] A. R. Cavalli, D. Lee, C. Rinderknecht, and F. Zaïdi, "Hit-or-jump: An algorithm for embedded testing with applications to IN services," in *Proc. of the IFIP TC6 WG6.1*. Deventer, The Netherlands: Kluwer, B.V., 1999, pp. 41–56.

[18] A. Baresel, H. Sthamer, and M. Schmidt, "Fitness function design to improve evolutionary structural testing," in *Proc. GECCO'02*, 2002, pp. 1329–1336.

[19] N. Tracey, J. Clark, J. McDermid, and K. Mander, "A search-based automated test-data generation framework for safety-critical systems," *Systems engineering for business process change: new directions*, pp. 174–213, 2002.

[20] S. Wagner and J. Jürjens, "Model-based identification of fault-prone components," in *Proc. EDCC-5: Fifth European Dependable Computing Conference, volume 3463 of LNCS*. Springer, 2005, pp. 435–452.

[21] B. Korel, I. Singh, L. Tahat, and B. Vaysburg, "Slicing of state-based models," in *Proc. ICSM'03: Int. Conf. on Software Maintenance*, 2003, pp. 34–43.

[22] D. Hogrefe, "OSI formal specification case study: the inres protocol and service," University of Bern, Tech. Rep. IAM-91-012, 1991.