EXTFV: Automated Test Case Execution and Automated Generation of Artifacts related to the Verification and Validation Process

Valdivino Alexandre de Santiago Júnior[1], **Wendell P. da Silva**[1], **Raffael C. da Costa**[1], **Diogo V. dos Santos**[1]

[1]Instituto Nacional de Pesquisas Espaciais (INPE)
São José dos Campos – SP – Brazil

valdivino.santiago@inpe.br, {silvawp,raffaelc,diogodvs}@gmail.com

Abstract. *In this paper we present EXTFV, a tool that allows automated test case execution considering system and acceptance testing levels, where the entire system is under evaluation, for embedded software. EXTFV presents another important feature: automated generation of test specifications and reports in PDF format. These artifacts can be considerably large if we take into account system and acceptance test cases for complex embedded software systems. Make such specifications and test reports manually, considering the execution of each test case is error prone and may require a lot of time. Although there are several frameworks/tools that support automated unit test execution, EXTFV comes into picture to address not only parts (unit) but the software system as a whole.*

1. Introduction

Software testing is a process/method related to Verification and Validation. A software testing process is composed of a set of activities and organizations can adopt different solutions. Typical activities include *Plan Test* [Santiago Júnior 2011], *Generate/Select Test Cases* [Delamaro et al. 2007], *Execute Test Cases* [Santiago et al. 2008], *Evaluate Test Results* [Binder 1999], and *Select Test Cases for Regression Testing* [Mathur 2008].

Regarding automation, test case generation and test results evaluation are probably the most discussed activities by academia and industry. One explanation for this is the fact that it is more challenging to try to propose strategies to automatically select a set of test input data from an infinite set of possibilities and mechanisms for automatic generation of expected results than just dealing with the stimulus (execution) of the test input data to the Implementation Under Test (IUT).

However, automated test case execution is an important feature if one wants to really decrease the time spent during the entire software testing process. The reason is that it is not very interesting to automate the generation of test cases if a tester can not execute them automatically taking into account system and acceptance testing for complex software, and the effort that might exist for regression testing.

Although there are several open source and commercial frameworks and tools that support automated unit test execution, such as JUnit [JUnit.org 2013], CUnit [St.Clair and Kumar 2013], and Cantata [QA-Systems 2013], tools and frameworks for system and acceptance testing, where the entire system is under evaluation, for embedded software are not very common.

In this paper we present a tool called *Execução Automatizada de Casos de Teste e Geração Automatizada de Fornecimentos associados ao Processo de Verificação* (EXTFV - Automated Test Case Execution and Automated Generation of Artifacts related to the Verification and Validation Process). EXTFV is a tool that allows automated test case execution in the context of system and acceptance testing for embedded software. In addition, EXTFV presents another important feature: automated generation of test specifications and reports in PDF format. These artifacts can be considerably large if we take into account system and acceptance test cases for complex embedded software systems. Make such specifications and test reports manually, considering the execution of each test case is error prone and may require a lot of time.

This paper is organized as follows. Section 2 describes EXTFV's architecture and main functionalities. Section 3 presents aspects related to the usability of the EXTFV tool. Section 4 presents conclusions and future directions to follow.

2. EXTFV's architecture and main functionalities

EXTFV builds upon the NetBeans platform reusing components that provide common functionalities to applications such as loading and initialization, file system, user interface (HMI - Human Machine Interface), configuration, activity log, and a Java Native API (JNA) bindings component which allows communication with external devices such as analog, serial, and parallel ports. On top of this platform there is the *EXTFV Main Module* (component). All elements described above form the *EXTFV Application over the NetBeans Platform* component. One of the most important component of the tool is the *EXTFV Main Module*. A static architecture view of such module showing the dependency relationships between its internal components is in Figure 1.

Basically, the design of the *EXTFV Main Module* component follows the Model-View-Controller (MVC) design pattern with auxiliary components that deal with configuration (*Configuration*), data model (*Entity*), screens (*Editor*, *Actions*, *Nodes*), graph of objects in memory (*Models*), eXtensible Markup Language (XML) handling (*XML*), and artifacts (*Artifacts*). There is also an additional component that is related to a specific Data Acquisition Board (*DT9804*).

It is important to emphasize the *Entity* component of *EXTFV Main Module* which defines the data model of the application. Figure 2 shows the data model. This model defines the data manipulated by the tool. Essentially, there are communication channels (*Channel*) which abstract the communication among EXTFV and devices or with the IUT itself. Note that there is support for serial (*SerialChannel*), parallel (*ParallelPortChannel*), analog input/output (*AnalogESChannel*), digital input/output (*DigitalESChannel*), and socket (*SocketChannel*) communications. *TestCase* is a container for several test steps (*TestStep*), i.e. a test case is composed of 0 or more test steps.[1] Moreover, EXTFV allows several types of test steps: *ManualActivityTestStep*, to present an activity to be manually performed by the tester during the test; *WaitTestStep*, to pause for a period of time which can be specified in seconds and milliseconds to provide more flexibility to the tester; *FormattedMessageTestStep*, to allow the declaration of two formatted data messages according to a set of data fields. The first message is a request, and the second the expected

[1]EXTFV allows that a test case can be "composed" of 0 test step. But the tester must avoid this situation when adding test steps to a test case.

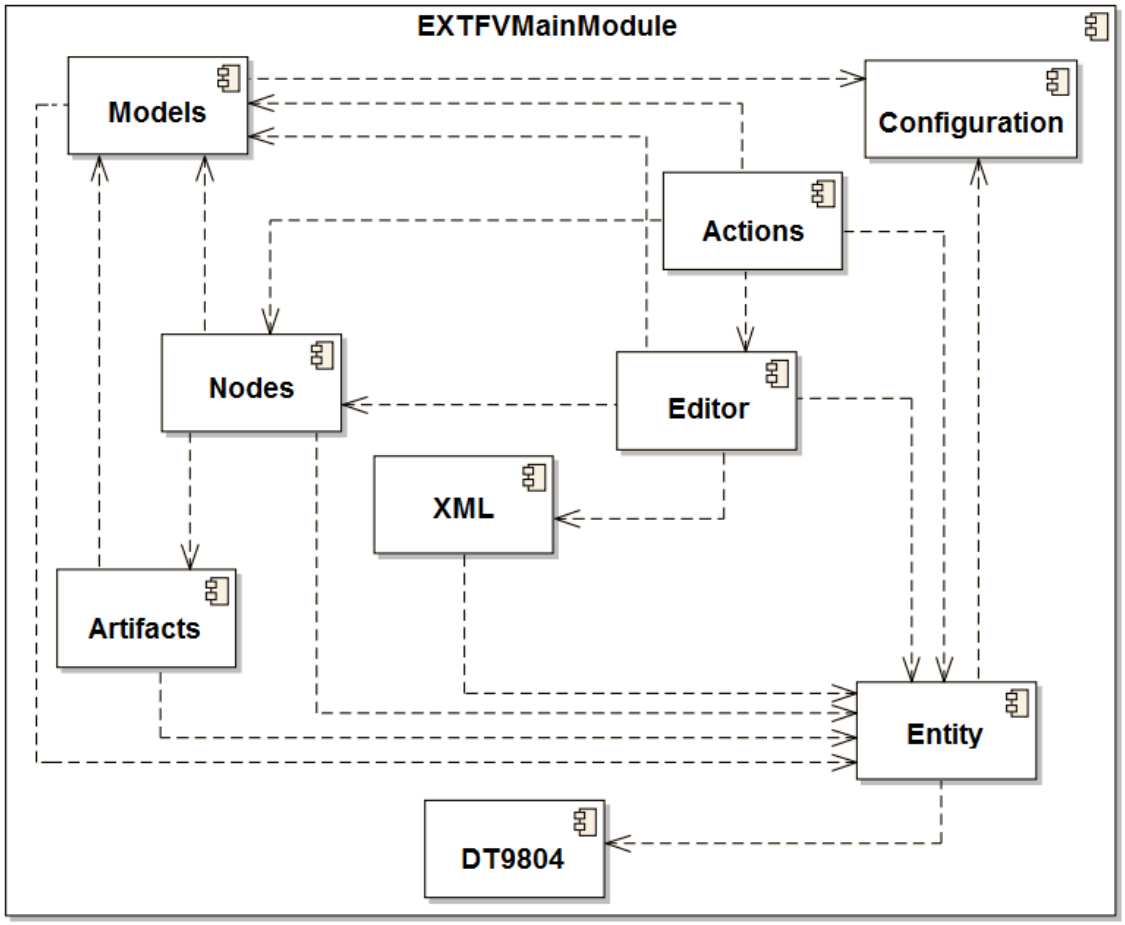

Figure 1. Static architecture view of the *EXTFV Main Module* component

response to the request. This type of test step is particularly convenient to describe communication protocol messages that have data fields of fixed or variable length and inferred content (e.g. checksum); *ReceiveDataTestStep*, to support receiving arbitrary data in binary or text formats; and *SendDataTestStep*, to support sending arbitrary data in binary or text formats. A message library (*MessageLibrary*, *MessageFieldLibrary*) helps the definition of command/response communication protocols messages (primary/secondary communication). It is important to emphasize that the user can define his/her own type of protocol message format, as mentioned in *FormattedMessageTestStep*. This feature allows to test any kind of START/STOP protocol for serial asynchronous communication using RS-232 or Universal Serial Bus (USB) interface standards. All these entities are maintained by the main structure, i.e. *TestProject*.

The main functionalities of EXTFV are: (i) automated generation of Test Case Specifications and Procedures artifact in PDF format. This artifact contains the test items (what will be tested), the test cases related to a test item, and a description of the test cases where each test case is composed of a set of test steps. In the case of a *FormattedMessageTestStep*, each test step is composed of test input data/expected response in the message format defined by the tester; (ii) automated generation of Test Report artifact in PDF format. The Test Report is a document which contains a summary and a detailed report of test cases execution. In the detailed description and considering the *Format-*

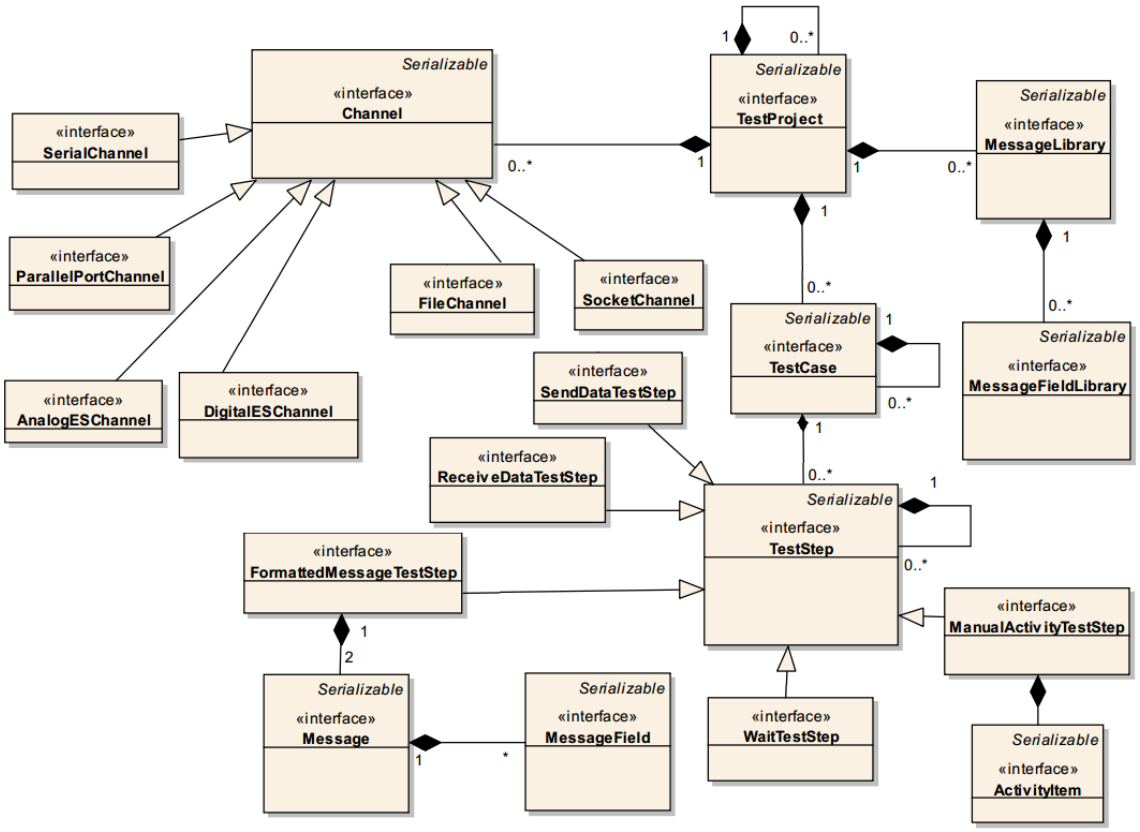

Figure 2. The *Entity* component: data model

tedMessageTestStep, in each test step appears the data sent from EXTFV to the IUT and the actual result sent from the IUT and received by EXTFV, in the message format defined by the tester. Not only Test Reports but also Test Case Specifications and Procedures can be very large considering system and acceptance testing, and their automated generation is an important feature for testing complex embedded software; (iii) automated execution of test cases; (iv) automated comparison between the actual and the expected results. Although the tester must manually input into the tool the expected result to certain test input data, EXTFV provides the automated verdict (pass, fail, problem) by comparing actual with expected results. If they differ, EXTFV highlights the problem in its main Graphical User Interface (GUI) and also adds such information in the Test Report. Hence, EXTFV has an automated *Solved Example Oracle* [Binder 1999] implemented; and (v) Data exporting and importing (Test Project) to XML format.

Technologies used to develop EXTFV were the NetBeans platform, Java, PostgreSQL database management system, and Input/Output (I/O) libraries such as the open source RXTX library, which includes native libraries for Windows (rxtxSerial.dll), and a commercial library, DT-OpenLayers, for a specific Data Acquisition Board (DT9804).

3. Usability

Due to space limitations, it is not possible to show any GUI of EXTFV. All interfaces of the tool, as well as instructions for installing and using it, can be seen in the the EXTFV's User Manual [INPE 2013] which is available on http://www.lac.inpe.br/valdivino/extfv.html.

The main steps to use EXTFV are briefly described below. The first step (S1) is to create a Test Project. After that, a communication channel must be created (S2). As mentioned in Section 2, EXTFV supports five types of communication channels to allow system and acceptance testing for embedded software.

The next three operations are particularly useful if test steps of the type *FormattedMessageTestStep* are going to be selected. The message library allows the creation of message templates. Thus, the third step (S3) is the creation of a message template and its addition in the message library, the fourth step (S4) is the addition of data fields to a message template (previous action), and, after that, the tester must fill in the data fields of a message template (S5). Having created these message templates in the library, it is very easy to reuse them in different test steps in various test cases.

Then, test items must be created (S6). A test item defines which characteristic/functionality of the IUT will be tested. The next action is to create a test case and relate it to a test item (S7). It is important to stress that 1 test item might have more than 1 test case related to it. In the sequence, test steps must be added to a test case (S8). As mentioned in Section 2, the user can select among five types of test steps. There are some interesting actions that the user can associate with a test step such as to make the same test step be automatically executed multiple times where there should be a time interval, which may also be defined by the user, between an instance of execution and the next one.

Next action is the execution of test cases (S9). Test cases can be executed one by one or the entire test suite can be executed at once. It is even possible to run a single test step of a test case but without generating report of test execution. At the end of the run (test case or full test suite) a dialog *Verdict* appears which allows the closing of execution, where the tester can assign a verdict. Even if in EXTFV exists a *Solved Example Oracle*, the tester has the flexibility to define the final verdict of the test case or the entire test suite.

The form of execution of test cases depend on the context of system and acceptance testing. For example, if the process of running system and/or acceptance test cases is at the beginning, in the early tests, it is interesting to try an execution of each test step of a certain test case to investigate possible communication problems (cabling) or even configuration of the IUT and/or EXTFV. As the process continues, one can start the execution of each test case, in accordance with the Test Case Specifications and Procedures artifact. In the case of regression testing, where part or even the entire test suite should be executed again, the tester can choose to run the full test suite at once (possibly removing some test cases that need not be performed again).

The last action is the generation of test process documentation (S10). EXTFV allows to easily and automatically generate Test Case Specifications and Procedures and Test Reports artifacts in PDF format. Again, due to space limitations and since Test Case Specifications and Procedures and Test Reports are usually large, it is not possible to show examples of such documents. However, examples of such documents generated considering real case studies for space domain can be seen on http://www.lac.inpe.br/valdivino/extfv.html. There are some other steps that the tester can perform such as data exporting and importing (Test Project) to XML format [INPE 2013].

4. Conclusions

This paper presented EXTFV, a tool that allows automated test case execution for embedded software but taken into account system and acceptance testing. In such testing levels, the amount of generated test cases can be considerably large, and manual execution of these may impact on the project software schedule. Not less important is the ability of the tool to automatically generate documentation related to the V&V process (in this case associated with software testing): Test Case Specifications and Procedures and Test Reports artifacts. Moreover, EXTFV has an automated *Solved Example Oracle* making it also part of the test results evaluation activity.

Ease of use is another relevant characteristic of EXTFV. If a tool needs to be used in practice by professionals from industry and academia, it is necessary that its operation is easy and intuitive. EXTFV follows this philosophy by means of its user-friendly interfaces. We have been applied EXTFV to case studies of space domain but certainly it can be used for any embedded software application that has communication via serial interfaces RS-232 or USB, parallel port, and Transmission Control Protocol (TCP) or User Datagram Protocol (UDP) sockets.

Future directions include improving EXTFV to support other modes of communication. Currently, EXTFV only supports the primary/secondary communication mode. Our goal is to make the tool to support services of the European Cooperation for Space Standardization (ECSS) - Packet Utilization Standard (PUS). Another effort is aimed at allowing test steps with conditional actions such as IfTestStep, WhileTestStep, enabling additional flexibility in the programming of test steps.

References

Binder, R. V. (1999). *Testing object-oriented systems*. Addison-Wesley Professional, USA. 1248 p.

Delamaro, M. E., Maldonado, J. C., and Jino, M. (2007). *Introdução ao teste de software*. Campus-Elsevier. 408 p.

INPE (2013). *protoMIRAX-185000-OPE-025: EXTFV – Manual do Usuário*. INPE.

JUnit.org (2013). *JUnit*. Available from: http://junit.org/. Access in: May 15, 2013.

Mathur, A. P. (2008). *Foundations of software testing*. Dorling Kindersley (India), Pearson Education in South Asia, Delhi, India. 689 p.

QA-Systems (2013). *Dynamic testing with Cantata: automated and easy*. Available from: http://www.qa-systems.com/cantata.html. Access in: May 15, 2013.

Santiago, V., Silva, W. P., and Vijaykumar, N. L. (2008). Shortening test case execution time for embedded software. In *Proceedings of the 2nd IEEE International Conference SSIRI*, pages 81–88.

Santiago Júnior, V. A. (2011). SOLIMVA: A methodology for generating model-based test cases from natural language requirements and detecting incompleteness in software specifications. Thesis (PhD in Applied Computing), INPE–São José dos Campos, 264 p.

St.Clair, J. and Kumar, A. (2013). *CUnit Testing Framewotk*. Available from: http://sourceforge.net/projects/cunit/. Access in: May 15, 2013.