Embedded Critical Software Testing for Aerospace Applications based on PUS

Rodrigo P. Pontes[1], Eliane Martins[2], Ana M. Ambrósio[3], Emília Villani[1]

[1]Instituto Tecnológico de Aeronáutica (ITA)
Vila das Acácias, CEP 12.228-900 – São José dos Campos – SP – Brazil

[2]Instituto de Computação – Universidade Estadual de Campinas (Unicamp)
Avenida Albert Einstein, 1251, CEP 13083-970, Campinas – SP – Brazil

[3]Departamento de Sistemas e Solo – Instituto Nacional de Pesquisas Espaciais (INPE)
Avenida dos Astronautas, 1758, CEP 12227-010, São José dos Campos – SP – Brazil

{rpastl,evillani}@ita.br, eliane@ic.unicamp.br, ana@dss.inpe.br

Abstract. *This paper discusses the practical experience of verifying an On-Board Data Handling (OBDH) software to be used in a future satellite application at INPE using the CoFI testing methodology. This technique is proper for aerospace applications and is based on modeling the system under test as finite state machines. The test cases are automatically generated from the developed models. The OBDH software considered in this paper follows the PUS standard from European Cooperation for Space Standardization, which is being adopted in Brazil. Among the important issues analyzed by this practical experience are the errors found, the time required for the modeling activity, the time required for testing, the reusability of the test cases, among others.*

1. Introduction

During last decades, the software role in space embedded systems has increased. However, the attention and efforts dedicated to its design and verification have not increased in the same way. Hardware is still the main concern of the development of embedded systems. When the project resources are limited, the efforts are addressed to hardware issues rather than software. As a consequence, software is also playing a significant role in accidents, Leveson (2005).

Considering this scenario, this work analyzes one specific technique for the verification of space embedded software: the CoFI (*Conformance and Fault Injection*), Ambrosio (2005). CoFI is a model based testing methodology that uses state machines to represent the behavior of the system under different assumptions. The test cases are generated automatically from these models and they are applied to the system under test.

The main purpose of this work is not to compare this methodology to others, but to identify the advantages and the limits of its utilization through a practical experience, a practical case study. The comparison among others testing methodologies were performed in Ambrosio(2005).

This work discusses the results of the application of the CoFI testing methodology into a case study in the space area: the on-board data handling (OBDH) software to be used in a future satellite application at INPE. This OBDH is being developed using an object-oriented implementation, Arias et al. (2008).

The OBDH software is based on the PUS (Package Utilization Standard), a proposal of the European Cooperation on Space Standardization (ECSS) that have also been adopted in Brazil. The use of standards in space area has been motivated by time-saving and dependability-improvement of the software development. The application of a testing methodology based on models that are derived from a standard will consequently reduce the cost with tests. The modeling process performed in this work is general, because it is developed from the PUS standard.

For this case study, important issues related to the CoFI applicability are discussed, such as the size of the models, the time spent on modeling the system, the time spent on the application of the tests, the number of errors detected, how critical the detected errors are, among others.

This work is organized as follows: Section 2 introduces the CoFI methodology and the Condado tool. Section 3 discusses previous works developed with the CoFI. Section 4 presents the PUS standard and details the telecommand verification service. This service is used is Section 5 to present the application of CoFI to the OBDH software including the models, the test cases and the results. The Section 6 brings some conclusions and discusses the contributions of this work.

2. CoFI Testing Methodology and the Condado Tool

The CoFI Testing Methodology consists of a systematic way to create test cases for space software. The CoFI is comprised of steps to identify a set of services. Each service is represented in finite state machines. The models represent the behavior of the System Under Test (SUT) under the following *classes of inputs* arriving: (i) normal, (ii) specified exceptions, (iii) inopportune inputs and (iv) invalid inputs caused by hardware faults.

The software behavior is represented by small models taking into account the decomposition in terms of: (i) the services provided by software and (ii) the types of behavior under the classes of inputs. The *types of behavior* defined in the context of the CoFI are: Normal, Specified Exception, Sneak Path, and Fault Tolerance. These behaviors are respectively associated to the following inputs: normal, specified exceptions, inopportune and invalid inputs. More than one model can be created in order to represent a type of behavior for a given service.

After the creation of the partial models, each model is submitted to the Condado tool that is able to tour the model, Martins (1999). The Condado tool generates the test cases from these models combining different sequences of inputs. Each test case is a sequence of inputs and its expected output(s) associated to the transitions of a tour. Each tour ends in the final state, if the final state is the initial state, the application of the test cases is performed without restarting the system. The CoFI test case set is the union of the test cases generated from each model and must cover all the transitions of the finite state machine models.

The main reasons of choosing the Condado tool are: the generation of independent test cases; the cover of all transitions of the model; availability of the tool; and the validity of all test cases. The latter is justified by the fact that all test cases are a sequence of inputs that starts in an initial state and they are led to a final state. The disadvantage is the generation of possible repeated test cases.

The generation of the test cases manually, besides being a tough task, might introduce errors during the process. Thus, as there was already an automatic tool for generating test cases, it was used.

It is not crucial the utilization of the Condado tool. Other tools can be used, provided that it covers, at least, all the transitions of the model and it accepts partial finite state machines.

3. Other CoFI Applications

In Ambrosio et al. (2008), the CoFI testing methodology was applied in the context of an independent software verification and validation process of the Quality Software Embedded in Space Missions (QSEE) Project carried on at INPE. The software under test in the QSEE Project is the software embedded in the Payload Data Handling Computer, which is part of a scientific X-ray instrument onboard of a scientific satellite under development at INPE. The CoFI methodology served as a guideline to focus the tester's attention on the faults and exceptions that occur during the software's operation, leading to situations that the developers had not thought of.

Pontes et al. (2009) compares two different verification techniques: model checking and the CoFI Test methodology. It uses an automatic coffee machine example as a case study to show the contributions of each technique. Because of weak points identified in both techniques, the work conclusion is that the two techniques are complementary to each other. The main contributions of the techniques are the detection of incomplete and inconsistent requirements, the introduction of testability requirements and an adequate treatment of all exceptions.

In Morais and Ambrosio (2010), an adaptation of the CoFI is proposed to be applied in the initial phases of the software development, as a new approach to refine software requirements. The new approach is applied to precisely define the operation requirements of a satellite.

4. Application with PUS standard

This section introduces the PUS Standard and details the "*Telecommand Verification Service*". This service is used as an example in the next section to illustrate the application of CoFI to the OBDH software.

4.1. The PUS Standard

The PUS (*Packet Utilization Standard*) is one of the standards of the ECSS (*European Cooperation for Space Standardization*), released in January 2003. The ECSS is an effort of European national agencies and European industrial associations to develop and maintain common standards. The main benefits of these standards are the costs and efforts reduction regarding conception and development of space missions, ECSS (2003).

The PUS, or the ECSS-E-70-41A standard, focuses on the ground and systems operations related to the utilization of telecommand and telemetry packets. It standardizes these packets and describes sixteen services which the OBDH (*On-Board Data Handling*) should provide. Figure 1 shows these sixteen services. The underlined service is the one used in this work.

Each service has an identification called "*Type*". Depending on the type of the service, there are specific activities, called "*Subtypes*", which are responsible for performing the user´s request. Therefore, the telecommand and telemetry packets are variable: they may correspond to the chosen type and subtype. Figure 2 shows the fields of a telecommand packet, highlighting the field "*Data Field Header*", where the type and subtype are defined in the request. The PUS addresses the shaded fields, although some of the white fields, such as "*Packet ID Type*" and "*Packet ID Data Field Header Flag*", have also been defined in PUS with default values.

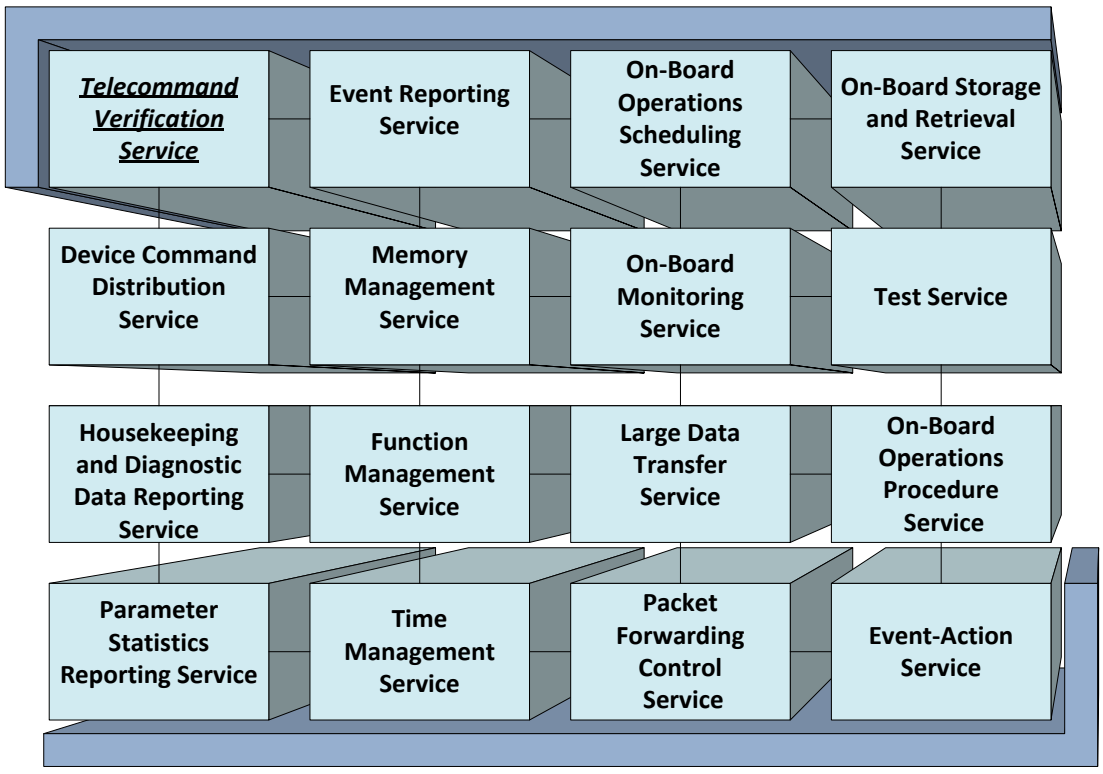

Figure 1. PUS Services.

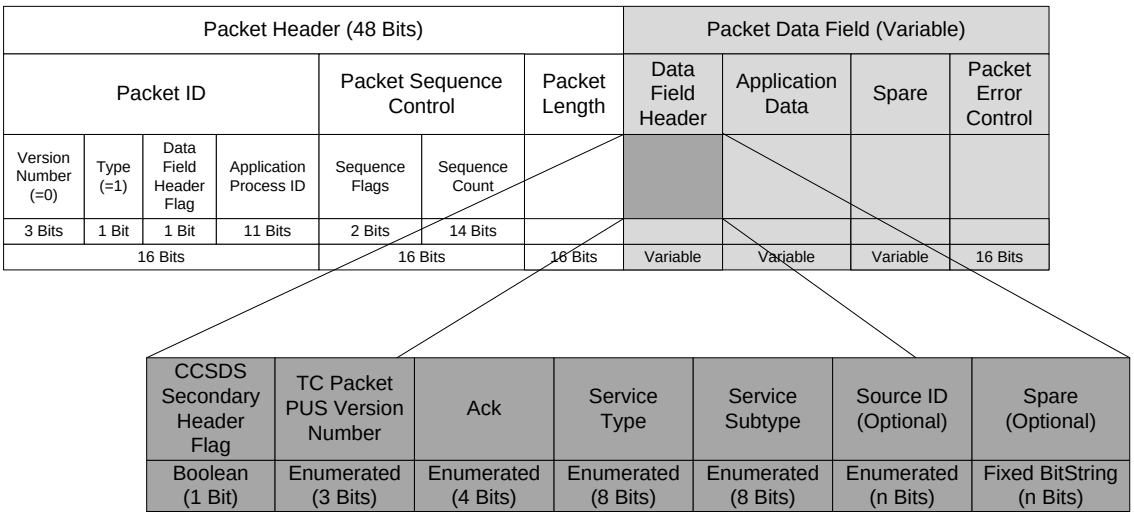

Figure 2. PUS Telecommand Packet.

4.2. Telecommand Verification Service

According to ECSS (2003), The Telecommand Verification Service provides the capability of checking the execution of the each telecommand packet, from its acceptance through to its completion of execution. There are four different stages for the telecommand verification. Although providing the verification of the telecommand, it is not necessary that every telecommand should be verifiable at each stage. The stages are:

- ***Telecommand acceptance***
- ***Telecommand execution started***
- ***Telecommand execution progress***
- ***Telecommand execution completion***

Within the range between the telecommand acceptance and the telecommand completion of execution, the user can request an execution success report, which allows him to follow the exact point of the execution. The success report is requested through the "Ack" field of the telecommand packet. This field is shown in Fig. 2 above.

When a failure occurs at any stage, this service must send a failure report to the user containing the error code and some additional information regarding the cause of this failure. It helps the user to understand the main reason of such failure.

In short, each stage should have two reports: Success Report and Failure Report. It results in eight reports that the telecommand verification service shall provide.

The type number of this service is 1. The subtype numbers of each report are listed in Table 1. For this work, only the "Telecommand Acceptance" and "Telecommand Execution Completion" stages were used.

Table 1. Telecommand verification reports and its subtypes identification.

Report	*(Type , Subtype)*
Telecommand Acceptance Report - Success	***(1 , 1)***
Telecommand Acceptance Report – Failure	***(1 , 2)***
Telecommand Start of Execution Report – Success	***(1 , 3)***
Telecommand Start of Execution Report – Failure	***(1 , 4)***
Telecommand Progress of Execution Report – Success	***(1 , 5)***
Telecommand Progress of Execution Report – Failure	***(1 , 6)***
Telecommand Completion of Execution Report – Success	***(1 , 7)***
Telecommand Completion of Execution Report – Failure	***(1 , 8)***

5. Application of CoFI to Telecommand Verification Service of an OBDH

In this section the telecommand verification service is used as an example to illustrate the application of the CoFI testing methodology to the OBDH software of a satellite that follows the PUS standard. This service is chosen because it is a mandatory service for any OBDH software that follows the PUS. The strategy used in this work is summarized in Fig. 3.

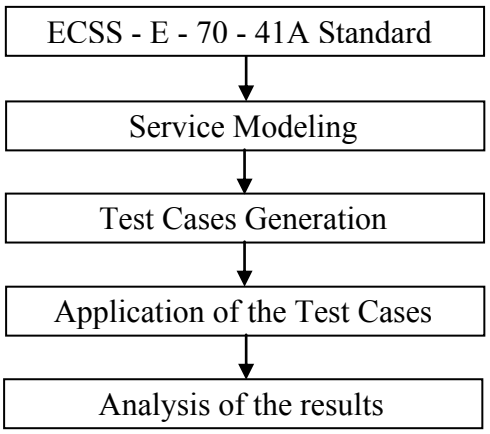

Figure 3. Strategy.

From the description of the service provided by the PUS standard, finite state machines are specified to represent the behavior of specific scenarios. This step uses the CoFI testing methodology to develop the models. It is important to note that this is not done automatically. Then, the test cases are obtained from these state machines using the Condado tool.

The next step is to execute manually the test cases against the OBDH and observe the responses of the OBDH. Both of these activities use a TET (Test Execution

Tool), which is also under development. These steps are shown in Fig. 4. Finally, the errors found with the test cases application are used to analyze the contribution of the testing methodology.

One important point to highlight is that, differently from the works discussed in Section 2, in this work the starting point for the development of the finite state machines is not the requirement specification, but a standard, the PUS, which is used as basis to develop on-board computer software. As a consequence, one of the issues analyzed in this work is the viability of reusing these test cases for any other software that follows this same standard.

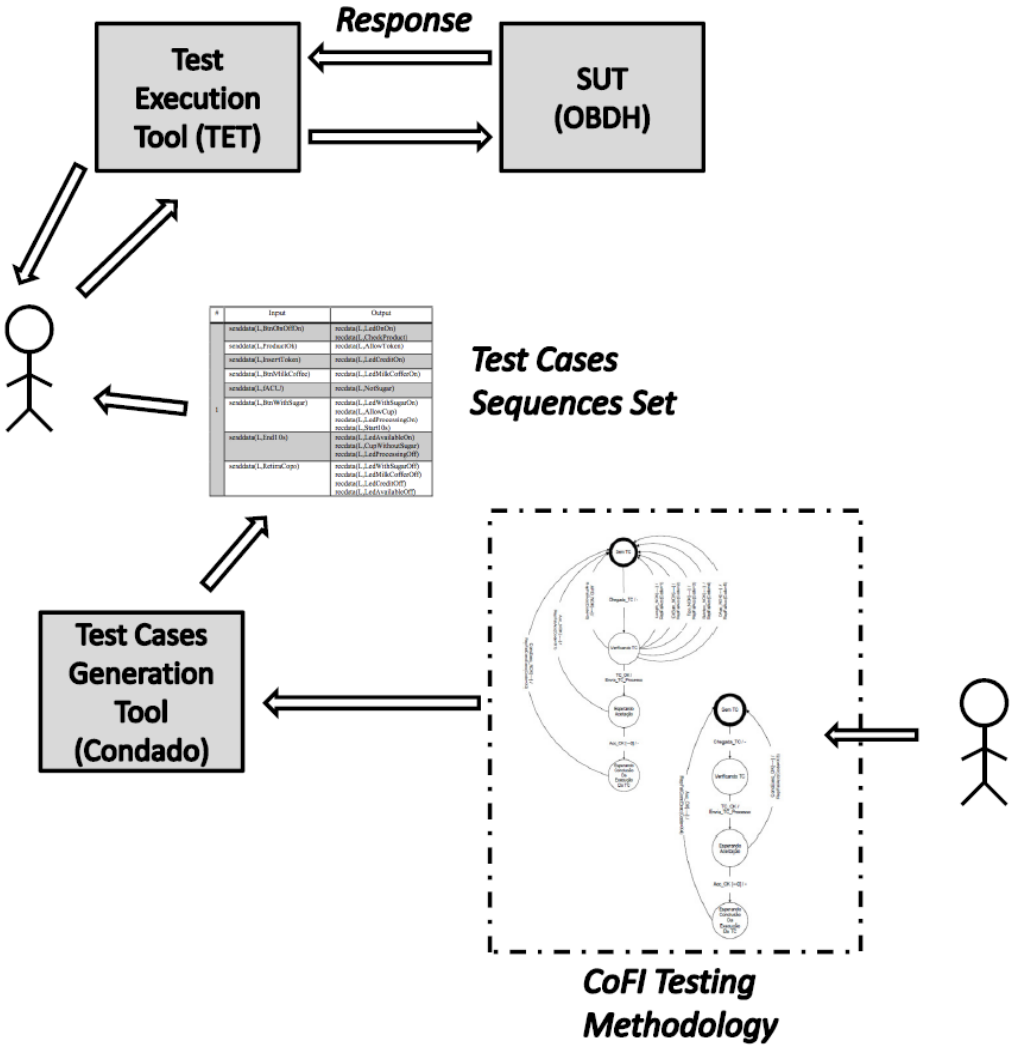

Figure 4. Test application activities.

5.1. Service Modeling in Finite States Machine

After the analysis and understanding of the PUS standard, the telecommand verification service was modeled in finite state machines.

Following the CoFI methodology, four different classes of state machines should be developed: (i) normal, (ii) specified exceptions, (iii) inopportune inputs and (iv)

invalid inputs caused by hardware faults. However, hardware faults are not addressed by the PUS standard. As a consequence, only classes (i) to (iii) could be modeled.

In the case of the telecommand verification service, each class is modeled by one finite state machine. The first model represents the normal behavior of this service and it is shown in Fig. 5. Four states represent the current stage of the service depending on the event associated to the transition. The events "TC_Arrival" and "TC_OK" are events from the embedded software, and thus, the person who is applying the test cases cannot observe their occurrences.

The responses of the system are within the telemetry packets. They contain the reports mentioned in section 4.2. The responses "RepSucAcc" (*Success Report of Telecommand Acception)* and "RepSucCompExec" (*Success Report of Telecommand Completion of Execution)* are only sent if their respective bits, '3' and '0' in the telecommand "*Ack*" field, are set to '1'. Otherwise, they are not sent. The "*Ack*" field of the telecommand is represented by the four bits "0123" inside the brackets in the events "Acc_OK[0123]" and "CompExec[0123]".

Telecommand Verification Service –
Normal Behavior Model

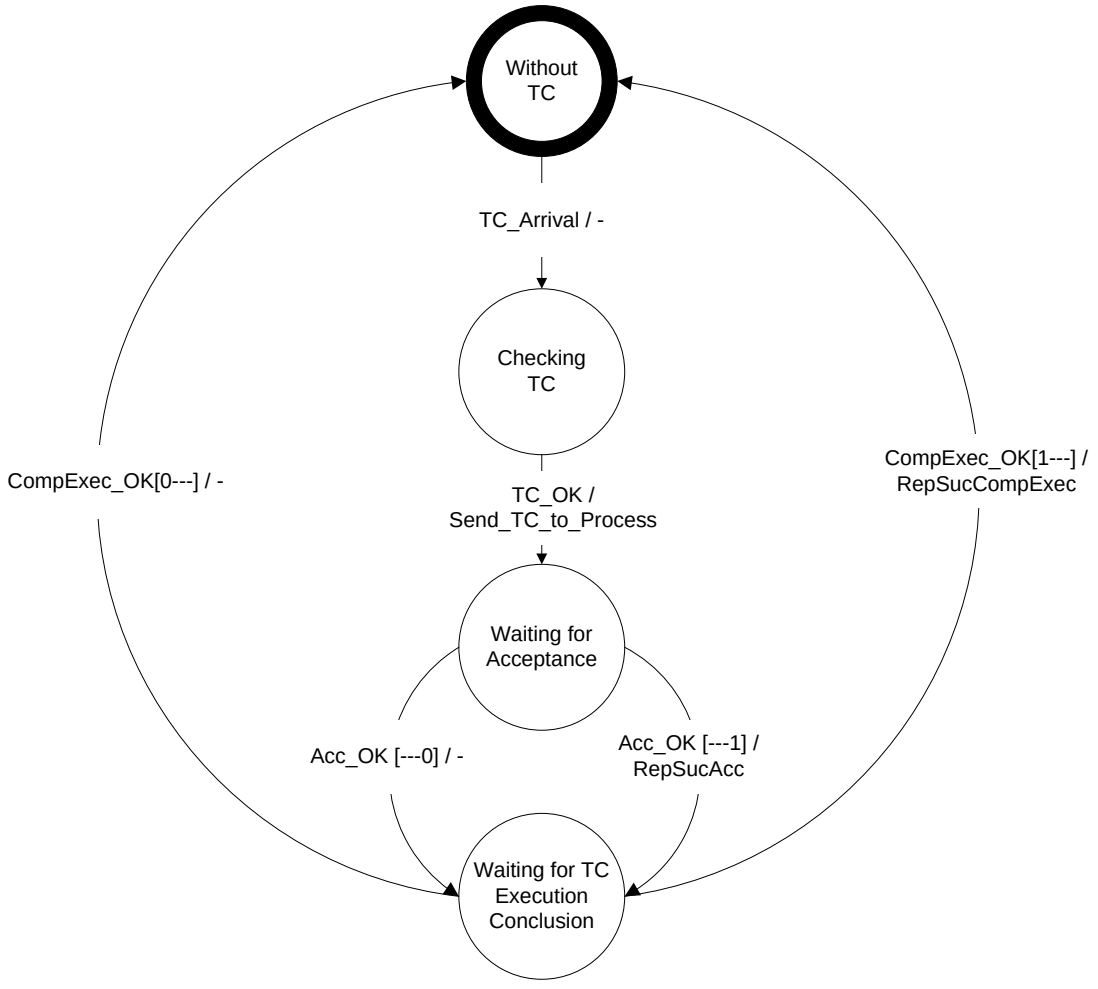

Figure 5. Telecommand Verification Normal Behavior model.

As a second example, the model developed for the Specified Exceptions Behavior is presented in Fig. 6. This class encompasses the events that the standard has defined for specified types of failure or abnormal functioning of the system. In this case, the PUS defines six failures that may occur in the acceptance stage and their respective codes. For other stages, the errors are mission-specific. The codes "X1" and "X2" in Fig.6 show that these errors are not in the standard. Also, it is important to note that the "Ack" bits are represented by this configuration "[----]". It means that the corresponding failure report must be sent regardless the value of these bits.

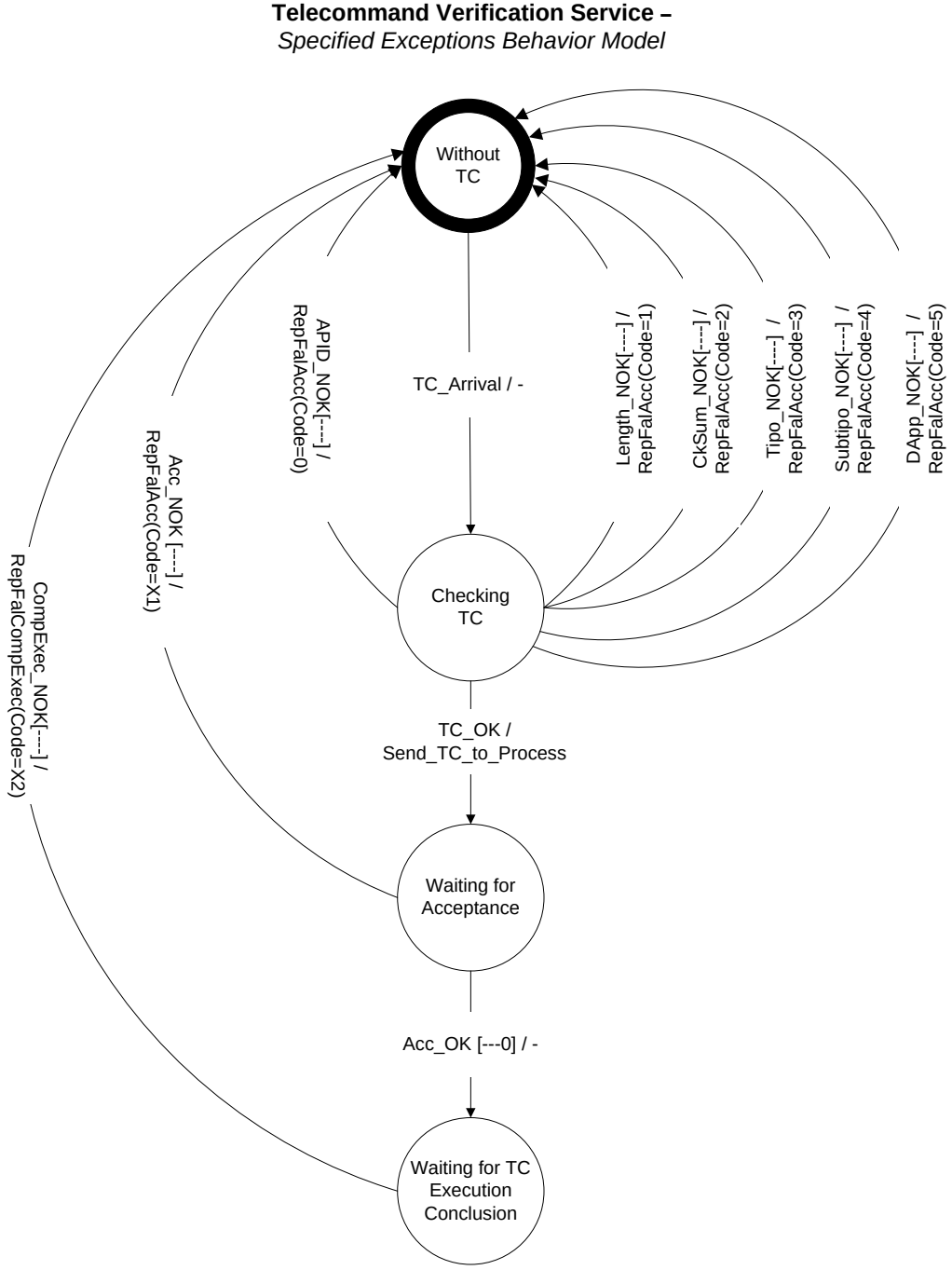

Figure 6. Telecommand Verification Specified Exceptions Behavior model.

The third, and last state machine developed, is the Sneak Paths Behavior, shown in Fig. 7. This model consists in the expected events occurring in inopportune moments. In this model, when the OBDH software is in the acceptance state, waiting for the acceptance event from the process application, if it receives the event "CompExec_OK[----]", the service software shall send a failure report regarding this failure, and return to its initial state. Analogously, when the service software is in execution completion state, if it receives the event "Acc_OK[----]", it shall send the failure report to inform the failure. Note also that, as well as the model of Fig.6, the "*Ack*" field has the configuration "[----]", meaning that regardless its bits values, if the those events occur, the report must be sent and the service software must return to its initial state. The error codes are mission-specific.

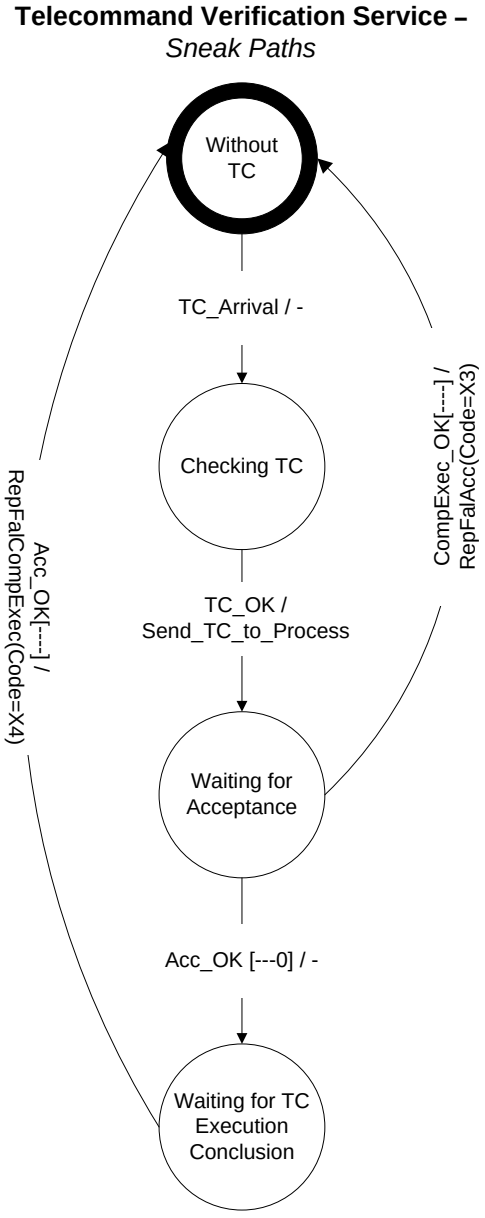

Figure 7. Telecommand Verification Sneak Paths Behavior model.

In general, the size of the finite state machine models is considered small. The biggest model has only four states and eleven transitions.

5.2. Generation of Test Cases

Sixteen test cases were generated from the three developed models. The Tables 3, 4 and 5 show, respectively, some test cases from Normal Behavior Model, Specified Exceptions Behavior Model and Sneak Paths Behavior Model. A test case is a sequence of inputs and outputs.

Table 3. Test cases for Normal Behavior Model.

CASE NUMBER	***INPUT***	***OUTPUT***
2	*TC_Arrival*	-
	TC_OK	*Send_TC_to_Process*
	Acc_OK[---1]	*RepSucAcc*
	CompExec_OK[---0]	-
4	*TC_Arrival*	-
	TC_OK	*Send_TC_to_Process*
	Acc_OK[---1]	*RepSucAcc*
	CompExec_OK[---1]	*RepSucCompExec*

Table 4. Test cases for Specified Exceptions Behavior Model.

CASE NUMBER	***INPUT***	***OUTPUT***
7	*TC_Arrival*	-
	APID_NOK	*RepFalAcc(Code=0)*
8	*TC_Arrival*	-
	Length_NOK	*RepFalAcc(Code=1)*
9	*TC_Arrival*	-
	Checksum_NOK	*RepFalAcc(Code=2)*
10	*TC_Arrival*	-
	Type_NOK	*RepFalAcc(Code=3)*

Table 5. Test cases for Sneak Paths Behavior Model.

CASE NUMBER	***INPUT***	***OUTPUT***
15	*TC_Arrival*	*-*
	TC_OK	*Send_TC_to_Process*
	Acc_OK[---0]	*-*
	CompExec_OK[---1]	*RepFalAcc(Code=X4)*
16	*TC_Arrival*	*-*
	TC_OK	*Send_TC_to_Process*
	CompExec_OK[---1]	*RepFalAcc(Code=X3)*

5.3. Application of Test Cases and Analysis of the Results

The test cases application to the OBDH software was characterized by the following events and results.

Initially, only nine of the test cases were applied to the OBDH software. The other seven test cases could not be applied because the TET did not allow to generate telecommand packets containing the following errors: invalid checksum, invalid packet size and, invalid sequence count. The first consequence of the application of the CoFI testing methodology was a request to modify the TET, improving its flexibility and usability.

As a response to this request, a new version of the TET was generated and three other test cases were applied in a second moment. The remaining four test cases could not be applied because the modeled events are internal to the on-board software. They are related to the communication between the PUS service and the application processes. These events cannot be generated by the TET. These test cases are related to the Sneak Paths Behavior model and basically represent the situations when the application process gives input events to the telecommand verification service in the wrong stages. This functionality may eventually be considered in the future for incorporation in the testability environment of the OBDH.

Regarding the detection of errors in the OBDH software, two test cases resulted in erroneous output. The first one is the application of a test case with one specified exception. In this case, the OBDH software stopped receiving telecommand packets and sent telemetry packets indicating the error code for “acceptance failure”, even if the telecommand packet was a correct one. The second error is related to the reception of two inconsistent telemetry packets. These errors are considered critical, because they can cause the systems’ blockage, compromising the whole mission.

After the presentation of the results, the development team corrected the OBDH software and a new application of the test cases resulted in no error.

The process of modeling, generating and applying the test cases set spent forty hours. This time includes also the time intervals spent by the development team on modifying the TET and the OBDH software, and the second application of the test cases by the testing team.

6. Conclusions

This paper analyzes the contributions of one specific verification technique for the development of space embedded software. The verification technique is the CoFI testing methodology and it is applied to the OBDH software of a satellite that follows the PUS standard.

The main conclusions of this work are the following.

This is a new methodology that is still in its fourth practical experimental application. The results of the utilization of this methodology are being evaluated. Its main limitations are that it does not cover system performance tests, it does not cover tests regarding combination of services, and it does not guarantee the coverage of the code, because it is a black-box testing methodology. However, it has shown itself as a good method to cover tests at system level for acceptance purposes, in which source code is not available.

The CoFI guides the decomposition of the system behavior in different classes of behavior for each different service the system provides. The model of each class of behavior contains only the events related to that class. As a consequence, the models are small. They can be easily understood and analyzed by the development and testing teams. The equivalence of the set of test cases generated from the state machine of the complete system behavior and the set of test cases from the partial models of the system, i.e. smaller state machines, is proven in Ambrosio (2005).

One important contribution of the CoFI methodology is on the specification of the Test Execution Tool (TET). In the case study, the application of CoFI resulted in the elaboration of some requests for providing flexibility of this tool.

Based on the results of the execution of the generated test cases, the relevant contribution of CoFI was in the detection of errors in the OBDH software, which were considered as critical ones. This detection resulted in important corrections of the OBDH software.

All the activities related to the generation of the testing models, the generation of the test cases and the execution of the test cases, described in this paper, was performed by a team independent of the development team. All the models were created based on a standard (the PUS), and not based on the requirement specification of the software under test. This approach shows the reusability of the test cases. The same set of test cases can be applied to any other OBDH software that is based on the same standard.

The next activities are related to the extension of this work to other PUS services, using the same methodology. The On-Board Operations Scheduling Service is being considered due to its complexity.

7. Acknowledgements

The authors would like to thank the collaboration and support of Fabrício Kucinskis, Ronaldo Arias and Thiago Pereira of the Aerospace Electronics Department (DEA) of INPE. The authors would also like to thank the financial support of the Project Sistemas Inerciais para Aplicações Aeroespaciais (SIA), from the FINEP/CTA/INPE.

References

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, São José dos Campos(Brazil).

Ambrosio, A. M.; Mattiello-Francisco, M. F; Martins, E. (2008) "An Independent Software Verification and Validation Process for Space Applications", Proceedings of the 9th Conference on Space Operations (SpaceOps). Heidelberg (Germany).

Arias, R.; Kucinskis, F. N.; Alonso, J. D. D. (2008) "Lessons Learned from an Onboard ECSS PUS Object-Oriented Implementation", Proceedings of the 9th Conference on Space Operations (SpaceOps). Heidelberg (Germany).

ECSS – European Cooperation for Space Standardization (2003) "ECSS-E-70-41A – Ground systems and operations: telemetry and telecommand Packet Utilization", Noordwijk: ESA publication Division. Available online in: <http://www.ecss.nl>.

Leveson (2005), "N. Role of Software in Spacecraft Accidents", Journal of Spacecrafts and Rockets, Vol. 41, No. 4, pages 564-575.

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

Morais, M.H.E; Ambrosio, A.M. (2010) "A new model-based approach for analysis and refinement of requirement specification to space operations", Proceedings of the 10th Conference on Space Operations (SpaceOps). Huntsville (Alabama, USA).

Pontes, R. P. et al. (2009) "A Comparative Analysis of two Verification Techniques for DEDS: Model Checking versus Model-based Testing" ,Proceedings of 4th IFAC Workshop on Discrete Event System Design (DEDes), Spain, pages 70-75.