Available online at www.sciencedirect.com

ScienceDirect

IFAC Papers Online CONFERENCE PAPER ARCHIVE

IFAC-PapersOnLine 49-30 (2016) 071–074

Developing and Testing Software for the 14-BISat Nanosatellite

Rogerio Atem de Carvalho, Milena S. de Azevedo, Sara C. M. de Souza, Galba V. S. Arueira, Cedric S. Cordeiro

Centro de Referência em Sistemas Embarcados e Aeroespaciais (CRSEA), Instituto Federal Fluminense (IFF), Campos/RJ, Brazil (Tel: 55-22-2737-5691; e-mail: ratem@iff.edu.br).

Abstract: The aim of this paper is to present the development of the software for controlling one of the scientific payloads of the 14-BISat nanosatellite, the Flux-Φ-Probe Experiment (Fipex). This satellite was developed by the Instituto Federal Fluminense (IFF) as part of the multinational QB50 mission for the Low Termosphere characterization. In order to provide the necessary support for developing this software, a software toolset was developed in order to to form an agile, integrated development environment for the C/C++ languages for microcontrollers.

Keywords: Embedded systems, Artificial satellites, Automatic testing, Code generation, Finite state machines.

1. INTRODUCTION

Botef (2015) lists Agile Methods as one of the key components for aerospace manufacturing competitiveness. In this direction, Grenning (2011, p. 2) affirms that Test Driven Development (TDD), specifically, is an effective way of weaving test into the fabric of embedded software development. However, a recent case study by Berger and Eklund (2015) showed that one of the main constraints for achieving a faster time-to-market product development is an inflexible test environment that inhibits fast feedback to the changed or added features.

Additionally, Youn at al. (2015) bring attention to the fact that the ever growing pace of software use in airborne systems in parallel with the modern software development and verification technologies and methods makes certification standards for safety-critical systems, such as DO-178, become inadequate in certain points. In that direction, the specific use of TDD for IEC-61508 standard for safety-critical software is analyzed by Ozçelik and Altilar (2015), who conclude that the technique supports most of the standard and do not conflict with the rest of it.

This finding suggests that there is room for further development of Agile practices and tools for embedded systems, in special those based on TDD. In this direction, this paper aims at briefly presenting the experience of using a Tools & Techniques Set (TTS) for developing complex embedded systems called VALVES (VALidation and Verification for Embedded Systems), named after the device that controls the flux of fluids in pipes, as an analogy to the process of controlling the flux of source code into production. Valves is an evolution of the basic TTS presented by Carvalho et al. (2014, 2015, and 2016), and is composed by an integrated set of elements that forms a cohesive toolset for the C/C++ languages for microcontrollers and other embedded systems platforms: (i) a tool for modeling Finite State Machines, (ii) a Domain Specific Language (DSL) for automated test generation, build, and deployment (iii) a mechanism for supervised FSM execution, (iv) a toolchain for compiling and debugging, (v) a version control system, and (vi) Continuous Integration process and tool. The elements (ii), (iii), and (vi) were developed by the authors, while the others were integrated to the previous to form the TTS.

The toolset was used to develop the controlling software for one of the 14-BISat cubesat's payloads, the Fipex (Flux-Φ-Probe Experiment), from Dresden Technological University (Germany). The development of the payload occurred in parallel with the development of the satellite's control software, using a Interface Control Document (ICD) as an artifact for synchronization.

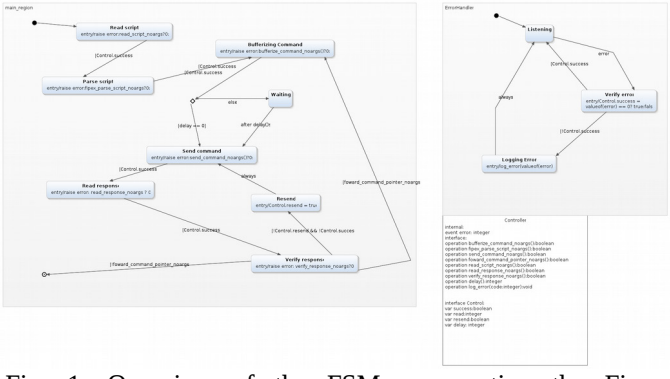

Fig. 1. Overview of the FSM representing the Fipex Controller behavior (left), and Fipex behavior (right).

The following topics briefly present the Valves' Validation and Verification chain used to develop Fipex Controller, the payload's controller software, together with remarks on this specific development case.

Peer review under responsibility of International Federation of Automatic Control.
10.1016/j.ifacol.2016.11.128

2. VALIDATION CHAIN

The Validation Chain represents the part of the TTS responsible for checking if the system is in accordance with the requirements, therefore, it is a type of checking that occurs at higher abstraction levels. Finite State Machines (FSM) are used by the TTS to provide a means of modeling and simulating embedded systems behavior. In order to design the FSM, a graphical modeling tool is used as a basis for providing the validation process for Valves. In that direction, Yakindu Statechart Tools was the environment of choice, given that it is an integrated modeling environment for the specification and development of reactive, event-driven systems using the concept of FSM.

Although Yakindu is able of simulating the execution of FSM in its modeling environment, which is usually based in a PC machine, a plugin for the Eclipse Integrated Development Environment (IDE) was developed for executing the automated tests in a supervised way, in order to provide Validation in the target hardware, in this case a microcontroller. In other words, the FSM is modeled by the user using a PC, however its source code is generated to run in the target platform, where it is properly ran and tested. Therefore, Valves substitutes the usual simulation in a PC computer by the real, controlled execution in the target hardware.

Running the FSM in a controlled way means to fire its events “by hand”, when the user clicks with the mouse pointer on the transitions of its visual representation. Of course, in the case of time-constrained transitions, the user has no direct control over the specific behavior of the machine, and they are fired automatically, as expected. In other words, it was implemented a Validation environment using FSM as an ubiquitous language, or a common language used by all stakeholders.

One of the main requirements for this functionality was that it should generate the smallest overhead possible, in order to run on microcontrollers. Thus, it was necessary to keep most of the testing machinery outside the target hardware. With regard to achieving this goal, while maintaining the TDD premise of “tests are also documentation”, a Domain Specific Language (DSL), named Handwheel, was created in order to automat and document the codification of tests, which are ran in the hardware where Yakindu's rans. Handwheel provides a very high level of abstraction for writing tests, thus making this task easier to accomplish, even for non-experienced coders. Tests are automatically translated into C code with very small footprint. Thereby, it is possible to treat the target hardware as a black box, consuming a minimum of memory and processing budget, while allowing sophisticated, high-level testing, by means of an abstraction level usually found only in Information Systems.

An interesting byproduct of this solution is that auto-testing scripts can be created and used in the production environment for checking the system's health and send warnings and reports to the controlling element. This is possible because Valves establishes a protocol for sending instructions to the target system, and, accordingly, this same protocol is used for testing.

Fig. 1 presents two FSM modeled using Yakindu: the first represents the behavior of the Fipex Controller software, developed by the authors, while the second one represents the behavior of the Fipex device, and was used to simulate it.. Fig. 2 shows the code written in Handwheel for testing the first FSM, as well as its successful execution during a testing cycle.

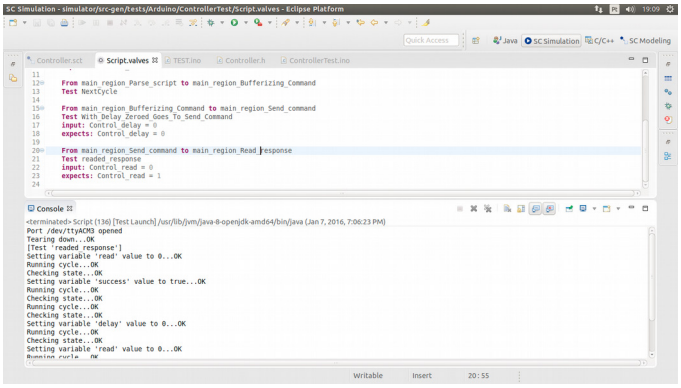

Fig. 2. Handwheel code used for testing and executing (simulating) the Fipex Controller FSM.

3. VERIFICATION CHAIN

It is at the Verification level that the hardware platform of choice will determine the use of specific tools. For instance, ARM platforms with Linux-based RTOS (Real Time Operational System) distributions allow a “comfortable” use of C++ with more complex supportive libraries, such the ones for I/O, while pure C with simple test facilities are more suitable to microcontrollers such as MSP430 – the processor of 14-BISat’s Onboard Computer (OBC). Valve’s verification chain is formed by a series of open source tools for compiling and debugging C/C++ code.

It is important to note that Handwheel is not only the language for simplifying tests, but also the connection element between the Validation and Verification tasks, given that it is also used to build and deploy the generated code. For doing so, it is supported by specific makefiles developed by the authors in order to simplify compilation, building and deployment of the code onto the target hardware. For the Fipex Controller, *gcc* was used to compile and link code in pure C, and *mspdebug* was used to upload the executable code into MSP430’s flash memory.

Given that 14-BISat is a multi-year project, the development of the software for its payload involved different people contributing to it. Hence, it was necessary to provide ways for (a) control the evolution of the software and (b) keeping each new version of it consistent. Valves integrates into its

toolset open source tools for providing code versioning and Continuous Integration (CI), which provides automated deployment when the build is successful. These controlling tasks are integrated around the version control system of choice, which is GitLab. CI is the practice of merging all working copies of the code, from the various developers, to a single shared source code repository, several times a day, making a complete build each time, in order to keep the software consistent.

A very interesting feature of Valves integration scripts is the fact that all these tasks can be done in different machines distributed through a TCP/IP network. In fact, it is possible to have many programmers developing for the same piece of hardware, which can be plugged to a remote server anywhere. Valves provides the ways for keeping the code correctly tested, versioned and consistent, even if the programmers and the hardware are not in the same place, which means economy of resources and bigger consistency of results.

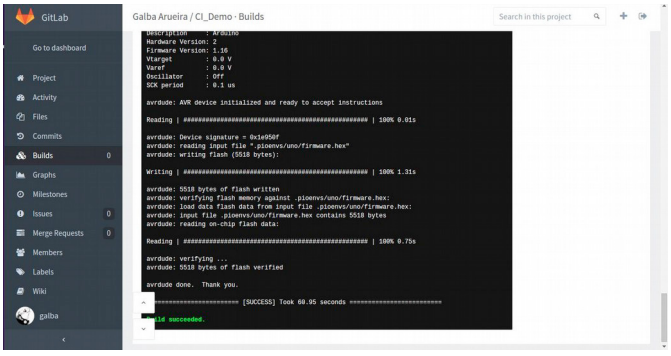

Fig. 3. Building Fipex Controller source code using *gcc*, properly tied to GitLab.

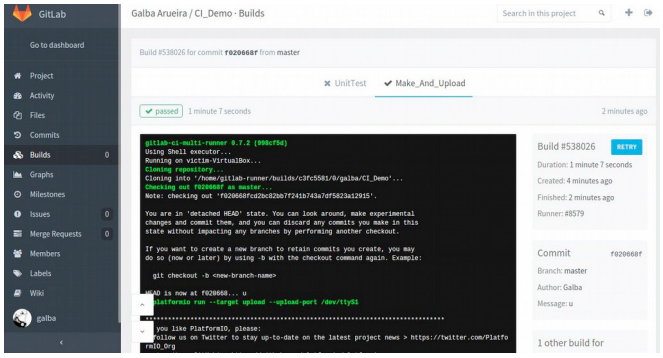

Fig. 4. Deploying Fipex Controller executable code to a remote MSP430 instance, after a successful build.

4. LESSONS LEARNED

14-BISat is developed at the Reference Center for Embedded and Aerospace Systems of the Instituto Federal Fluminense, Brazil, together with Universidade do Porto and Tekever S.A. (Portugal). It is a 2U nanosatellite, part of the QB50 mission, which will carry a series of experiments for investigating the Low Thermosphere.

Together with its bigger brother, GAMASat, a 3U satellite, 14-BISat will be the data server for an *ad hoc* network in the space using GAMALink, which is a S-Band Software Defined Radio and dedicated protocol. GAMALink has two modes, the Intersatellital Link (ISL) mode, which makes the communication between two spacecrafts, and the space-ground mode, responsible for exchanging data with the ground segment. Given the QB50 mission demands, it is necessary to have at least two hosts to serve as data backup for the other satellites taking part of the GAMALink network, 14-BISat and GAMASat.

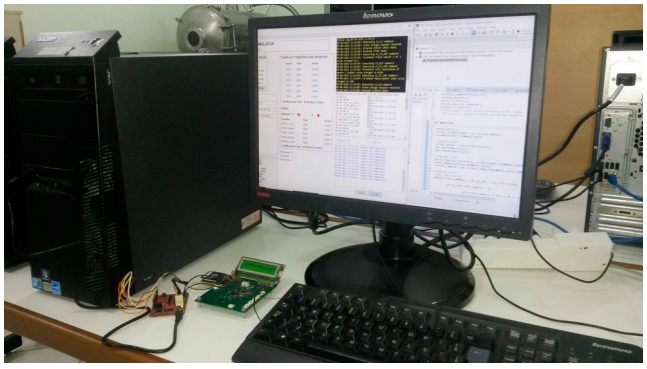

Fig. 5. 14-BISat's prototype OBC with a display for debugging purposes and an external flash memory, connected to an auxiliary board for programming. On the PC's screen, the Fipex Simulator software (left) and part of Fipex Controller source code (right).

Being a host makes 14-BISat communicate a lot, which in turn makes it use a lot of power, which is one of the bigger, if not the biggest, issue in nanosatellites operations. For this reason, every effort must be done to reduce the power budget for 14-BISat, including using the minimum processing power required to achieve its operations. One of the direct consequences of this demand for minimum use of power is the necessity for writing highly efficient code, in order to reduce the number of machine (CPU) cycles.

At this point it is necessary to discuss the use of the technique of modeling the software and then generating code based on the models, as offered by Valves. Modeling embedded systems using FSM is a very powerful way for validating the requirements as well as for avoiding error conditions such as deadlocks and starvations. However, in order to guarantee the advantages of using FSM for modeling, it is necessary to generate the code for both executing and testing them. While Valves provided 14-BISat team with a very good program skeleton for the Fipex Controller, when it came to write algorithms with reduced processing and memory footprint, it

was necessary to do it by hand – as expected in any case, given that Model Driven Development (MDD) never delivered the old promise of eliminating all the work of writing code.

In that regard, the first lesson learned was that although Valves can generate the base code with really small footprint, it only generates the program skeleton with its main flow, formed by the FSM. Although some features, such as timers, are commonplace in embedded systems, and therefore could also be automatically generated, developers wrote the rest of the code by hand in order to optimize it to the extreme, given that the old programming adage "a good programmer is better than a good compiler" is still valid. While it is possible to implement features for generating optimized code for a given platform, this type of feature is so tied to the a single processor, that many times it is not worth to implement. In other words, the effort applied for implementing the fully automatic code generation procedures directed to a given microprocessor would in the end supply a tool which would ask for so many parameters for the developer that it would be better for him or her to write the given piece of code by hand in most cases.

The second lesson learned was regarding the limits of automated testing for highly optimized embedded systems. Test Driven Development has a say similar to "good testing implies in no need for debugging", however, this is only possible when higher levels of code abstraction are possible, something that is relatively rare in embedded systems, specially those based in microcontrollers. Higher abstraction levels usually imply in generic code, which in turn implies in more parametrization and extra levels of indirection, and, to some extend, some code bloat. Although compilers can reduce bloating in many cases, sometimes some processing and/or memory overheads are unavoidable and then could not be used indiscriminately by 14-BISat team. In other cases, some testing artifacts cannot simply be used at all. For instance, Valves provides a library for code doubles called Clover, which relies on Shared Libraries for simulating C-functions with specific desired behaviors. However, it is not possible to use Shared Libraries in most low power microcontrollers, such as MSP430, because these libraries can only be implemented on top of a complete operational system infrastructure. Still, it was decided not to use Assembly code, in order to keep the code as clear and testable as possible, with the cost of some overhead in very specific cases - careful benchmarking supported this decision.

5. CONCLUSIONS

This paper briefly presented how the software for interfacing with one the payloads of the Brazilian nanosatellite 14-BISat was developed, supported by a toolset that was also developed by the satellite's engineering team. This toolset, Valves, enables the concurrent and distributed software development, providing a means for cheaply and productively modeling and testing complex embedded software.

Valves is still a work in progress, although it already provides a very good basis for generating reliable embedded code. The effort for improving it now goes towards improving Handwheel for better supporting processing and memory footprint benchmarking testing.

Fipex Controller was successfully tested during 14-BISat's Flight Readiness Review (FRR) campaign, in a hardware in the loop environment, however, its final test is yet to come: the flight itself. Whatever the final results of this software development process, the main conclusion is that the development of Fipex Controller proved that Test Driven Development is still challenging for microcontrollers, with very few software and working examples to work on top of it, and Valves will make its way for contributing to this challenge.

REFERENCES

Berger, C.; Eklund, U. Expectations and Challenges from Scaling Agile in Mechatronics-Driven Companies – A Comparative Case Study. Agile Processes in Software Engineering and Extreme Programming, Springer, 2015.

Botef, I. "Frameworks for Efficient Aircraft Operations". Aircraft Engineering and Aerospace Technology: An International Journal, v. 87, n. 3, 2015.

Carvalho, R. A. ; Ferreira, H. S. ; Toledo, R. F. ; Cordeiro, C. S. ; Moura, G. L. Interfacing with the Science Unit: Preparing the Software Side. In: Proceedings of the 6th European CubeSat Symposium, 2014.

Carvalho, R. A.; Ferreira, H. S. ; Toledo, R. F. ; Azevedo, M. S. TDD for Embedded Systems: A Basic Approach and Toolset. Technical Report, 2015. arXiv:1507.07969v2 [cs.SE]

Carvalho, R. A.; Arueira, G. V. S. ; Azevedo, M. S. ; Toledo, R. F. Developing the Software for Cubesats in a Concurrent Engineering Environment: a toolset and case study. In: Second IAA Latin American CubeSat Workshop IAA Latin American CubeSat Workshop, 2016, Florianópolis.

Grenning, J. W. Test Driven Development for Embedded C, Pragmatic Programmers, 2011.

Ozçelik, O.; Altilar, D. T. "Test-Driven Approach for Safety-Critical Software Development". Journal of Software, v. 10, n. 7, 2015.

Youn, W. K.; Hong, S. B.; Oh, K. R.; Ahn, O. S. "Software Certification of Safety Critical Avionic Systems: DO-178C and Its Impacts". IEEE Aerospace & Systems Magazine, v. 30, n. 4, 2015.