An Approach for Verification of a Satellite Simulator – an evolving system

Paulo Diego Barbosa da Silva
Emília Villani
Brazilian Aeronautics Institute of Technology, ITA
São José dos Campos, São Paulo, Brazil
paulodiego1@gmail.com, evillani@ita.br

Ana Maria Ambrosio
Denise Rotondi Azevedo
Brazilian National Institute for Space Research, INPE
São José dos Campos, São Paulo, Brazil
ana.ambrosio@inpe.br, denise.rotondi@inpe.br

Abstract— **Satellite simulators are developed in the context of a space mission lifecycle to represent the real behavior of a satellite during operation and may be used for different purposes. To attend a particular purpose new functions are added or modified according to the mission phase needs, requiring models re-adaptation in a system evolving concept. The process of verification of satellite simulator software requires high-efficiency in accomplishing realistic functional and behavioral requirements. Based on the complex set of requirements the satellite behavior is represented in the simulator through software models specified by tables of cause-effect rules. Considering that the Satellite Simulator is an evolving systems and it needs to assure that the logic implemented in the simulator conforms to the requirements, the manual verification process becomes impracticable, therefore demanding a compatible verification approach. The approach suggested here unifies two techniques Conformance and Fault Inject (CoFI), constructed on Model-Based Testing and Model Checking added to a method so that it can translate the tables of cause-effect rules into finite state machines. This paper presents the verification approach illustrating it with the Data Collection Subsystem (DCS) model of the CBERS satellite simulator being developed at National Institute for Space Research (INPE).**

Keywords—Satellite simulator; Test automation; Verification; Validation; Formal methods; Model checking; Model based-testing.

I. Introduction

As part of a space mission, an operational satellite simulator plays an important role as a tool for training operators, for understanding satellite failures in order to solve them in scenarios as real as possible and for validating operational procedures before sending them to the real satellite.

Operational Simulators are considered as evolving systems once they are built using an infrastructure over which models are added to mimic satellite subsystems, space dynamics and ground stations. In general, satellite simulators are developed in the context of a mission lifecycle, with a particular purpose. Nevertheless, the purpose may change according to the mission phase needs, requiring models re-adaptation in a system evolving concept [1].

At the National Institute for Space Research (INPE) the development of operational simulators has been occurring since 1991; the first satellite simulator was created for the SCD1 (Data Colleting Satellite) part of SCD's family and has continued for families of other satellites as China-Brazil Earth Resources Satellite (CBERS) and scientific French-Brazilian Micro-satellite [2].

The solution adopted for the CBERS Satellite operational simulator was to separate a common infrastructure from the models that represent the satellite, thus, enabling the infrastructure reuse in several other missions. Concerning the satellite specification for the simulator, the behavior of each subsystem is currently documented in cause-effect tables in a standardized and repeatable way. That was the best solution found to represent the information understandable by engineers of the different areas need to build a satellite: orbit and attitude control, mechanics, thermal, computing, electronic, etc. and for the software development team [3]. This representation demands an efficient verification process to guarantee that all the functional and behavioral requirements are present in the final software.

The current V&V (Verification and Validation) process being applied to the simulator subsystems models is *ad hoc* and manually executed; therefore, it is cost and time-consuming and has low efficiency for requirements covering verification. So, the need for an approach that guarantees the conformance between the functional requirements and the satellite software implementation is mandatory. This approach should have a development-in-progress and reusable characteristic so that it can fit the necessity of constant evolution of the satellite simulator software.

The adoption of a particular standard to represent the CBERS 3&4 satellite simulator functional and behavioral requirements through tables of cause-effect rules has facilitated the models understanding, for both, the software development team responsible for implementing the subsystem behavior specified by the cause-effect tables and the verification team.

In this paper we present the exercise of an approach for verification of models of a satellite simulator with evolutionary features that is yet under development. In short, the approach consists of translating cause-effect rules into Finite State Machine (FSM) combining Model Based-Testing according to CoFI methodology [4] and Model Checking [5]. From a set of FSMs the approach will make possible the automatic creation of a concise abstract set of test cases, which may be re-used and evolve to ensure conformance between the satellite simulator software and the requirements that describe its behavior. The approach is illustrated with the Data Collection Subsystem (DCS), a model of the CBERS satellite.

This paper is organized as follows: Section II describes the complete approach adopted in this work. Section III describes the techniques and tools used, presents the case study and the proposed translation method. Section IV discusses the application of the approach to the DCS subsystem and its results. Section V presents the related works and finally, Section VI presents the conclusion and final considerations.

II. The Approach of Verification adopted in the work

A. *The Approach Overview*

This section presents an overview of the verification approach of the satellite subsystem model based on cause-effect rules. This approach considers the following activities: (i) *translate* the cause-effect tables into FSMs, via our Translation Method; (ii) *apply* the CoFI methodology creating a CoFI set of FSMs, (iii) *create* CoFI-Total FSM, (iv) *check* the CoFI-Total FSM properties with the UPPAAL tool [6], (v) *update* the CoFI Set models, (vi) automatically *generate* a set of Abstract Test Cases applying the updated CoFI Set of FSMs to MME [7] and Condado [8] Tools, (vii) *transform* the Set of test cases from abstract to concrete, (viii) *execute* the concrete test cases against the Model implementation in the Satellite Simulator and (ix) *compare* the outputs observed during the test execution with the expected outputs foreseen in the abstract test cases. A schematic view of verification process is depicted in the Figure 1:

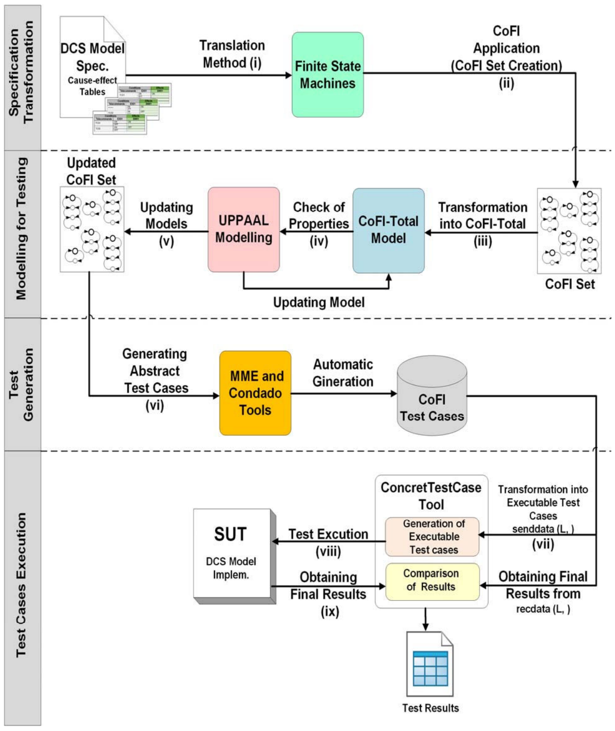

Fig. 1. Overview of the V&V approach.

B. *Activities Definition*

The following is a brief description of each approach activities:

(i) Translate the cause-effect tables into FSMs: This step consist of the application of a method developed to translate the system requirements, arranged in cause-effect tables, into FSMs. Here the DCS model specification was used to exemplify the method application.

(ii) Apply the CoFI methodology: The second step describes the CoFI Methodology application which results in the creation of four classes of models that helped to cover the entire system requirements behavior described in tables.

(iii) Create CoFI-Total FSM: Aiming to represent the complete SUT in one FSM all FSMs produced from the application of CoFI Methodology were combined in a single Timed Automata that represents the complete behavior. This single automata is called here as CoFI-Total.

(iv) Check of properties of CoFI-Total model with UPPAAL tool: In this step is performed the complete check of properties of the CoFI-Total model using the UPPAAL tool.

(v) Update the CoFI set's models: The results of the properties that were analyzed in the previous step became feedback for CoFI Set and for CoFI-Total FSM model. This step may be done several times.

(vi) Automatically generates abstract test cases: This activity intended to introduce the FSMs from the CoFI Set in MME tool and then generate the abstract test cases using the Condado tool.

(vii) Transform the test cases from abstract to concrete using the ConcreteTestCase tool: In this step, ConcreteTestCase tool was used to transform the abstract test cases in concrete, i.e. test cases able to be executed by the System under Test (SUT).

(viii) Execute test case on SUT: In this step, the concrete test cases are executed in the DCS subsystem implemented as part of the satellite simulator.

(ix) Compare the results: Obtain the final test results with ConcreteTestCase tool. This tool performs the comparison of the outputs generated by Condado and the outputs produced by the simulator, and then gives the verdict of the resulting tests.

III. Techniques and Tools, Case Study and the Translation Method

This section describes the approached techniques and the tools used to support our proposal as well as the case study that motivated this research and the Translation Method that allowed to apply the existing tools and the model-base testing concepts to an evolving system.

3.1 Techniques and Tools Adopted

The two techniques adopted as guide for the development of this work and the four tools used for supporting them are described in the sequence.

A. *CoFI Methodology and MME and CONDADO Tools*

CoFI (Conformance and Fault Injection) is a model-based testing methodology (MBT) developed to address conformance and fault injection testing systematization of embedded software in space missions [9]. CoFI guides the construction of a set of Finite State Machines (FSMs) representing the expected behavior of services provided in requirement specification. From FSM test cases may be automatically generated. The methodology advises the tester to represent the system behavior into different and partials FSMs avoiding the explosion of states and consequently the number of test cases. Classes of models should be established for normal behavior, specified exceptions, sneak paths and fault tolerance. These models cover possible scenarios that can occur in a space system as found in.

The MME tool enable on to edit FSMs, while the Condado tool automatic generates test cases using the *Transition Tour* algorithm [10]. Both tools were developed in the context of the Project ATIFS in a partnership between the Campinas University and INPE [11].

B. *Model Checking and UPPAAL Tool*

Model checking is a prominent formal verification technique for assessing functional properties of information and communication systems that requires a model of the system under consideration, modelling a system with a desired property and systematically checks whether the given model satisfies this property. Model checking is a technique that allows verifying finite state concurrent systems [12].

The UPPAAL model checking is a computational tool that allows the modeling of timed automata. This tool was developed in a partnership between two universities, the Uppsala University, at Sweden and the Aalborg University, at Denmark. UPPAAL Tool allows the modeling, system simulation and the properties verification in the implemented models. In the verification process performed by UPPAAL, the CTL language (Computational Tree Logic) for properties specification was used. When a property is false, the tool provides traceability path to achieve the state where the property is not true [13].

C. *The ConcreteTestCase Tool*

The ConcreteTestCase is a tool developed in the research context of this work in order to help in the process of automating the test execution of the CBERS simulator. The tool has two main functions: (1) translating the test cases to a format suitable for execution in the CBERS simulator and (2) comparing the simulator outputs with the correlated test case generated by Condado tool. Details of these two functions are given below:

1) The generation of executable test cases

In the generation of executable test cases for the simulator, the input is a file produced by Condado, named ".seq" that contains the set of abstract test cases. Each test case described in this file is defined with an input event "senddata (L, xx)" and an output action "recdata (L, yy)", the xx parameter of the senddata means the input event and yy in recdata means an output of a transition in the FSM. The ConcreteTestCase tool obtain xx from "Senddata (L, xx)" to generate an executable test sequence, as illustrated in. Figure 2.

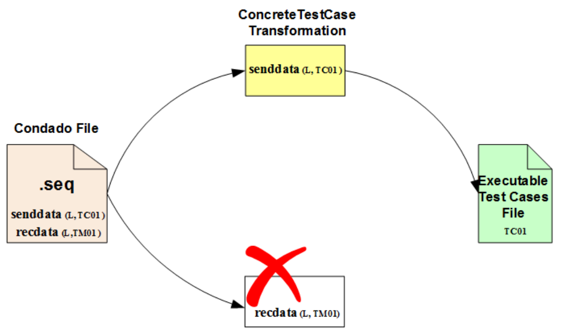

Fig. 2. Generation of executable test cases.

2) Comparison of the Test Results

In order to obtain the verdict of the final results for the tests applied in the simulator, the ConcreteTestCase tool works with three separated input files. The first one is the ".seq" file, using the recdata (L, yy) the tool compares it with the merged simulator output files. The simulator output files list the sequence of input events while the other lists the sequence of output actions. If a match occurs between the output action presented in the Condado file and the merged simulator output files, the tool lists the results as "OK", it means that there is Conformance between the outputs, otherwise the result is "N-OK", which means faults. Figure 3 illustrates the artifacts handled by this function:

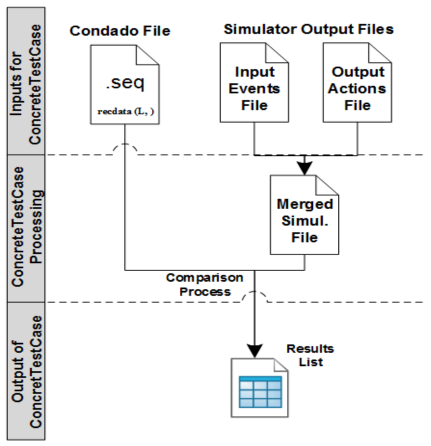

Fig. 3. The schematic for obtaining the test veredict.

3.2 The Case Study and the Translation Method

In order to better describe this activity was first exposed the Case Study adopted in this work, then a description of the steps of the Translation method.

A. *Case Study: The DCS Subsystem*

The case study adopted to illustrate the approach was the logical behavioral specification of the Data Collection Subsystem (DCS). The DCS comprises six pieces of equipment: UHF-band and S-band antenna, diplexer, S-band transponder, UHF transmitter and the oscillator. It is designed to receive signals from the Data Collecting Platforms (DCP) and transmit them to the Ground Station in the S and UHF communication bands. In general, the DCS provides the following functions: (a) Receive signals from the DCPs. (b) Convert the signals to an intermediate frequency band (c) Modulate the signals for S-band and UHF-band antennas. (d) Transmit the modulated signals to the ground station at S-band and (e) UHF-band antennas.

Among the devices that comprise the subsystem there is a set of ON/OFF switches, which allows to change the operating channel, activate or deactivate redundant equipment, turn on and off the equipment. The switches positon combination defines the subsystem behavior depicted in the subsystem specification document. The subsystem behavior specification is given through a set of rules represented in different cause-effect tables. These cause-effect rules can be understood as follows.

Telecommands (TCs) that change the DCS switches (SWs) position are the only kind of input events considered in the subsystem. The combination of switches defines a Working State (WKs) and consequently the combination of Working States defines the Equipment Operating Mode (OMeq). Finally, the combination of the equipment operating modes gives the DCS Operating Mode.

When a SW is changed according to the sequence of TCs, the corresponding telemetries (TMs) and Power Consumption values (PWs) are updated. Figure 4 shows how a sent TC changes a SW, the TM and correlated PW.

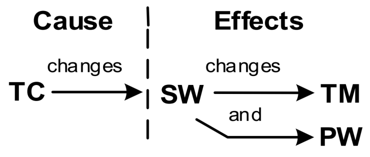

Fig. 4. The relation between input event and output action in DCS specification.

Table 1 shows how the TC01 and TC02 interact with a SW01: turning ON and turning OFF the switch affect its state.

TABLE I. TELECOMMANDS VS SWITCH CONFIGURATIONS.

Conditions	Effects
Telecommands	**SW01**
TC01	ON
TC02	OFF

Table 2 shows how the combination between the switches SW01 and SW02 affects the working states of the transponder in channel 1 (WKA1_01) and the transponder in channel 2 (WKA2_01). If the switch SW01 is ON and SW02 is in Channel 1 then, the equipment working states WKA1_01 will be ON and WKA2_01 will be OFF.

TABLE II. SWITCH CONFIGURATIONS VS. WORKING STATES.

Conditions		Effects	
SW01	**SW02**	**WKA1_01 Transponder 1**	**WKA2_01 Transponder 2**
ON	CH1	ON	OFF
	CH2	OFF	ON
OFF	CH1	OFF	OFF
	CH2		
X	X	OFF	OFF

Table 3 presents the manner as the combination between the transponder (WKA1_01) and the oscillator (WKA1_02) provides the changes in the equipment OMA1 (the operating mode for the transponder in the channel 1).

TABLE III. WORKING STATES VS. EQUIPMENT OPERATING MODES.

Preconditions		Effects
WKA1_01 Transponder 1	**WKA1_02 Transponder 2**	**OMA1**
ON	ON	Operating
	OFF	No oscillator
OFF	ON	Stand-by
	OFF	Powered off

3.3 The Translation Method

From the given specification of the subsystem behavior in cause-effect rules to the use of the testing techniques and tools there is a great challenge, to develop a method to translate the subsystem behavior models in FSM that will serve as input to the application of the complete proposed verification approach.

Although this method is still under development, its first execution in the DCS case study has proved to be effective. The method has been divided into eight steps that serve as guidance for obtaining FSMs from the tables of cause-effect rules present in the behavioral specification of a satellite simulator model. The steps described below use the DCS case study for the sake of a better understanding.

Step 1 - The first step is to identify and understand what are the OMs (DCS Operating Modes) and the state of the switches in each OM are.

Step 2 - Step two aims to analyze how each input event (TC) enables and disables the SWs and thus changing the working states (WKs) of each piece of equipment that composes the subsystem, and thus modifying TMs, PWs and OMs. The DCS Model Technical Specification [14] gives us the definition of the events (TCs), different valid outputs and the rules of how they are related.

Step 3 - The third step defines the valid initial sequence of activation for a **complete operation**[1] of the subsystem. This sequence is composed of TCs and respective TMs as specified in [15], [16].

Step 4 - This step lists all valid sequences of operation, but only those that are accepted as standard procedure of the subsystem, i.e. a partial activation of the system operations when not all of the subsystem equipment is working.

Step 5 – In this step we draw the first FSM for the main set of equipment, taking into account the sequence of events (TCs) and states (active devices) established in step 1 and 3. (For DCS, we considered only the channel 1)

Step 6 – This step is for drawing a FSM for a set of redundant equipment (channel 2).

Step 7 – The seventh step represents a FSM to make the connection between the main and redundant FSMs, using the TCs for that.

Step 8 – For stablishing the degraded states is necessary combine the telecommands, in different sequences from those present in the DCS engineering book. Degraded sequences need to be confirmed to experts, since the subsystem engineering handbook may not describe all combination of such sequences and their expected behavior. Part of this work will be covered by the CoFI methodology.

Figure 5. FSM for the telecommands sequence described in Step 3.

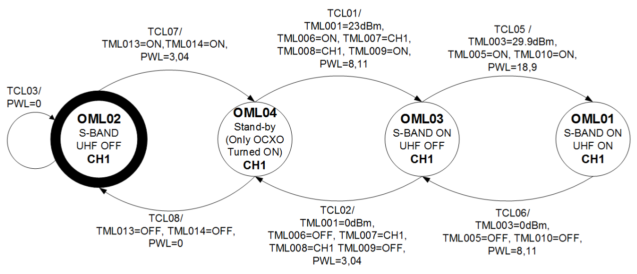

Fig. 5. FSM of the sequence stablished at step 3.

IV. DCS Modeling and Testes Execution

This section describes the Modelling for Testing, using CoFI and model checking, the transformation of CoFI-Total model and also the utilization of MME and Condado tools.

4.1 Applying CoFI Methodology

The first activity on using the CoFI methodology was to analyze what the specification defines as normal functioning of the subsystem, to, next, create a FSM to represent the "normal" behavior of the subsystem operation. For this case, we used the complete FSM illustrated in Figure 5.

In the following, an FSM was modeled to "specified exceptions". According to the DCS behavioral model this specified exception occurs when one channel changes to another. Although the DCS subsystem can work normally in both channels, their modeling is accepted as an exception behavior - stop operating on Channel 1, starting work on Channel 2. This occurs because the DCS can operate continuously on Channel 1, no need to switch to Channel 2.

After that, it was created a FSM to "sneak paths". This is basically to complete the FSM, make each state treat all the TCs that can happen in the SUT.

The application of CoFI methodology generated seven FSMs. The models generated by CoFI methodology received the name **CoFI Set**.

4.2 Conceiving CoFI-Total Model

The **CoFI-Total** model arose from the union of those seven models, which was very simple, first the establishment of the connection between channel 1 and 2, in the meantime removing the intersected states and transitions. Second, the addition of the degraded states and removing of the intersected states and transitions.

4.3 Applying Model Checking with UPPAAL Tool

The CoFI-Total model was the input to UPPAAL tool, which checks whether the model meets properties established in subsystem requirements. 75 properties were analyzed, the deadlock absence was the first one analyzed to guarantee that the system should never reach a situation in which no progress can be made. Whether the property set was not completely satisfied, the model was improved and checked again, until all the properties are satisfied. That means, in a given state, with a sequence of several TCs, a variable will always present a value defined by the rules presented in the DCS model specification, e.g. The variable SYSTEM.OML02_CH1 implies PW == 0, this property analyses if in the state OML02_CH1, the PWL is 0.

During the model checking analysis, we evaluated the requirements and checked them against the model to identify misunderstandings and inconsistencies in the CoFI-Total model, resulting 25 updating in the CoFI-Total model.

Finally, we pointed out that the CoFI-Total model meets the DCS subsystem requirements, taking into account the analyzed properties, proving that the model is suitable for the test case creation.

4.4 Executing the test cases

Condado Tool automatically generated a set of 444 test cases. During the execution of the first 185 test cases we found that, 39 test cases passed indicating compliance with the requirements and 5 critical faults were detected in the implementation. For continuing the test execution, we need the correction of the discovered faults to be done by the software development team. Meanwhile the verification team is working in the modeling of another subsystem model to be tested.

Concerning the time to execute the 185 test cases we spent about 10 hours and 9 minutes using a microcomputer with processor Intel Core 2 Duo and 4GB of RAM, which was available for testing purposes.

[1] A complete operation is defined here as the activation of all the equipment that comprises the system, including main and redundant equipment.

V. RELATED WORKS

Prerogative The related work found in literature focuses on the uses of Model-based Testing to support automatic generation of test cases either by the comparison between simulation approaches or formal approaches. Frequently, the purpose of the comparison is runtime performance of different tools in the same verification approach.

Sijtema et al [17] discusses the experience of industrial use for formal methods applied during the development of a software bus (Xbus) and its pros and cons. The experiment were obtained during two phases using formal methods (in both the design step and the integration testing step): a first phase in which was developed the XBus at Neopost Inc., and a second, post-case study analysis phase, where was performed model checking of the XBus protocol, and measured the quality and performance of the model-based testing process.

Ambrosio et al [18] presents an ISVV (Independent Software Verification and Validation) process that applies reviews for verification and a systematic testing methodology to guide validation. This process was applied to a pilot project named Quality Software Embedded in Space Missions (QSEE) at INPE. These features allowed systematizing validation activities supported by the test case generation.

Dorofeeva et al [19] presents a comparative analysis of methods for automatic test suites generation using W, Wp, HSI, H, UIOv, UIO, and DS FSM-based conformance testing methods. The experiments here are applied on two communications protocols.

Tretmans and Brinksma [20] discusses the goal of Côte de Resyste which is to develop theory, tools and applications for automatic specification based testing using formal methods. The ioco-test theory provides a well-defined and rigorous basis for formal testing with proved test derivation algorithms. The prototype test tool TorX can completely automatically derive tests from formal specifications, execute them, and analyze the results. The successful application of TorX to different case studies showed the feasibility of the methodology, and the improvements of the testing process which were gained in terms of more, longer and provably correct tests.

Pontes at al [21] presents a comparative among two verification techniques that are based on the system modelling as state machines. The first technique is the formal verification of the system specification using timed automata and the model checker UPPAAL. The second technique is the test execution of the delivered software product. The specification of the test cases is based on the CoFI methodology.

Enoiu et al [22] describes how test case generation that is aimed to satisfy logic coverage on function block diagrams can be solved as a model checking problem, using model checking tools to automatically create traces. That can be transformed into executable tests and how a toolbox in which logic coverage criteria can be formalized and used by a model checker to generate test cases.

Several works has been done in this area, which emphasizes on the importance and benefits of the application of MBT and Model Checking for test automation. However, no work, to the best of our knowledge, has been done to formulate a method to generate a finite state machine from cause-effect tables of evolving system to support as primary input the test automatic generation.

VI. CONCLUSION

This paper proposes a verification approach for a satellite operational simulator, which is an evolving system. The approach is illustrated with the modelling of a real satellite subsystem, the DCS. The DCS behavior is specified through cause-effect tables. The challenge was to translate these tables into a set of FSMs. For that a Translation method that is still in development, has been defined. Based on a set of FSMs, we may achieve the automatic generation of test cases to test the DCS model implementation. The results took into account the way the rules of the DCS model were represented. The final result will change whenever the way of representation of the subsystem rules changes.

The continuation of this work aims to finish the translation method and seek for generating the test cases directly from the UPPAAL model.

The DCS subsystem is part of the CBERS Satellite Simulator. All the 15 other satellite subsystems follow the standard to specify the subsystem behavior in rule-based tables, so the presented approach, when ready, will contribute a lot for reducing time and cost of an evolving system.

We believe that the generic ideas presented here are the first step towards an evolving verification and validation approach, which will be compatible with an evolving system.

ACKNOWLEDGMENT

The authors would like to thank the support given by Coordination for the Improvement of Higher Education Personnel, CAPES, Brazil, Brazilian National Institute for Space Research (INPE) and to Brazilian Aeronautics Institute of Technology (ITA). We would like to thank to Márcio Branco, Felipe Costa and Italo Rodrigues members of the CBERS 3&4 satellite simulator developing team at INPE. A special appreciation to Christopher Cerqueira, post-graduate student at INPE, Ivan L. Tosetto Junior and Lucius B. Trannin Cividanes, both Team Engineering Division at INPE and also Alheri Dakwat, a post-graduate student at ITA.

REFERENCES

[1] EUROPEAN COOPERATION FOR SPACE STANDARDIZATION (ECSS). Space engineering: System modelling and simulation. ESTEC, P.O. Box 299, 2200 AG Noordwijk - The Netherlands, apr. 2010. ECSS-E-TM-10-21A.

[2] Ambrosio, A. M.; Cardoso, P. E.; Orlando, V.; Neto, J. B. Brazilian Satellite Simulators: Previous Solutions Trade-off and New Perspectives for the CBERS Program, Spaceops Conference, AIAA, Rome, Italy, 2006.

[3] Tominaga, J., Cerqueira, C. S., Kono, J., andAmbrosio, A. M. (2012). Specifying satellite behavior for an operational simulator.In roceedings...Simulation and EGSE facilities for Space Programmes, (SESP 2012)., SESP 2012.

[4] Ambrosio, A.M., CoFI: Uma abordagem combinando teste de conformidade e injeção de falhas para validação de software em

aplicações espaciais, INPE-13264-TDI/1031, São José dos Campos, 2005.

[5] Baier, C.; Katoen, J. Principles of Model Checking. The MIT Press (April 25, 2008).

[6] Anjo, J.M.S.; Villani, E. Modelagem e Verificação de uma Proposta para Arquitetura de Controle de um Efetuador Robótico Baseada em Labview. VI National Congress of Mechanical Engineering, August 18 – 21, 2010 – Campina Grande – Paraíba – Brazil.

[7] Melo, P.C.B.; Junior, R.A.P, Modelador de máquinas de Estado – MME v1.0.0, PIBIC/UFRN/CRN/INPE – 2003, Projeto ATIFS. Available at: <http://www.inpe.br/atifs/ferramentas/ferramenta_mme.php> Accessed on: 2016/05/25.

[8] Martins, E.; Sabião, S.B.; Ambrosio, A. M. ConData: a Tool for Automating Specification-based Test Case Generation for Communication Systems. **Software Quality Journal**, v.8, n. 4, p. 303-319, 1999.

[9] Mattiello-Francisco, F.; Ambrósio, A.M.; Villani, E; Martins, E; Dutra, T.; Coelho, B. . An experience on the technology transfer of CoFI methodology to automotive domain. LADC'2013, April 2-5, 2013, Rio de Janeiro, Brazil. ISBN 978-85-7669-274-4.

[10] Naito, S and Tsunonyama, M. "Fault Detection for Sequencial Machines by Transition Tours", Proc. 11th IEE Fault Tolerant Computing Symposium (1981) pp 238-243.

[11] Ambrosio et al. ATIFS, Ambiente de teste baseado em injeção de falhas de software. 2003, Projeto ATIFS. Available at: <http://www.inpe.br/atifs/index.php> Accessed on: 2016/05/25.

[12] Clarke, E.M; Grumberg, O; Peled, D. Model Checking. The MIT Press; n edition (January 7, 1999).

[13] Losso, R.; Villani, E.; Saotome, O.; Góes, L.C.S. Modelagem e Verificação de Sistemas Operacionais de tempo-real para sistemas críticos embarcados. XVIII Congresso Brasileiro de Automática / 12 a 16 Setembro 2010, Bonito-MS, Brazil.

[14] Tominaga, J; Ambrosio, A.M. RTD-SRS-1006/00. DCD Model Technical Specification. INPE, Brazil, 2011.

[15] Tosetto, I. RBL-HBK-1030(F4)/00. FM4 Data Collection Subsystem (DCS) Handbook. INPE, Brazil, 2014.

[16] Tominaga, J.; Gonçalves, C. RBL-TRP-1817/00. DCS Subsystem In Orbit Verification Results. INPE, Brazil, 2015.

[17] Sijtema, M. ; Belinfante, A.; Stoelinga, M.I.A.; Marinelli, L. Experiences with formal engineering: Model-based specification, implementation and testing of a software bus at Neopost. Science of Computer Programming, Volume 80, Part A, 1 February 2014, Pages 188-209.

[18] Mattiello-Francisco, F.; Ambrósio, A.M.; Martins, E. An Independent Software Verification and Validation Process for Space Applications. SpaceOps 2008 Conference (Hosted and organized by ESA and EUMETSAT in association with AIAA).

[19] Dorofeeva, R.; El-Fakih, K.; Maag, S.; Cavalli, A.R.; Yevtushenko, N. FSM-based conformance testing methods: A survey annotated with experimental evaluation. Elsevier. Information and Software Technology 52 (2010) 1286–1297.

[20] Tretmans, J.; Brinksma, E. (2003) TorX: Automated Model-Based Testing. In: First European Conference on Model-Driven Software Engineering, December 11-12, 2003, Nuremberg, Germany (pp. pp. 31-43).

[21] Pontes, R.P.; Essado, M.; Véras, P.C.; Ambrósio, A.M.; Villani, E. A Comparative Analysis of two Verification Techniques for DEDS: Model Checking versus Model-based Testing. Elsevier, IFAC Proceedings Volumes. Volume 42, Issue 21, 2009, Pages 66-71. 4th IFAC Workshop on Discrete-Event System Design.

[22] Enoiu,E.P.; Čaušević, A.; Ostrand, T.J.; Weyuker, E.J.; Sundmark, D.; Pettersson, P. Automated test generation using model checking: an industrial evaluation. International Journal on Software Tools for Technology Transfer. June 2016, Volume 18, Issue 3, pp 335–353.