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.

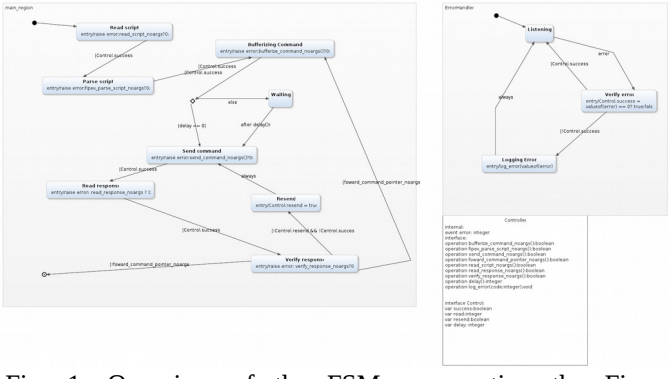

Fig. 1. Overview of the FSM representing the Fipex Controller behavior (left), and Fipex behavior (right).

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.

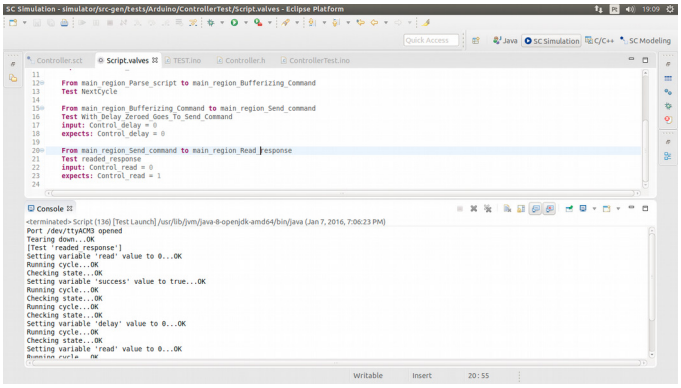

Fig. 2. Handwheel code used for testing and executing (simulating) the Fipex Controller FSM.

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