doi: 10.5028/jatm.v6i4.369

FSM-Based Test Case Generation Methods Applied to Test the Communication Software on Board the ITASAT University Satellite: A Case Study

Arineiza C. Pinheiro[1], Adenilso Simão[1], Ana Maria Ambrosio[2]

ABSTRACT: The software in satellite applications has become increasingly larger, more complex and more integrated, so its verification and validation require exploration of new approaches. In this paper we present a Model-Based Testing (MBT) approach applied to the Communication Module of the ITASAT-1 university satellite. The models are Finite State Machines (FSM) representing the software behavior. In order to manage the difficulties to model the software behavior the approach employs the Conformance and Fault Injection (CoFI) testing methodology associated with the JPlavisFSM tool in the real context of a satellite's critical software. The former advises the modularization of the modelling into different types of behavior into different FSMs, while the latter integrates several FSM-based methods to derive test cases, provides facilities to design and to check properties of the models and computes metrics. The main result of this case study was the evaluation of the drawbacks on the design of the testing models supported by CoFI and JPlavisFSM. The models, test sets, metrics with the application of our approach applied to the Communication Module are presented. The paper discusses the benefits as well as the points requiring new researches.

KEYWORDS: Finite state machine, Model-based testing, Test-case generation methods, Testing methodology.

INTRODUCTION

The increasing development of university satellites has allowed researchers to experiment new approaches in space area. In traditional satellite development, experimental methods are usually avoided to prevent risks and cost increase, whereas university satellites open an opportunity to explore new approaches using a real system (Alencar, 2013).

The software for satellite applications has become increasingly more complex, as it includes more functions and is more integrated. The development of satellite-related software usually requires a series of rigorous tests, along with all the satellite development phases. The development tendency of this kind of software points out the use of formal models for the designing and testing. To adopt formal models, such as the Finite State Machines (FSMs), not only does it benefit the identification of ambiguities and gaps in the requirements (Morais and Ambrosio, 2010; Morais, 2011; Pontes *et al.*, 2012), but it also makes the automatic generation of test cases feasible (Ambrosio *et al.*, 2005; Romero *et al.*, 2012) by applying the vast theory of automatic test case generation.

The specification of test cases for embedded software in satellites should take into account characteristics such as: real-time requirements, integration of technologies and fault tolerance mechanisms. The occurrence of failures in this kind of software may cause large losses, so they should be thoroughly tested using systematic and rigorous approaches, in which

1. Instituto de Ciências Matemáticas e de Computação/USP – São Carlos/SP – Brazil **2.**Instituto Nacional de Pesquisas Espaciais – São José dos Campos/SP – Brazil.
Author for correspondence: Ana Maria Ambrosio | Instituto Nacional de Pesquisas Espaciais | Avenida dos Astronautas, 1.758, CEP: 12.227-010 –São José dos Campos/SP | Brazil | Email: ana.ambrosio@inpe.br
Received: 04/15/2014 | **Accepted:** 10/20/2014

various testing techniques are combined to exercise different aspects of the system.

The Model-based Testing (MBT) is an approach that reduces cost with the automation of test cases generation and with the reuse of models created during the software designing and testing activities; besides that, it provides high fault detection and traceability. In MBT, the test designer can specify the test model using different modeling notations, such as FSMs, Labeled Transition Systems (LTS), Unified Modeling Language (UML) state machines, etc. FSM is a formal modeling technique, adopted due to its rigor and simplicity. FSM-based testing has been studied for several decades (Moore, 1956; Chow, 1978) and it has still presented contributions (Simão *et al.*, 2009; Simão and Petrenko, 2010; Yin *et al.*, 2010; Pedrosa and Moura, 2010; Hierons and Ural, 2010; Simão *et al.*, 2005) such as a tool that integrates different FSM-based methods into a single software product named PLAVIS (PLAtform for Software Validation & Integration on Space Systems); similar work is given in Santiago *et al.* (2008).

In MBT, supporting tool plays an essential role, since the cost of building the models for testing only should be balanced by the possibility of generating and executing larger test suites (Utting and Legeard, 2007). However, the difficulties still remain to build the models. In our work, we have applied the Conformance and Fault Injection (CoFI) methodology (Ambrosio, 2005) in order to guide the tester to create the models associated with the JPlavisFSM tool (Pinheiro, 2012). CoFI was firstly proposed to lead a tester to understand the problem well enough while creating a set of FSMs representing normal and abnormal behavior of software on-board a satellite. It defines detailed steps to apply FSM-based testing tools, in a systematic way and it has been applied in several cases (Ambrosio *et al.*, 2007; Pontes *et al.*, 2009; Anjos *et al.*, 2011; Mattiello-Francisco *et al.*, 2013). JPlavisFSM tool comprises not only four methods to automatically generate test cases starting from a given specification in FSM but also the Mutation Testing technique (Fabbri *et al.*, 1994), that allows the adequacy evaluation of a test cases set.

This paper presents a case study in which the CoFI methodology and the JPlavisFSM tool are combined to test the Communication Module of the ITASAT-1 satellite (Sato *et al.*, 2011). All the steps applied to design the models have been illustrated. We employed the JPlavisFSM tool to the test case generation, exploring all the FSM-based methods available. The main result of this case study was the identification of the drawbacks to create the testing models that is the key activity to successfully apply MBT techniques to satellite-related software. On the modelling work, we found that the previous knowledge on the testing methods theory had facilitated the modelling process, which took 16 days only. The length of time to understand the system under test, to do the modeling and to generate the test cases accounted for 33 days. We did not evaluate how the models impacted on the quality of the tests, as the tests were not executed against the implementation of the Communication Module software, because the implementation was not finished before the conclusion of this work. However, we evaluated the quality of the test sets by mutation analysis applied to FSM.

The remainder of this paper presents the background, the CoFI and the JPlavisFSM tool, the case study, and finally, discusses the MBT applicability and the conclusions.

BACKGROUND

In this section, we introduce the main concepts necessary for understanding the results presented in this paper.

FINITE STATE MACHINES

Finite State Machines (FSMs) have been widely used for modeling reactive systems, ranging from simple protocols to complex embedded systems (Lai and Leung, 1995). Among the main advantages of FSMs, there are the solid theoretical background, the expressiveness power and the existence of numerous methods to generate test cases. FSMs are hypothetical machines composed of states and events, which correspond to transitions between the states. A transition is associated to two kinds of events: input and output. When an input event occurs in a given state, the FSM responds with an output event and may move to another state (Gill, 1962).

Formally, a Mealy FSM is defined as a tuple $M = (S, s_0, X, Y, D, \lambda, \delta)$, where: S is the nonempty finite set of states; s_0 is the initial state ($s_0 \in S$); X is the nonempty finite set of input symbols; Y is the nonempty finite set of output symbols; $D \subseteq (S \times X)$ is the specification domain; $\lambda : D \rightarrow Y$ is the output function; and $\delta : D \rightarrow S$ is the transition function.

Figure 1 shows an FSM example with 4 states and 8 transitions, where s_0 is the initial state, $S = (s0, s1, s2, s3)$, $X = (a, b)$ and $Y = (0, 1)$.

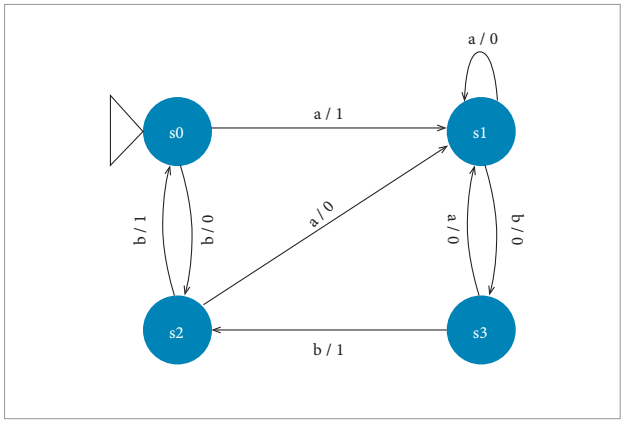

Figure 1. Example of FSM.

Given state s and input x, the transition (s, x) is *defined* if $(s, x) \in D$. An input sequence $\alpha = x_1x_2\ldots x_k$ is *defined* at state s if there exists a sequence of states $s_1, s_2, \ldots, s_{k+1}$, so that $s = s_1$ and $(s_i, x_i) \in D$, $\delta(s_i, x_i) = s_{i+1}$, for $1 \leq i \leq k$. The set of input sequence defined at state s is denoted $\Omega_M(s_i)$. The output and transition functions are extended to defined input sequences in the usual way; given a state s and the empty sequence ϵ, we have that $\delta(s, \epsilon) = s$ and $\lambda(s, \epsilon) = \epsilon$; given input sequence α and input x, we have that $\delta(s, \alpha x) = \delta(\delta(s, \alpha), x)$ and $\lambda(s, \alpha x) = \lambda(s, \alpha)\lambda(\delta(s, \alpha), x)$. A set of input sequences is initialized if it contains the empty sequence. An input sequence α separates states s and s', if $\lambda(s, \alpha) \neq \lambda(s', \alpha)$. A set of sequences is an *identifier* for state s, denoted $Id(s)$, if for each state $s' \neq s$, there exists a sequence in $Id(s)$ which separates s and s'. A set containing one identifier for each state is *harmonized*, if for each pair of state s and s', there exists an input sequence $\alpha \in Id(s) \cap Id(s')$ which separates them. An input sequence α is a *Unique Input/Output Sequence* for state s, denoted $UIO(s)$, if $\{\alpha\}$ is an identifier for s. A set of input sequence W is a *characterization set* for each pair of state s and s', if there exists an input sequence in W which separates them. A *preamble* for state s is an input sequence α, so that $\delta(s_0, \alpha) = s$, i.e., it takes the FSM from the initial state to s. A *state cover* is a set containing preambles for each state. A *transition cover* is a set P of input sequences, so that, for each defined transition (s, x), there are α and αx in P, where α is a preamble for s.

The FSM M is defined over X. Based on this definition, some structural properties in FSM, which are important for test case generation methods, are listed:

- *Completely specified* – an FSM is *completely specified*, or *complete* if all states (S) have a transition for each input event from the set of input symbols (X), so that $D = (S \times X)$. Otherwise, the FSM is *partially specified*, or *partial*;
- *Strongly connected* – an FSM is *strongly connected* if for every pair of states (s_i, s_j) there is a path that leads s_i to s_j, i.e., there is some input event sequence that performs a path of transitions with source in s_i and destination of s_j. If all other states can be reached from the initial state, the FSM is initially connected;
- *Deterministic* – an FSM is *deterministic* when there is only one transition with a particular input event from any state that allows transition to a next state. Otherwise, the FSM is nondeterministic;
- *Equivalent* – one state s_i is considered *equivalent* to s_j if there is no input event sequence which, when executed from the respective states, generates a different output sequence;
- *Reduced* – an FSM is *reduced* if there is no pair of equivalent states. Otherwise, it is unreduced.

FINITE STATE MACHINE (FSM)-BASED TESTING

In the context of FSM-based testing, a finite sequence of input events is a *test case*, or just a *test*. A *set of test cases* is a finite set of sequences, so that there are no two sequences α and $\beta \in T$, where α is a proper prefix of β. (There are two sequences α and β, α is prefix of β, if $\alpha \leq \beta$, such as $\alpha\omega = \beta$, to any ω. And, α is a proper prefix of β, if $\alpha < \beta$, such as $\alpha\omega = \beta$).

Given two FSMs M and I; two states s from M and t from I are distinguishable if there is an input sequence $\gamma \in \Omega_M(s) \cap \Omega_I(t)$, called *separating sequence*, so that the same input event sequence generates different output sequences for both FSMs, i.e., $\lambda(s, \gamma) \neq \lambda(t, \gamma)$. Two FSMs I and M are *distinguishable* if their respective initial states are distinguishable.

A *fault domain* $\Gamma(X)$ is the set of all possible implementations of M defined over the set of input symbols X. Similarly, $\Gamma_n(X)$ denotes the set of all FSM defined over the set of input symbols X with at most n states. The test set T is said *n-complete,* or just *complete*, for the specification M if for all $I \in \Gamma_n(X)$, so that I is distinguishable from M, there is at least one sequence $\alpha \in T$ that produces different output sequences when applied to M and I in the respective initial states. In other words, a test set is complete when it is possible to distinguish the specified FSM with n states from all other distinguishable FSM with the same set of input symbols and at most n states.

The main objective of the FSM-based testing is to compare a reduced FSM *model M* with *n* states to an *implementation* of the FSM model, which is assumed to be represented by an unknown FSM *I*. When a reduced FSM *M* which represents the correct version of the specified FSM is compared to FSM *I,* the following faults may be revealed (Chow, 1978):

- Output Error: *I* and *M* differ in the output of a transition;
- Transference Error: *I* and *M* differ in the final state. When the final state achieved in both FSM is different after applying a test case;
- Transition Error: it is the general term for output or transference error;
- Missing States: *I* has fewer states than *M;*
- Extra States: *I* has more states than *M.*

Several methods have been proposed for generating set of test cases which guarantee that the implementation does not contain any of the above listed faults, that is, which are complete regarding these faults. The completeness of the generated test case is proved by showing that no faulty implementation FSM would pass the test cases. The main differences among them are the cost to generate the sequences and the effectiveness (the power of the test cases to detect faults). One of the first methods to be proposed was the W (Chow, 1978), which is considered a precursor of the area, since most of the following methods are based on it: UIO (Sabnani and Dahbura, 1988), UIOv (Vuong *et al*., 1989), Wp (Fujiwara *et al*., 1991), HSI (Petrenko *et al*., 1993) and SPY (Simão *et al*., 2009).

MUTATION TESTING

The Mutants Analysis is one of the most popular criteria of Defects-based testing, which aims at deriving test requirements from knowledge about typical mistakes made by designers or developers. Mutation Testing is a testing technique that can be widely employed as a way of evaluating the test cases generated. The criteria can be used as:

- Black-box testing, when we consider the specification as a test artifact, or
- White-box testing, when the artefact considered is the source code. In this context, the test artifact could be represented formally by a model, as FSM.

The Mutants Analysis consists of assessing the adequacy of a test set *T* for the test artifact *P*. First, you run the test set *T* against P. If a fault occurs, a defect was found. Otherwise, it is assumed that some defects already exist but cannot even be detected.

The next step is to derive one or more products from the original artefact *P*, which gives the new products P_1, P_2, ..., P_n, also called mutants of *P*. Mutants are generated from mutation operators, which are dependent on the programming language and determine what types of syntactic changes can be made in the artefact. The mutation operators' aim:

- Inducing simple syntactic changes based on the typical mistakes made by developers, such as changing the value of a constant, or
- Forcing certain test goals, such as performing each node or arc of the product (Delamaro *et al.*, 2007).

The test set T is executed against each of the mutants generated. The main objective is to "kill" all mutants. It is expected that, due to the changes made, the mutants have different behavior from the original product, featuring a defect.

Occasionally, when the artifact is a source code, a mutant and the original product can still have the same result for any test case in the set. In these cases, the tester has to check the code and determine whether there is equivalence between the products. If so, the mutant is called as equivalent. The equivalence between programs is an undecidable problem and, therefore, there is no automated solution to the problem and the manual intervention of the tester is necessary. However, when the product is a model, there is no equivalency problem, because it could be determined automatically due to the formality of the FSM.

After the mutants' execution and equivalence analysis, the mutation score is calculated. The aim is to determine the adequacy of test cases used in a range from 0 to 1, providing a quantitative measure of how efficient the test set is in revealing the difference between the mutant and the original product. The mutation score *ms*(P, T), to product *P* and the set of test cases *T* is calculated as:

$$ms(P,T) = \frac{DM(P,T)}{M(P) - EM(P)}$$

where: DM (P, T): number of mutants killed by *T*; M (P): total number of mutants generated from *P*, and EM (P): number of generated mutants that are equivalent to *P* (Delamaro *et al.*, 2007).

The mutation score is obtained from the ratio between the number of mutants killed by the set *T* and the number of mutants that could be killed, given by the difference between the total number of mutants generated and the number of mutants classified as equivalent.

THE JPLAVISFSM TOOL AND THE CONFORMANCE AND FAULT INJECTION METHODOLOGY

This section presents the main features of the JPlavisFSM tool and the CoFI methodology which were combined to guide the creation of FSMs, representing the system's behavior.

JPLAVISFSM TOOL

In a joint cooperation project among Brazilian researchers, the test platform named PLAVIS was conceived. This platform integrated existing tools for automatic generation of test cases starting from FSM, which had been developed by PLAVIS-project's members. PLAVIS was a web-based test platform providing different functions to support FSM-based testing (Simão *et al.*, 2005). Years later, Pinheiro (2012) improved usability features in the previous platform, now named JPlavisFSM tool.

The JPlavisFSM is a standalone software, whose graphical user interface (GUI) is imported from the open source tool JFlap (http://www.jflap.org/) to design the FSM. From the GUI, one can easily draw states and transitions, edit and adjust them before automatically generating a set of test cases as well as evaluating the adequacy of a set of test cases.

The JPlavisFSM tool includes a function to analyze FSM structural properties namely *complete, strongly connected, deterministic*, etc. This kind of analysis is very important because the applicability of the test methods depends on presence or absence of some structural properties. For example, the W method requires a FSM with the properties of *complete, strongly connected, deterministic*, and *reduced*. One advantage is to reduce the need of theoretical knowledge, particularly for testers unfamiliar with such theory.

Other functionalities were implemented to assist test case handling, such as test set execution, inclusion/exclusion and enabling/disabling. Test sessions can be created, as shown in Fig. 2. In this window, the user can visualize the FSM on the left and the generated test cases on the right side (each line has one test case). At the bottom, there is an option that helps the tester to create particular test sets by himself.

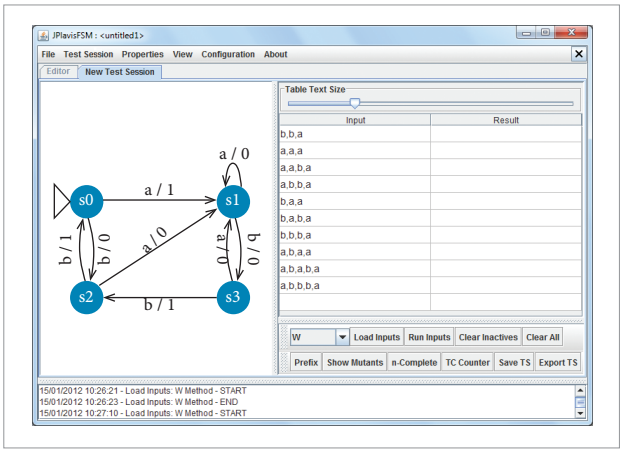

Figure 2. JPlavisFSM Tool.

One of the best features of the JPlavisFSM's test session is to support the construction of a test case set with a high probability of finding defects in the System Under Test (SUT). In a test session, the tester can import (or load) a set of test cases, which was automatically generated through one of the testing methods given by JPlavisFSM or manually generated by the tester. Once imported, the set of test cases can be executed (i.e., applied against the specification) and the mutation score obtained.

The JPlavisFSM releases the following testing methods to generate test cases set: W, UIO, HSI and SPY. The W method generates test cases as follows. First, an initialized transition cover and a characterization set of the specification are determined. Secondly, for each sequence of the transition cover, each sequence of characterization set is appended. The UIO generates test cases as follows. First, an initialized transition cover is determined, as well as UIOs for each state. Secondly, for each sequence of the transition cover, the UIO of the state reached by that sequence is appended. HSI method generalizes the W method, and can be applied to partial FSMs. It first determines a transition cover and a set of harmonized state identifiers. Then, for each sequence of the transition cover, each of the sequence in the identifier for the state reached by that sequence is appended. The SPY method generalizes the HSI method; the main difference is that both the transition cover and the harmonized state identifiers are computed "on-the-fly", i.e., during the execution of the method, trying to minimize the number of test cases which are required to obtain complete test cases. For the HSI and SPY methods two versions are provided: the original

one and the auto-completed; the latter auto-completes the FSM before applying the original method. New methods can be incorporated in JPlavisFSM as plugins or be imported at runtime (provided that the compatible input/output formats are used) as well as providing more flexibility to the tool and the possibility to explore all the other features for the test set analysis.

Two different ways of evaluating a test cases set are provided:

- Mutation Analysis; and
- n-Completeness Analysis (Simão and Petrenko, 2010).

The Mutation Analysis, provided by JPlavisFSM, allows a test set to be evaluated and also to be enhanced. This enhancement can be achieved by expanding the test set with test cases targeted at undetected mutations. The mutant operators (Fabbri *et al.*, 1994) were designed according to the classes of faults detected on the FSM-based test models. When the user creates a test session, the mutants are automatically created and the user can access all the mutants to verify their statuses: *alive*, when the ongoing test fails to reveal the defect of the mutant; or *dead*, the other way around. This feature could help the user to upgrade test sets since this criterion guides the user to create tests that will run paths, which may contain defects. The n-Completeness analysis provides another way to verify the quality of a given test set, checking whether the test set is complete or not.

COFI METHODOLOGY

The CoFI methodology (Ambrosio, 2005) guides the functional testing activity. It was proposed considering the needs of space application's validation and therefore defines steps for creating test cases for software embedded on-board of satellites in a systematic way in order to get as much reusability in testing as possible. The main objective is to help the tester to define FSM-based models which represent the behavior of the SUT.

The SUT behavior's decomposition into FSMs starts with the identification of services, which can be a system function from the user's viewpoint. Each service shall be described by a set of FSMs that map different classes of behavior. This classification takes into account different kinds of input events:

- *Normal* (absence of faults);
- *Exceptions* (foreseen faults);
- *Unexpected inputs* (inputs occurring when they are not expected); and
- *Hardware faults*.

In short, the CoFI methodology consists of three steps: identification, modeling and test case generation. Figure 3 shows the three steps of CoFI methodology described hereafter.

(A) Identification

Based on a given specification, the tester has to understand properly the external behavior of the SUT, as in the black-box testing. The information to accomplish this step can be extracted from textual specification documents, requirements, use-cases, sequence-diagrams, and also from interviewing experts in the SUT if necessary. In this step, it shall be identified:

- *Services* that an user can recognize or execute in the SUT;
- *Events* and *actions*: it is necessary to identify all the possible events and actions (or outputs) that can be, respectively, commanded and trigged (or observed). The selected events and actions will be abstracted as inputs and outputs in the FSMs;
- *Control and observation points*: they are addresses or mechanisms to input data and to obtain responses from the SUT;
- *Physical faults*: faults occurring in hardware. They are to be considered whenever the SUT includes embedded software in a hardware and this software must have mechanisms to treat the faults;
- *Facilities and constraints*: the tools and theirs commands that supports the test execution against the SUT and the events that cannot be activated during the test execution.

(B) Modeling

In this step, for each *service*, the tester should develop four classes of behavior:

- *Normal*;
- *Specified exceptions*;
- *Sneak path;* and
- *Fault tolerance*.

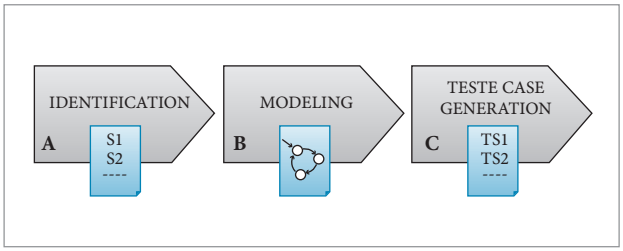

Figure 3. Main steps of the CoFI Methodology.

At least one FSM model shall be created representing the partial system behavior.

The *normal behavior model* defines the sequence of events that the SUT normally expects when it is executed. The tester has to identify the normal mode the SUT will be subordinated to and the events and actions expected for this case. For the *specified exceptions* model the tester has to recognize the exceptions mentioned in the documents, as timeouts and unknown commands and thus define the events expected in this context, called *exception events*. The *sneak path* model deals with the correct inputs arriving in wrong time. The objective is to predict the unexpected behaviors, i.e., to describe what the machine should do if an input occurs in a state where it is not defined. The tester can use the *normal* model as a basis, write it in tabular form (event × state) and fill the blank cells with the corresponding expected behavior. In some cases, there is no possible behavior to model, and then the tester evaluates the best option for the SUT. The transitions trigged by the *unexpected inputs* should be designed based on the normal model. Finally, the *fault tolerance* model shall map the possible physical faults when the SUT implements mechanisms to tolerate hardware faults. For each physical fault, the tester has to adapt the normal model including the fault events. At the end of the modeling step, the tester has a set of models representing the SUT's behavior, from which test cases may be automatically generated.

(C) Test Case Generation

This step can be supported by testing tools that automatically generate a set of test cases (also named test sequences). In the experiences about applying CoFI described in Ambrosio *et al.* (2007), Pontes *et al.* (2009), Anjos *et al.* (2011) and Mattiello-Francisco *et al.* (2013), the Condado tool was used to automate the test cases generation. In our work, the JPlavisFSM tool was used because it provides five different testing methods, including some variations that can be applied in partial FSM. Moreover, it has facilities to analyze the models' properties and the adequacy of test case sets.

CASE STUDY: ITASAT'S COMMUNICATION MODULE

We now discuss the case study we have carried out applying CoFI methodology and JPlavisFSM tool in a MBT approach for a real system. The SUT is the Communication Module's software of the ITASAT-1 satellite. The ITASAT Mission (Sato *et al.*, 2011) is part of a program funded by the Brazilian Space Agency (AEB) in the context of Action 4934 for Development and Launching of Small Technological Satellites developed by the *Instituto Tecnológico de Aeronáutica* (ITA) and other Brazilian universities. ITA is the responsible member for the project implementation and the *Instituto Nacional de Pesquisa Espacial* (INPE) provides technical consulting and laboratory infrastructure. The ITASAT-1 is a university satellite planned to execute experimental payloads namely, Digital Data Collection Transceiver, Heat Pipe Experiment, Micro Electrical Mechanical System for attitude determination and an Inter Satellite Link.

The satellite architecture is composed of five service subsystems: Mechanical Structure (MSS), Thermal Control (TCS), Electric Power (EPS), Attitude Control and Data Handling (ACDH) and Telemetry & Telecommand (TMTC); the last three subsystems are shown in Fig. 4. The functions of the ACDH are commanded by the on-board computer (OBC). The Communication Module (CM), located in the ACDH subsystem, is in charge of the communication between the OBC and the TMTC. Figure 4 highlights the Communication Module and indicates that it is the System under Test (SUT), the focus of this study.

The CM comprises not only two receivers and two transmitters in redundancy, but also the software in charge of receiving commands arriving from the ground stations and transmitting telemetry collected from satellite pieces of equipment to ground stations.

The CM Software performs critical functions so it has to work properly for the space mission success. Its functions are to receive the telecommands from ground stations and to transmit the telemetries collected by the satellite's payloads and sensors, to manage the payloads status and to turn off the pieces of equipment when a critical failure occurs leading the satellite into survival mode.

Considering that the CM is one of the most critical parts of this satellite, it was chosen as case study to experimentally evaluate the proposed approach. The three main steps that guided this study are illustrated in Fig.5.

(A) Identification

The first activity was to study the Requirements Document of ITASAT-1 (Table 1 illustrates some of such requirements)

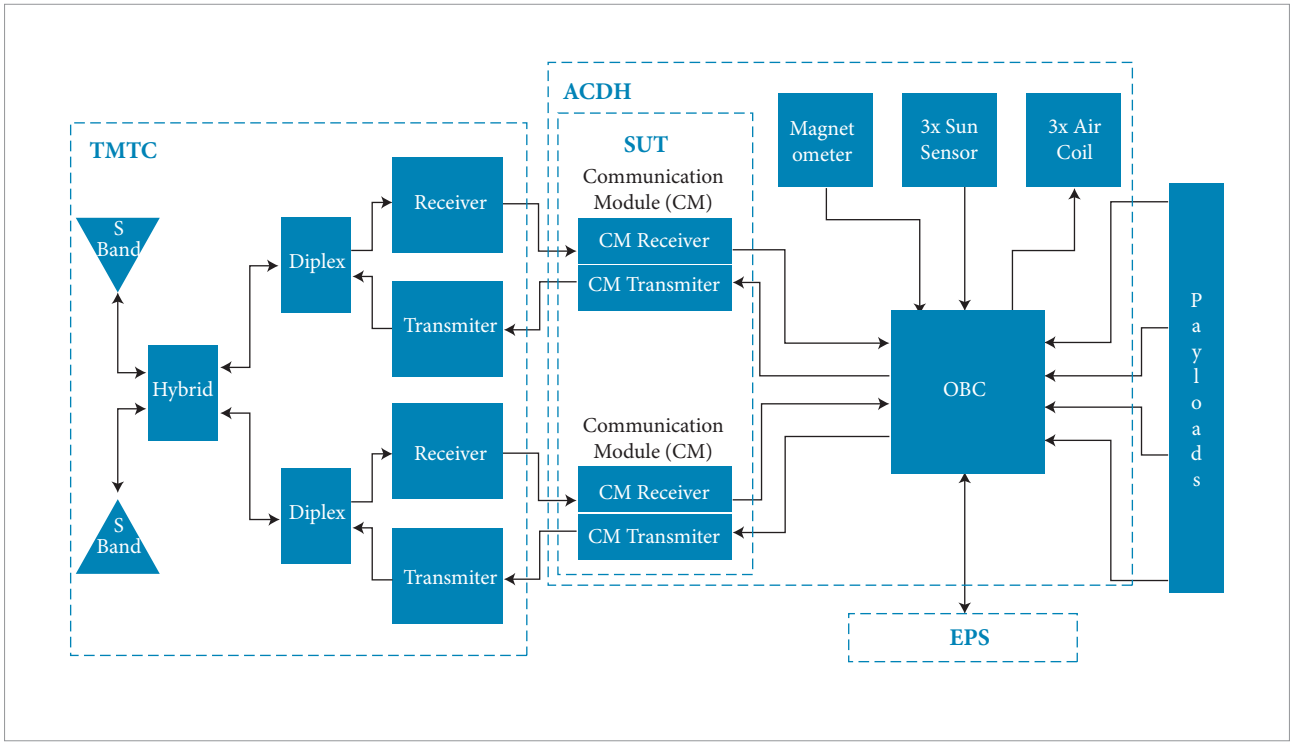

Figure 4. ITASAT-1 satellite architecture.

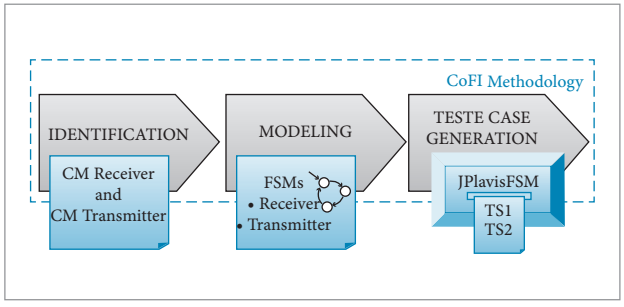

Figure 5. Development of the case study.

in order to understand the context and the behavior of the CM's software. The studies lasted 12 days, five to understand the ITASAT-1 satellite documentation; five to the CoFI guidelines and two to identify the main elements required by CoFI, namely, services, events, actions, faults, and so on.

We identified two *services* corresponding to the two main functions of the Communication Module: (S1) receive commands from ground and (S2) transmit the collected telemetries from on-board equipment. These functions are implemented respectively by the *CM Receiver Software* and by the *CM Transmitter Software*.

The *events* and *actions* extracted from the requirements are those naming the transitions of the FSMs illustrated in Figs. 6 and 7. The events and actions are abstract representations, i.e., they do not indicate a physical signal arriving. One example of a CM Receiver's event is *CmdOBCOK*, that indicates the presence of one command to be executed. One action is *StoreTC1wait*, which indicates that the just-arrived telecommand shall be stored on board. Other examples are *EndTimerB* as event and *StartTimerB* as action to be performed by the CM Transmitter. For the CM Receiver Software, 12 events and 13 outputs were identified, while for the CM Transmitter Software, 19 events and 12 outputs were identified. Specified *exceptions* were identified in seven requirements, which are listed in Table 1. Concerning the *physical faults*, although they can occur, the Communication Module Software is not in charge of dealing with them, therefore, for testing purposes, the *fault tolerance* models were not created.

(B) Modeling

It was observed that on adopting the CoFI methodology's guidelines the state explosion problem was avoided. First, we modelled the *normal* behavior of each service in the CM Receiver FSM (*R_N*) and the CM Transmitter FSM (*T_N*), as shown in Figs. 6 and 7, respectively.

For the FSM models used here, namely the Mealy machine (Mealy, 1955), all the transitions should be represented by a pair of input-output, which means that each event should be

associated with an action. Since in this software test some events were not associated with an observed output (or action), the output "*Terminate*" was used in the model to indicate the end of a transition. Another modeling condiction was made up to represent one transition requiring the occurrence of two events or transitions with two outputs. Figure 6 illustrates the *ACKandStartTimerC* output, which, in practice, indicates two actions: "send an *Acknoledge message*" and "start the *Timer C*".

Regarding the *specified exceptions*, the FSMs R_Ex13, R_Ex2, R_Ex4 and R_Ex5 map the exceptions of the CM Receiver, and T_Ex1 and T_Ex2 map those of the CM Transmitter. Table 1 matches the FSMs to the ITASAT-1's requirements they model. For sake of space, not all FSMs are shown in this paper. For more details see Pinheiro and Ambrosio (2013).

The CM Software has to handle timers which trigger timeout events needed for the communication timing. On the first analysis, to handle timers seemed an issue, because classical FSM does not treat time events. Therefore, these events were abstracted away in the model. The CoFI guides the definition of timers as external devices that are started by an action and produce an event to the FSM when the time is expired, indicating a timeout. For the *normal* behavior, the software only signalizes the start of a timer, as represented by *StartTimerC* and *StartTimerA* events. For the *specified exception* models, it was needed to represent events to indicate expired time, so we used the *EndTimerC* and *EndTimerA* events.

The first CM Receiver's FSMs were pointed out as non-deterministic by the JPlavisFSM, because the event *INT2TC* associated to *StoreTC* action occurred in several states. On further analysis of the real software's behavior we defined two abstractions to solve this problem:

- A finite buffer to store at most three TCs, called TC1, TC2 and TC3; and
- Two actions to differentiate TC stored and TC sent.

Then, to create several actions (outputs): *StoreTC1wait, StoreTC1send, StoredTC2wait*, and so on, instead of using only the *StoreTC* action.

Table 1. ITASAT-1 Requirements vs. Specified Exceptions Models.

Specified exceptions			
#	Requirement	Description	FSM
1	FRq11152000-12	"When the OBC does not send a request in TC seconds, the CM receiver finalizes the communication routine"	R_Ex13
2	FRq11152000-13	"The request of the OBC shall be one of the following: the transmission of TC or TC error log from CM receiver to OBC, or the execution of a (correct) command from OBC." Obs.: Terminate execution if the command is not OK.	R_Ex2
3	FRq11152000-16	"If the CPU does not send the acknowledge in TA seconds (value TBD), the CM receiver shall terminate the communication routine."	R_Ex13
4	FRq11152000-27	"The CM receiver shall check the first 16 bits, used for synchronization and packet addressing, and shall discard the TC packet if its address data does not match with the module address."	R_Ex4
5	FRq11152000-29	"If the verification described in Frq11152000-2 (The CM receiver module shall check the mod the repeated data and EDAC field) results in failure, the CM receiver shall store a TC error log (as specified in [AD3]) in the TC buffer"	R_Ex5
6	FRq11152000-52	"When any analog channel verified present unexpected results, the CM transmitter shall generate a pulse in the survival flag pin (which is read by the CM receiver).	T_Ex1
7	FRq11152000-58	"When the OBC does not send a request in TC seconds, the CM transmitter finalizes the communication routine."	T_Ex2

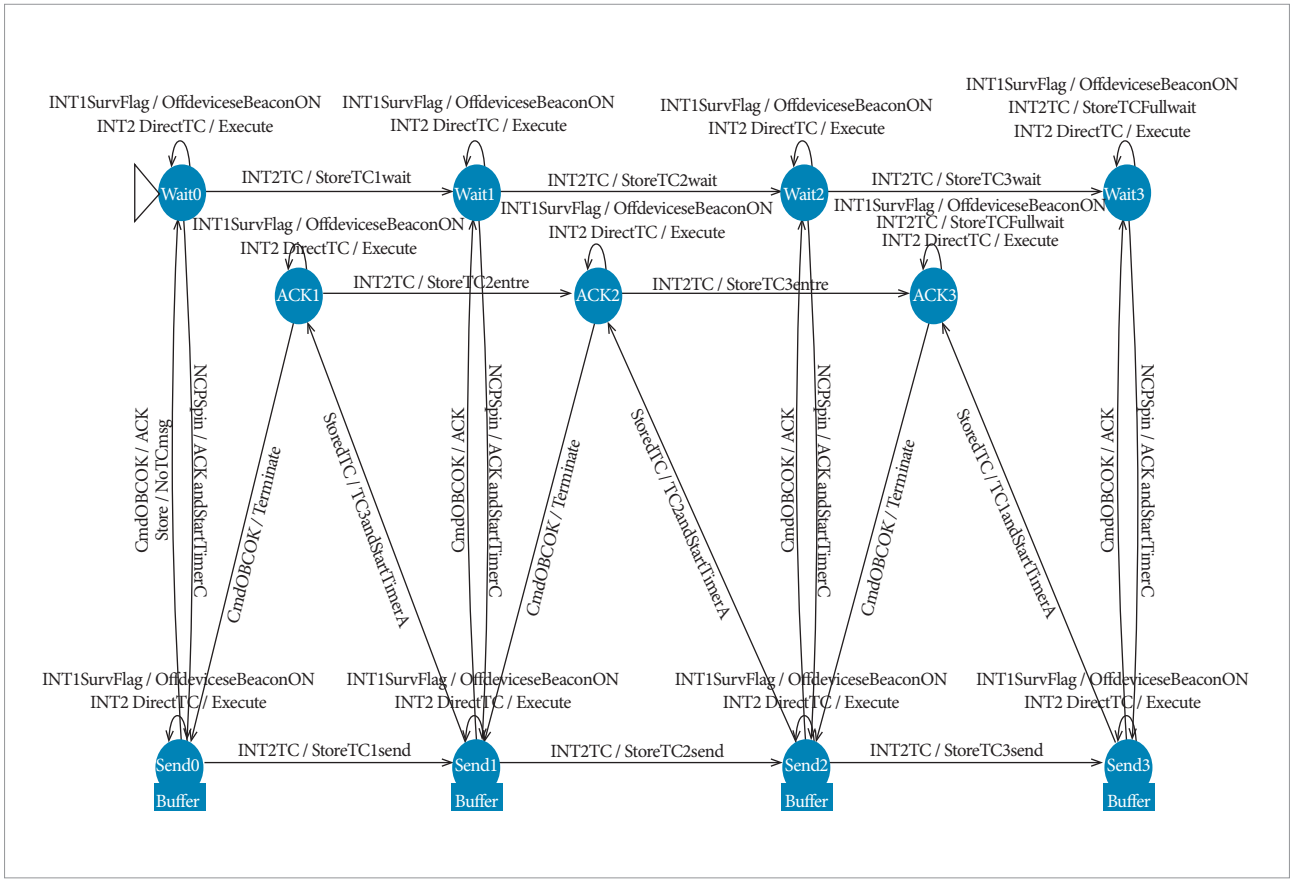

Figure 6. R_N: Finite State Machine representing the normal behavior of the CM Receiver.

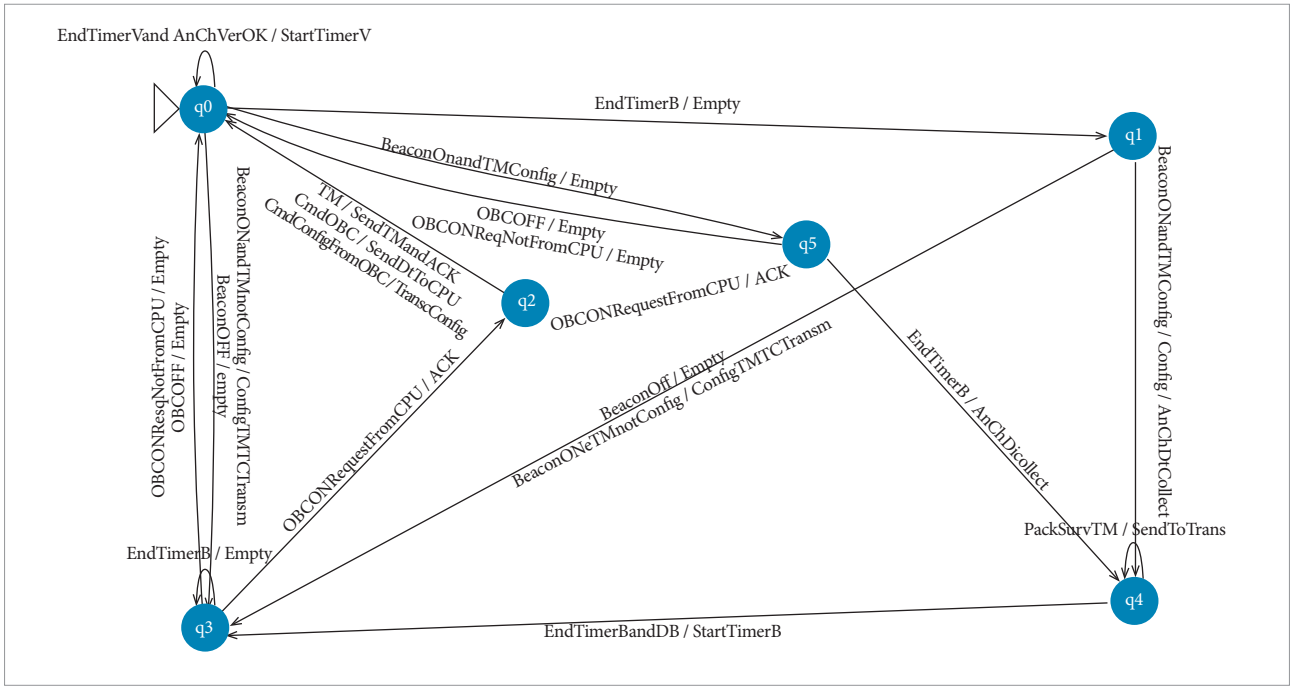

Figure 7. T_N: Finite State Machine representing the normal behavior of the CM Transmitter.

After concluding the normal behavior modeling, we analysed all events presented in the normal model, in order to create the *sneak path* models. However, because of the logic implemented in the hardware design construction, events never occur in an inappropriate state, so sneak path models were not designed. The *fault tolerant* models were not designed as well, because the CM is not in charge of dealing with the hardware faults. In the end of the modeling step, we had eight FSMs, as shown in Table 2.

The JPlavisFSM indicated incompleteness for all FSMs, consequently the traditional W method could not be applied. The completeness is a difficult property to be achieved when modelling the behavior of real systems, that is why the JPlavisFSM tool provides other methods, as HSI and SPY that do not require this propriety.

Summarizing, the modeling steps accounted for 16 days, being 8 days to create the normal and specified exception models including reviews and re-work in order to adequate the models to precisely represent the software behavior; and generate the test cases through the methods. Three adequacies were considered:

- Adequacy to the system – Do models represent the system?
- Adequacy to the FSM concepts and generation methods – How can one abstract the model to achieve the FSM needs?
- Adequacy to the final model – Have the final models achieved the system objectives and the methods needs?

(C) Test Case Generation

We applied all methods for the eight FSMs. The W method's applicability is restricted to complete, deterministic and initially connected FSMs. As the FSMs were not complete we used the JPlavisFSM's function to auto-complete it before using the W method. The auto-complete functionality adds all the missing transitions in a given state as self-loops, i.e., transition that starts and ends in the same state. Althought the auto-complete option leads to the generation of many test cases, which are not significant in practice, it allows the applicability of the W. One test case generated by W starting from the CM Receiver FSM normal behavior (Fig. 6) is given by the following sequence of inputs *INT2TC, StoredTC, NCPSpin,* and its corresponding expected output is *StoreTC1wait,TC1andStartTimerA,ACKandStartTimerC.*

Table 2. Summary of models design.

	FSM models by classes of behavior			
Services	Normal Behavior	Exception Specified	Sneak Path	Fault Tolerance
CM Receiver	1	4	0	0
CM Transmitter	1	2	0	0

The UIO method requires a *strongly connected* FSM, whereas the SPY and the HSI can be applied into a *non-completed* FSM. Thus, these methods have a wider applicability in comparison to the others; moreover, the test set generated by them is smaller than those genereted by W and UIO methods. On the other hand, it is possible to apply the UIO Method because it requires a *separation sequence* defined for each state. One example of a test case generated by UIO method starting from the CM Transmitter FSM is *BeaconONandTMnotConfig, OBCONRequestFromCPU, CmdOBC, EndTimerVAndAnChVerOK.*

The HSI and SPY are based on *separating sequences* by pair of states (see Background section). Thus, each pair of states must have a transition with an event in common with distinct outputs. In the FSMs of the CM Transmitter software no event occurs in more than one distinct state, so these methods were applied neither to T_N nor to T_EX1 and T_EX2. One test case generated by the SPY method is *INT2TC, NCPSpin, StoredTC, INT2DirectTC, INT2TC, INT2DirectTC, INT2TC* which starts with the same input as the example to W method, but explores a valid sequence of events.

In general, all methods of the JPlavisFSM were applied to the created models. Table 3 shows the test sets generated by all methods for each FSM. The '-' symbol indicates the method could not be applied and '*' symbol indicates the use of auto-complete functionality.

Particularlly, the original W method did not generate any test case for the *non-complete* FSM, so there is no column for it. The original W was executed only thanks to the auto-complete feature. This option increases the number of transitions, leading to a large number of test cases.

The UIO method generated smaller sets, but the method does not guarantee the generation of complete sets. The HSI and SPY were successfully applied to CM Receiver FSMs because there was a *separating sequence* that distinguished their states. For CM Transmitter FSMs, it was not possible

to use the HSI and SPY methods, therefore using the auto-complete feature they could be applied. The auto-complete feature was executed in CM Receiver FSMs only to show the increase in the number of test cases, caused by the extra transitions artificially included in the FSM to enable the application of the methods.

We have also explored the JPlavisFSM's functionality to calculate the *minimal test set* composed by test cases produced by all the methods. After creating a test set comprising all test cases, all the prefixed test-cases are discarded and the minimal test set is generated by the tool. The test sets shown in the HSI* and SPY* columns were not considered, since the original methods, HIS and SPY, could be applied.

Table 4 illustrates the number of test cases derived from each model (considering the total set and the minimal set), the number of mutants created to each model and the mutation score obtained with both sets of test case. The mutation scores for the minimal and the total sets were the same, since only prefixes are removed from the total set. The number of test cases (total and minimal sets) includes the tests generated by W*, UIO, HIS and SPY, but it does not include HIS* and SPY* to FSMs from CM Receiver.

All the methods (except UIO) generate complete test sets, but it is a theoretical concept which assumes that the implementation can be represented by an FSM with the same number of states of the specification.

In general, to apply FSM-based testing implies handling a very huge number of test cases, so the tester must either be prepared to automate the test execution or select and prioritize the test cases. In our approach, we suggest a test strategy that could be adopted: first is to select the smallest test set generated by original methods; then, if there is still enough time to the testing activity, other sets could be applyed to improve the system's reliability. In this case study, the SPY and UIO methods should be selected for CM Receiver FSMs and CM Transmitter FSMs, because the Mutation Score of these sets were 1.0 for each one, and they are complete sets as well.

The inputs and outputs represented in the models are abstracted to enable the creation of FSMs, this fact led to the generation of *abstract test suites*, so a post-processing of the test sets will be necessary before the test execution. This allows the test cases to be designed before defining some details of implementation and before the implementation is ready. Moreover, in the FSMs of the Comunication Module some events shall be replaced by a set of others, as the case of the *INT2DirectTC*, which corresponds to 23 different telecommands.

DISCUSSIONS AND RELATED WORKS

Various methods have been published to optimize the number of test cases to minimize the efforts required to obtain an acceptable level of software quality (Lai, 2002). However, these methods and test tools are yet rarely used in the software industry. Issues, such as 'What are the hindrances

Table 3. Number of tests generated by method for each Finite State Machine.

FSM	W*	UIO	HSI	HSI*	SPY	SPY*
R_N	234	48	38	78	22	40
R_Ex13	356	44	34	89	26	42
R_Ex2	201	37	27	67	21	40
R_Ex4	201	37	27	67	21	40
R_Ex5	201	37	27	67	19	37
T_N	316	21	-	184	-	72
T_Ex1	316	21	-	184	-	72
T_Ex2	335	19	-	158	-	61

Table 4. Results of the generated Test Sets.

FSM	Number of test cases		Mutation Analisys	
	Total Set	Minimal Set	# Mutants	Mutation Score
R_N	342	285	1852	1.0
R_Ex1	460	408	1720	1.0
R_Ex2	286	264	1320	1.0
R_Ex4	286	263	1357	1.0
R_Ex5	284	265	1357	1.0
T_N	593	432	775	1.0
T_Ex1	593	432	775	1.0
T_Ex2	573	454	591	1.0

to the adoption of the generation methods based on FSM in the industry?' or 'What is the gap between academic and industrial testing methods?' have been investigated in order to ease the difficulties of employing FSMs in the real context. In this section we discuss these questions in general speaking and concerning our approach.

The fault domain, introduced in Background section, is defined for general purpose FSM. That fault domain provides all possible faults that can happen in the model. However, it may be interesting to adapt the model to the real scenario that should be tested. In Srivastava and Singh (2009) a fault model based on FSM to embedded systems was proposed. The fault model provides three kinds of errors that could occur among the connections between the hardware and software components: *unconnected outputs*, when the implementation should activate an output but does not activate, in other words, there is no output after a transition happens; *unconnected inputs*, when an input event has no behavior effect, there is no transition to another state as expected; and *redirected inputs*, which occur when an input is redirected to an incorrect state and generates a wrong behavior. The requirements for the construction of the test set were defined according to the new fault domain. The main focus of Srivastava and Singh is on the definition of the domain applied to embedded systems; and a testing generation method is not applied.

In Santiago *et al.* (2008) a test environment that supports the generation of test sequences based on Statecharts and FSMs is presented. The simplicity of the model is considered the main advantage of employing FSMs as a technique for modeling reactive systems. Methods such as W and UIO were applied in the context of embedded systems.

Although the applicability of MBT has been widely investigated, there are some reasons for reluctance in adopting academic methods for FSM-based testing. We have evaluated some of the reasons presented in Lai and Leung (1995) and Lai (2002), such as *feasibility*, that concerns the use of the testing methods in real cases; *extreme formalism*, that gives the impression of being too academic; *need for training* or education in this area; and *resistance to changing*, because it is not necessary the use of formal methods to do tests, according to the current point of view of many industries acting in the satellite-based software sector.

The difficulty in using MBT starts with the modeling phase. Identifying the system inputs and outputs, as well as the system behaviors, is not a trivial task. A broad knowledge about the system and about FSM is necessary to understand how a non-formal specification could be transformed into a formal one. In our approach, CoFI methodology helped to deal with these difficulties. The tester is guided, step by step, to model the system behavior into several simple FSMs. Moreover, the JPlavisFSM tool facilitated fitting the models to the testing methods, automating the test cases generations, producing test sets' metrics allowing the tester to choose the best testing set.

The tester must yet interpret the generated test sets due to the abstraction of the model, though the practicality of MBT was improved by CoFI and JPlavisFSM. The formalism of the FSM-based testing methods is transparent to the JPlavisFSM tool user. The analysis of FSM properties is automatically generated.

Some validations with the use of FSM-based testing techniques and CoFI methodology have been conducted (Anjos *et al.*, 2011; Pontes *et al.*, 2012; Mattiello-Francisco *et al.*, 2013). In such cases the Condado (or ConData) (Martins *et al.*, 1999) was used as a tool to automatically generate test cases. Condado tool does not have neither the facilities to FSM structural analysis nor different methods to be chosen, however the single FSM property it requires is the FSM be connected. In the approach presented here, the FSMs have more elaborated structures, so the facilities provided by the JPlavisFSM tool allowed improving the application of the CoFI. Besides that, it was applied to real embedded software of satellite application.

Concerning the training aspect, JPlavisFSM tool can be used to teach how the generation methods work. The tool has the option to import extra methods. The user may implement his/her own method and apply the n-Complete tool to analyze if the method is correct. The JPlavisFSM may be used to support the modeling phase too, since the GUI is simple, clean and it provides functionalities for analyzing FSMs. The usability improvements provided by JPlavisFSM and the guidelines of CoFI methodology were tentative of reducing the gap between the good results obtained in academia with testing methods and its use in practical cases of industry as discussed in Lai and Leung (1995).

CONCLUSION

The gap between academic research and industrial practice is still large. Thus, it is important to analyze the performance of testing activity supported by FSMs in real systems, once FSM-based methods are richly explored in the academic context.

This paper has reported on the applicability of the FSM-based testing in the real, developed Communication Module software of the ITASAT university satellite. As testing is a costly activity, automating the generation of test cases can help reduce the costs. In this sense, MBT is an approach that derives test cases from a formal model built to support the testing activity. On the other hand, real systems like ITASAT software should be tested taking into account requirements of timing and reactions to events. In this context, the systematic approach of CoFI methodology supported by the formalism implemented by JPlavisFSM could be an alternative to test the system. We have observed restrictions and benefits on applying this approach. Some observed limitations, which should be issues for new investigations, are:

- The modeling phase is not trivial and some knowledge about FSM is required to start the process. The tester has to learn the basic concepts and structural properties of FSM;
- The applicability of the generation methods is highly dependent on structural properties of FSM generated in modeling phase. The tester has to know the basic theory about the generation methods;
- The abstraction of the system is required to build models, so the test set generated is an abstract set of inputs and outputs, which needs to be interpreted before being executed. So, the post-processing of test sets could aggregate an extra cost to the test execution phase.

The observed benefits were:

- The CoFI methodology guides the modeling phase, which helps the inexperienced testers start using the FSM technique;
- The FSM-based test generation was automated leading to cost reduction of the test activity;
- The JPlavisFSM tool eliminates the need of profound theoretical knowledge about FSM structural properties and testing generation methods.

The MBT still has some practical limitation, but the initiatives that were made in academia helps to reduce them. The definition of new generation methods which do not require properties, such as FSM minimality, has to be explored. At the same time, the industry needs to step forward on the use of formal methods. These initiatives will improve the efficiency and effectiveness of testing activity.

As future work, we envision the following. The generated test sets must be post-processed and executed against the Communication Module implementation of ITASAT and the complexity of these activities should be analyzed against the obtained results.

ACKNOWLEDGMENTS

The authors would like to thank professors Emília Villani, David Fernandez and Wilson Yamagutti for the opportunity of applying the proposed approach in the ITASAT project. The authors would also like to thank the financial support of FAPESP, CNPq and CAPES. The authors are very thankful to reviewers for their useful comments.

REFERENCES

Alencar, W.A.F., 2013, "Model checking aplicado a software embarcado crítico do satélite universitário ITASAT", Dissertação de mestrado, ITA, São José dos Campos, Brazil.

Anjos, J.S., Gripp, J., Pontes, R. and Villani, E., 2011, "Applying the CoFI testing methodology to a multifunctional robot end-effector", São José dos Campos, SP, Brazil.

Ambrosio, A.M., Martins, E., Vijaykumar, N.L. and Carvalho, S.V., 2005, "Systematic generation of test and fault cases for space application validation", In Proceedings of the 9th ESA Data System in Aerospace (DASIA), Edinburgh, Scotland. Noordwijk: ESA Publications.

Ambrosio, A.M., 2005, "CoFI: uma abordagem combinando teste de conformidade e injeção de falhas para validação de software em Aplicações Espaciais", Tese de doutorado, INPE-13264-TDI/1031. 206p.

Ambrosio, A.M., Mattiello-Francisco, F., Santiago, V.A., Silva, W.P. and Martins, E., 2007, "Designing fault injection experiments using state-based model to test a space software", Third Latin-American Symposium on Dependable Computing (LADC), Morelia, México. Lecture Notes in Computer Science (LNCS) series. Springer Editors: Bondavali, A.; Brasileiro, F.; Rajsbaum, S. Springer, Berlin. pp. 170-178. doi: 10.1007/978-3-540-75294-3_13.

Chow, T.S., 1978, "Testing software design modeled by finite-state machines," IEEE Transactions on Software Engineering, Vol. 4, No. 3, pp. 178–187. doi: 10.1109/TSE.1978.231496.

Delamaro, M.E., Maldonado, J.C. and Jino, M., 2007, "Introdução ao Teste de Software", Elsevier.

Fabbri, S.C.P.F., Maldonado, J.C., Masiero, P.C. and Delamaro, M.E., 1994, "Mutation analysis testing for finite state machines", In Fifth International Symposium on Software Reliability Engineering, Monterey, California, USA, pp. 220-229.

Fujiwara, S., Bochmann, G.Von, Khendek, F., Amalou, M. and Ghedamsi, A., 1991, "Test selection based on finite state models", IEEE Transactions on Software Engineering, Vol. 17, No. 6, pp. 591-603. doi: 10.1109/32.87284.

Gill, A., 1962, "Introduction to the theory of finite-state machines", McGraw-Hill, New York.

Hierons, R.M. and Ural, H., 2010, "Generating a checking sequence with a minimum number of reset transitions", Automated Software Engineering, Vol. 17, No. 3, pp. 217-250. doi: 10.1007/s10515-009-0061-0.

Lai, R. and Leung, W., 1995, "Industrial and academic protocol testing: the gap and the means of convergence", Computer Network ISDN System, Vol. 27, No. 4, pp. 537–547. doi: 10.1016/0169-7552(93)E0110-Z.

Lai, R., 2002, "A survey of communication protocol testing", The Journal of Systems and Software, No. 62, pp. 21-46.

Martins, E., Sabião, S.B. and Ambrosio, A.M., 1999, "ConData: a Tool for Automating Specification-based Test Case Generation for Communication Systems", Software Quality Journal, Vol. 8, No.4, pp. 303-319.

Mattiello-Francisco, M.F., Ambrosio, A.M., Villani, E., Martins, E., Dutra, T. and Coelho, B., 2013, "An experience of the technology transfer of CoFI methodology to automotive domain", LADC, Industrial track. BDBComp Biblioteca Digital.

Mealy, G.H., 1955, "A Method for Synthesizing Sequential Circuits". Bell System Technical Journal, Vol. 34, No. 5, pp. 1045-1079.

Moore, E.F., 1956, "Gedanken–experiments on sequential machines", in Automata Studies. Princeton University Press, pp. 129-153.

Morais, M.H.E and Ambrosio, A.M., 2010, "A new model-based approach for analysis and refinement of requirement specification to space operations", In Proceedings of the 10th Conference on Space Operations. Huntsville, Alabama, USA. SpaceOps 2010. American Institute of Aeronautics and Astronautics (AIAA). doi: 10.2514/6.2010-2231.

Morais, M.H.E., 2011, "Uma abordagem para a melhoria da qualidade de requisitos baseada em modelos de estados", Dissertação de mestrado. INPE, 2011. São José dos Campos.

Pedrosa, L. and Moura, A., 2010, "Generalized partial test case generation method", In 4th International Conference on Secure Software Integration and Reliability Improvement Companion (SSIRI-C 2010). Washington, DC, USA: IEEE Computer Society, pp. 70-77.

Petrenko, A., Yevtushenko, N., Lebedev, A. and Das, A., 1993, "Nondeterministic state machines in protocol conformance testing", in Proceedings of the 6th International Workshop on Protocol Test systems VI (IFIP TC6/WG6.1). Amsterdam, The Netherlands, The Netherlands: North-Holland Publishing Co., pp. 363-378.

Pinheiro, A.C. and Ambrosio, A.M., 2013, "Modelagem do Módulo de Comunicação do Satélite ITASAT – Segundo a metodologia CoFI", Relatório de Pesquisa, sid.inpe.br/mtc-m19/2013/10.04.19.58-NTE. INPE, São José dos Campos. Retrieved in March, 19th 2014, from http://urlib.net/8JMKD3MGP7W/3EUF338

Pinheiro, A.C., 2012, "Subsídios para a aplicação de métodos de geração de casos de testes baseados em máquinas de estados", Dissertação de mestrado. ICMC. USP. São Carlos, Brazil.

Pontes, R.P., Morais, M.H.E., Véras, P.C., Ambrosio, A.M. and Villani, E., 2009, "A comparative analysis of two verification techniques for DEDS: Model checking versus model-based testing", In 4th IFAC Workshop on Discrete Event System Design (DEDS), Valencia, Spain, pp. 70-75.

Pontes, P.R., Véras, P.C., Ambrosio, A.M. and Villani, E., 2012, "Contributions of model checking and CoFI methodology to the development of space embedded software", Empirical Software Engineering – An International Journal, Vol. 10, No. 2, Springer. Online FirstTM, doi: 10.1007/s10664-012-9215-y - ISSN 1382-3256.

Romero, A.G., Ambrosio, A.M. and Souza, M.L.O, 2012, "Finite state-machine verification applied to hybrid systems", 21st SAE BRASIL International Congress and Exhibition, Expo-Center Norte, São Paulo, Brazil. doi:10.4271/2012-36-0429.

Sabnani, K. and Dahbura, A., 1988, "A protocol test generation procedure", Computer Networks and ISDN Systems, Vol. 15, No. 4, pp. 285-297. doi: 10.1016/0169-7552(88)90064-5.

Santiago, V., Vijaykumar, N.L., Guimarães, D., Amaral, A.S. and Ferreira, E., 2008, "An environment for automated test case generation from statechart-based and finite state machine-based behavioral models", In Proceedings of the 2008 IEEE International Conference on Software Testing Verification and Validation Workshop. Washington, DC, USA: IEEE Computer Society, pp. 63-72.

Sato, L.S., Saotome, O., Timm, C., Fernades, D. and Yamaguti, W., 2011, "Itasat-1: Brazilian university microsatellite for payload test and validation in low earth orbit", In Proceeding of the 8th Symposium on Small satellites for Eath Observation, Berlin, Germany.

Simão, A.S., Ambrosio, A.M., Fabbri, S.C.P.F., Amaral, A.S.M.S., Martins, E. and Maldonado, J. C., 2005, "Plavis/fsm: an environment to integrate FSM-based testing tools". In Simpósio Brasileiro de Engenharia de Software - Sessão de Ferramentas. Universidade Federal de Uberlândia. Uberlândia, Brazil.

Simão, A.S., Petrenko, A. and Yevtushenko, N., 2009, "Generating reduced tests for fsms with extra states," In Proceedings of the 21st IFIP WG 6.1 International Conference on Testing of Software and Communication Systems and 9th International FATES Workshop, ser. TESTCOM '09/ FATES '09. Berlin, Heidelberg: Springer-Verlag, pp. 129-145.

Simão, A.S. and Petrenko, A., 2010, "Checking completeness of tests for finite state machines", IEEE Transactions on Computers, Vol. 59, No 8, pp. 1023-1032. doi:10.1109/TC.2010.17.

Srivastava, S. and Singh, A., 2009, "Testing of embedded system using fault modeling", pp. 177-180. doi:10.1109/ ELECTRO.2009.5441142.

Utting, M. and Legeard, B., 2007, "Practical model-based testing: A tools approach", 1st ed. Morgan Kaufmann.

Yin, Y., Li, Z. and Liu, B., 2010, "Real-time embedded software test case generation based on time-extended EFSM: A case study", In Proceedings of 2nd WASE International Conference on Information Engineering (ICIE'10), pp. 272-275.

Vuong, S., Chan, W. and Ito, M., 1989, "The UIOv-Method for protocol test sequence generation", In Proceedings of the 2nd International Workshop Protocol Test Systems, pp. 161-175.