International Journal of Advances in Engineering and Management (IJAEM)
Volume 2, Issue 9, pp: 712-717 www.ijaem.net ISSN: 2395-5252

A Software Method for Telecommand and Telemetry Implementation of Aalto-2 CubeSat

Dr.C.Brintha Malar [1] Dr.K.Siva Sankar [2]
[1]*Assistant Professor, Udaya School of Engineering*
[2]*Associate Professor, Noorul Islam Centre for Higher Education*

Date of Submission: 15-11-2020 Date of Acceptance: 30-11-2020

ABSTRACT: This work concentrates on the development of telecommand and telemetry handler software for a 2 kg Aalto-2 nanosatellite, currently scheduled for launch in December 2016. The satellite is part of the international QB50 thermosphere mission and it is developed by Aalto University in Espoo, Finland. The telecommand and telemetry (TC/TM) handler, in charge of com- medications, is one of the most important systems of satellite software, which is executed by On Board Computer (OBC) software. In this thesis, the TC/TM handler subsystem is designed, giving it a special attention in maintaining sim- plicity and reliability. The design process is started with the derivation of re- quirements and constraints.
The software is implemented for Free RTOS, an open-source real-time operat- ing system, which is selected as operation environment of the satellite main OBC software. The library provides functions of dynamic function call in C. The UHF driver library handles incoming and outgoing low-level communications protocols, and the Coffee File System implements storage management.
Keywords: Aalto-2, TC/TM Handler, dynamic dispatch, dyncall, PUS, scripting, CubeSats, Free RTOS, concatenative language

I. INTRODUCTION

The communication subsystem is an essential element of every spacecraft. During the early days of space technology, spacecrafts had very primitive communication subsystem with simple periodical science and housekeeping data telemetry transmissions. Research and development was needed in order to gain more control and reliability, which is vital for missions such as space exploration, global position systems or weather survey. Decades have passed and technology has developed, so today the ground station gives full control over satellites. Nowadays, new ambitious missions are being planned that require more advanced, smarter, faster and more autonomous designs of hardware and software.

The data handling software usually runs in the On Board Computer (OBC). It is responsible for decoding incoming messages, called telecom- mands (TC), and encoding outgoing messages, which are called telemetry (TM) packets. The whole software element is called the Telecommand and Telemetry (TC/TM) Handler. Usually, on board software is developed for a specific platform and architecture, which means it is difficult to reuse in other spacecrafts. Therefore, this TC/TM software (as most of the satellite software) is tailor-made for certain spacecraft in order to give better design and implementation. The following work develops new approaches for communications software to achieve the requirements of a specific spacecraft mission. This new approach is done with concatenative programming language definitions and characteristics.

The goal of this work is to design and implement a software architecture for telecommand and telemetry handling, as part of the on board software of the Aalto-2 satellite, by utilizing concatenative programming model. The software shall fulfill the requirements and restrictions set by satellite modules hardware and mission specifications.

II. BACKGROUND

The work is done for the Aalto-2 CubeSat satellite, which belongs to the international QB50 satellite constellation. The next chapter gives a short overview of CubeSat satellites, their communication implementations and the QB50 project and the Aalto-2 satellite.

2.1 Small satellites and CubeSats

Small satellites are defined as those spacecrafts that weigh below 500 kg. The concept was proposed to reduce the building, developing and launch- ing cost. Among the categories in small satellites, the nanosatellite have become the most popular, due to affordable cost and the release of the CubeSat standard.

International Journal of Advances in Engineering and Management (IJAEM)
Volume 2, Issue 9, pp: 712-717 www.ijaem.net ISSN: 2395-5252

The CubeSat standard was proposed by professor Jordi Puig-Suari from California Polytechnic State University and professor Bob Twiggs from Standford University in 1998. Their goal was to grant graduated students the opportunity to design, implement, build and operate spacecrafts simi- lar to the first one of history: the Sputnik. In 1999 the CubeSat standard was released and it become a popular platform to develop satellites. Nowa- days, the standard is used widely by university teams across the world. CubeSats weigh between 1 kg to 10 kg and are shaped as cubes (10 x 10 x10 cm3), with a typical weight of 1,3 kg per unit (1U-CubeSat). This new satellite class was embraced due to its low developing cost, simplified design and cheap launching cost, which is normally performed alongside other larger satellites. CubeSats usually are similar with each other, be- cause they follow the same standard, often using similar (even identical) components (commercial components adapted or certified for space) and designs. CubeSats also usually use amateur radio bands (Very High Frequency, VHF; and Ultra High Frequency, UHF) for communication. As development cost for nanosatellites is relatively low, developing CubeSats allows hardware and software experimentation that is too expensive for larger satellites. However, new problems are always found as well as solutions that led to new discoveries and techniques. [13]

III. IMPLEMENTATION OF COMMUNICATION

The TC/TM architecture design that uses concatenative language features combined with native C and FreeRTOS. The designed architecture is divided in two structures: the Listener and the Interpreter. The first one, the Listener, is designed and programmed as finite state machine that communicates with the Interpreter, which is programmed as Turing machine.

3.1 TC/TM Handling Implementation

The telecommunication handler of the Aalto-2 is built in two different parts: the Listener and the Interpreter.

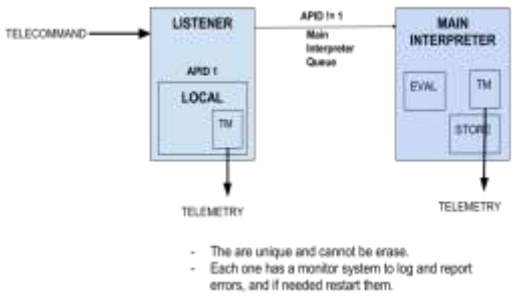

Figure 3.1: TC/TM Handler Architecture

In figure 3.1, the two software structures are being represented: the first one is the Listener, whose function is to receive and check incoming telecommands. Additionally, Listener implements a few services procedures. The second, is the Interpreter, which interprets a command or group of commands, routing them to the targeted module to be executed.

Both pieces use external C libraries that implement different key procedures. The most relevant one is the Dyncall library. The library is a group of files which implements run-time function calls in C. Another important library is the Stack library. It implements C functions and structures to create and manage a byte stack. Additionally, more libraries where build in order to meet the requirements, among those libraries the most relevant are, the System Monitor, which is a set of functions to that implements task watchdog. Finally, the file system support libraries, the Coffee File System (CFS) and the Raw File System (RAW); they both implement methods to manage the OBC file system.

Furthermore, there are some concepts/variables that are needed to be exposed before Listener and Interpreter function explanation.

First of all, they Sat Function variable and Format variable: the first one contains the satellite C function address, meanwhile, the second one stores the signature arguments. Both variables are stored in an encoded dictionary-like array that has all the satellite functions a specific APID can execute. A second important variable is the StackByte. The variable stores the location and position of the SatFunction arguments. When a command is called, its arguments can be located either inside the telecommand application data field that encoded that command or inside the satellite stack. Additionally, the last bit defines if ground station request immediately telemetry of the result after telecommand execution.

International Journal of Advances in Engineering and Management (IJAEM)
Volume 2, Issue 9, pp: 712-717 www.ijaem.net ISSN: 2395-5252

Finally, the C function evaluate single takes SatFunction, Format and a stack as arguments and it is tasked to evaluate and execute the encoded telecommand, returning a result of this execution as shown in figure 3.2.

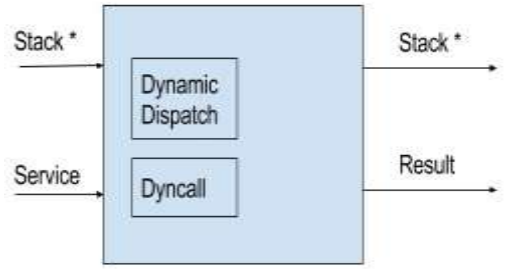

Figure 3.2: evaluate single block diagram

3.1.1 The Listener

This first piece of the TC/TM handler is the receiving of a packet and checking if it is a correct telecommand. These procedures are performed by a Free RTOS FSM task called Listener.

Figure 3.3: Listener Finite State Machine

The designed software can be depicted, as seen in figure 3.3, as a state machine with four basic states:

Idle: is the starting point where the Listener expects a command from the GS. The satellite receives byte per byte; therefore, the Listener remains at this state until there is at least one byte encoded as the hexadecimal value 0x18 (which corresponds with the first byte of the Packet ID format field). Finally, the Listener understands that this first byte may be the beginning of a TC. Then it changes the state to Header Processing. Otherwise, it remains at Idle state.

Header Processing: receives the rest of the header bytes and creates a command structure with APID, service, subservice and packet length. Then it checks if these parameters are inside the correct boundaries of the satellite definitions (e.g. maximum packet length, APID maximum number, etc). If all of them are correct, the Listener knows that the TC has a correct Header structure, and state is set to Data Processing. Otherwise is set to Recovery state.

Data Processing: is reached when the Listener has a correct PUS Header. While at this state, the Listener receives the rest of the packet (depending on the packet length value) and sets the following variables: Stackbyte, application data and PEC. Then it calculates a new checksum, and checks it with the one stored in PEC variable. If both variables are identical, then the Listener is sure that the message is a correct PUS format TC. Once a TC is correctly identified, the command can follow two different paths depending on APIDs value. If APID is 1, it means this command is meant for the Listener and will be execute immediately after. On the other hand, if APID is different from 1, the TC is meant for other application processes inside the satellite, and telecommand message is sent to the Main Interpreter. Either way, state variable is set to Idle. If the Listener has found that this message PEC is different from the calculated one, or an error happened during variable assigning process (e.g. malloc failed), the message is discarded and state is set to Recovery.

Recovery: is the state triggered when a problem has occurred in Header Processing or Data Processing. There are many different situations that triggered this state (e.g. Wrong value in TC, timeout reached, run out of RAM, etc). Despite the different situations, the Listener Recovery has one execution direction: first discard the processed TC (if wrong), then, if there is more byte data waiting to be taken, tries to find if there is a second TC by searching the 0x18 hexadecimal value. If this value is found, Header Processing state is set, so the Listener can process a possible TC. Otherwise it reaches the end of the buffer. Despite the encounter situation, the Listener generates a failure report (an error happened), and at the end, it resets local variables and state to Idle.

Finally, the Listener task can execute some telecommands locally to manage the satellite beacon, state reporting tasks and Interpreter management

3.2 Aalto-2 TC/TM Chomsky classifications

TC/TM software was classified using the Chomsky hierarchy. Now, the same analysis should be

International Journal of Advances in Engineering and Management (IJAEM)
Volume 2, Issue 9, pp: 712-717 www.ijaem.net ISSN: 2395-5252

performed with the TC/TM implementation of Aalto-2. As explained in previous section 3.1, the TC/TM is composed by two structures: the Listener behaves as a regular grammar, that receives TC that can change the state, in other words, a type 3 machine. On the other hand, the Interpreter employs memory, rising its category to type 2. This memory is structured as a stack-byte which stores elements of any type. Additionally it can do composition of functions by employing quotations. As this stack can have multiple modification from functions, other TCs and the interpreter, it can be seen as a random access memory. Therefore, type 1 is reached, because the memory can be modified. Finally, if overwritten is allowed, TCs can delete and write in the memory, theoretically having infinite virtual memory and reaching type 0. Summarizing, the TC/TM software is a finite state ma- chine that communicates with a Turing machine.

But the overall TC/TM structure is the combination of both models. Each TC is sent as a message that is first process by the Listener, then this command is processed and execute by the Interpreter and then it replies back. The process could be seen as FSM of three states (receive, interpret and reply). But that is a short description, because it does not take into count that interpreter has memory. Watching the big picture, the Listener FSM is just a function executed at the start whenever a TC is received, meaning that the TC/TM is categorized as the Interpreter model, which is type

IV. TESTING

This chapter introduces the testing methodologies used to detect problems and bugs as well as the modifications made. Also, these procedures are validating and verifying the implemented code, checking that all required functionalities are implemented.
The two chosen techniques are: the Software in the loop (SIL) testing and the Overall testing.

4.1 Software-in-the-loop testing

A designed software will not be qualified until implementation and test- ing is concluded. Therefore, software in the loop testing techniques, grants testing and correction at many levels. The test was performed connecting a real OBC module to the workstation using a serial RS-232 connector. Communication with the development board is performed with a software program called Hercules SETUP utility. The Hercules is a HW-Group serial terminal port (RS-232) which can send raw hexadecimal commands. Also, an extra bus connection was needed between the CCS program (at the workstation) and the OBC to flash and debug the software. [9] Unfortunately, the Hercules program has not a very good interface and was used for small and simple tests that required very little pre-processing.

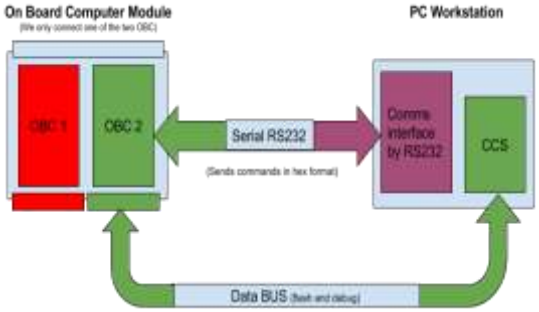

Figure 4.1: Software in the Loop Setup

The testing software utility used as substitution of the Hercules SETUP program was a Python Domain Specific Language (DSL) program, that simulates a ground station software. Moreover, the Monitor System was enabled for the Listener so it could be used for detecting execution problems as it will do once deployed. It uses log files to record any violation of the state machine rules and the level of seriousness of it.

4.2 Overall test

The On Board Computer software manages several tasks at the same time. These functions are synchronized through the task scheduler, which regulates the running tasks depending on the priority and the readiness of each. This test is performed using the maximum number of task/procedures should be able to manage at the same time by the satellite software, checking the adaptability and performance of the developed software with the rest of the system.
For this testing was chosen critical situations. Short commands are un- likely to encounter major problems, on the other hand, sending large commands and files can become a quite serious problem, e.g. Sending OBC boot loader command or payload scripts. Therefore, the testing procedure consisted in sending consecutive big dummy files through the radio link and verification of the files were not corrupted. The inverse testing was also performed; instead of uploading a large file, a short telecommand made a request to download a large dummy file. The telecommand generation and telemetry reception was left to the Ground Station software, which was implemented, as explained using a Python DSL.

International Journal of Advances in Engineering and Management (IJAEM)
Volume 2, Issue 9, pp: 712-717 www.ijaem.net ISSN: 2395-5252

The following paragraph shows the script codes performed tests using the GS software and their results:

The different tests performed using the GS module are: uploading a file (File Upload (slot, filename)), download some bytes from a file (File Download(slot, size, filename)) and check for errors from the monitor system (check monitor()).

Options:

TC = Telecommand.

TM = Telemetry requested. SB = StackByte.

IN = Input. OUT = Output.

RET = allows to connect the script result values with the Python environment.

I32, UIN8, etc = variable type.

FileUpload(slot,filename) creates a set of commands to upload a file into the satellite storage system. The ground station shell will look like this:

TC ToStackString SB[1 2] IN[ARRAY[FILE "FILENAME" UI32 SIZE]] TM;

TC MakePointerToLast TM;

TC AppendRawFile SB[1 2 16] IN[ARRAY[UI32 SLOT UI32 SIZE]] TM; TC DROP SB[16] TM;

The first command puts the input data in the stack. The second command creates a pointer to the beginning of the previous string. Third one sends the slot and size of the file, writing into filename location. Finally, it sends a request to drop the string address left. Figure 4.2 represents the stack movement

Figure 4.2: Stack variation per command executed

V. CONCLUSION

Among all possible telecommand and telemetry handler software architecture design approaches, an unconventional one was chosen for the Aalto-2 satellite. The goal was to implement and test validation of the telecommunication requirements for Aalto-2 satellite and verification of TC/TM Handler software. The TC/TM Handler was programmed for the Aalto-2 Satellite OBC as Chapter 3 explains. The software in the loop testing and the overall testing were performed to verify and validate the code.

Firstly, a novel approach was developed for processing telecommands (evaluate), which uses a concatenative programming oriented design implemented in C/Free RTOS. Analyzing these implementation, it is found belonging to a Type 0 Chomsky grammar.

Secondly, a simple finite state machine was build to support the Interpreter structure call Listener, which function is to received, check and route in- coming commands. The analysis of these structure reveals that it is a Type 3 Chomsky grammar.

Combining both structures, a TC/TM Handler is build, which satisfies the Aalto-2 communication requirements and functionalities.

Some recommendations and suggestions for further work are made after the latest testing results:

The TC/TM software is being executed inside Aalto-2 OBC software and it will fulfill its purpose once the satellite operates and performs its mission successfully in the real environment.

The ground station equipment needs the interaction of a person-user to sent and received most of the commands. Therefore, further work in automatising the satellite-ground station communication could be done.

The satellite autonomy system is simple. The user can upload scripts and schedule them in time. Improvements (such as implementing other PUS services or enabling quotations for scheduling) can be done to achieve more satellite autonomy and better overall performance.

Scheduling are not implement. There is a simple implementation that per- forms scheduling but it is needed a more complete function. However, building this functions are not a big problem, because the TC/TM model facilitates the scheduling execution. In other words, the prime materials are there and only needs someone to do it.

REFERENCES

[1]. CCSDS 100.0-G-1, Green Book, Telemetry Summary of Concept and Rationale. CCSDS, Frascati, Italy, 1987.
[2]. CCSDS 200.0-G-6, Green Book, Telecommand Summary of Concept and Rationale. CCSDS, Frascati, Italy, 1987.
[3]. ECSS-E-70-41A. ESA Publications Division, Noordwijk, The Nether- lands, 2003.
[4]. CCSDS 132.0-B-2: TM Space Data Link Protocol. Blue Book. Issue 2. CCSDS, National Aeronautics and Space Administration, Washing- ton DC, USA, September 2015.
[5]. CCSDS 232.0-B-3: TC Space Data Link Protocol. Blue Book. Issue 3. CCSDS, National Aeronautics and Space Administration, Washing- ton DC, USA, September 2015.
[6]. CHOMSKY, N. On certain formal properties of grammars. Information and Control 2 (1959).
[7]. EICKHOFF, J. On Board Computers, OnBoard Software and Satellite Op- erations. Springer, Institute of Space System, University of Stuttgart, Germany, 2011.
[8]. FACTOR -COMMUNITY. Factor Wikipedia. http://factorcode. org/, (accessed on March, 2016) 2016.
[9]. HW-GROUP. Hercules setup utility. http://www.hw-group.com/, (accessed on March, 2016) 2015.
[10]. JOVANOVIC , N. Aalto-2 satellite attitude control system. Master 's thesis, Department of Electrical Engineering and Automation, School of Electrical Engineering, Aalto University, Espoo, Finland, 2014.
[11]. JOV E´ , R. Contribution to the development of pico-satellites for Earth observation and technology demonstrators. PhD thesis, Department of Signal Theory and Communications, Universidad Polite´ cnica de Catalunya, Barcelona, Spain, 2015.
[12]. KUHNO, J. Aalto-2 uhf module report. Project Documentation, 2016.
[13]. LEE, S., HUTPUTANASIN, A., TOORIAN, A., WENSC HEL, L., AND MUNAKATA , R. CubeSat Design Specification, revision 10 ed., August 2007.
[14]. LLC, B. Concatenative Programming Languages: Forth, Postscript, Fac- tor, Cat, Hartmann Pipeline, Joy, Colorforth, Concatenative Programming Language. General Books LLC, 2010.
[15]. LTD ., R. T. E. Freertos api. http://www.freertos.org/, (accessed on January, 2016) 2004.
[16]. MAS SON, L. Recommendation for flight software implementation. Tech. rep., QB50, 2014.
[17]. NASA. Reference guide to the international space station: Utilization edition. http://www.nasa.gov/ ((accessed on September, 2016)
[18]. PRAKS , J., KES TILA¨ , A., HALLIKAINEN, M., SAARI, H., ANTILA, H., JANHUNEN, P., AND VAINIO, R. Aalto-1, an experimental nano satellite for hyper spectral remote sensing. IGARSS (2011).
[19]. PURDY, J. The big mud puddle: Why concatenative programming matters.http://evincarofautumn.blogspot.com (2012).
[20]. RIWANTO, B. A. Cubesat attitude system calibration and testing. Master 's thesis, Department of Electrical Engineering and Automation, School of Electrical Engineering, Aalto University, Espoo, Finland, 2015.
[21]. THOEMEL, J., SINGARAYAR, F., SCHOLZ, T., MASUTTI , D.AND TESTANI, P., ASMA, C., REINHARD , R., AND MUYLAERT, J. Status of the qb50 cubesat constellation mission. International Astronautical Congress (2014).