www.DeepakPublishing.com

Johl, S. et al. (2015): JoSS, Vol. 4, No. 2, pp. 357–369
(Peer-reviewed article available at www.jossonline.com)

www. JoSSonline.com

A Reusable Command and Data Handling System for University CubeSats

Shaina A. M. Johl and E. Glenn Lightsey

University of Texas at Austin, Austin, TX, USA

Abstract

As part of a series of CubeSat missions under construction at the Texas Spacecraft Laboratory (TSL), University of Texas at Austin (UT-Austin), a Command and Data Handling (C&DH) system has been developed with a centralized system architecture, with the goals of supporting a variety of mission requirements and reducing inter-mission reengineering. This system architecture uses one main flight computer controlling the actions and the state of the satellite. A commercial off-the-shelf (COTS), system-on-module (SOM) embedded computer running a Linux environment was used as the platform for the mission software. An integral component of the C&DH system development is the flight software (FSW), written in C++ using the object-oriented (O-O) architectural style. It is structured as a state machine, controlling the transitioning of the satellite between its operational modes that define its actions and behavior. The various testing procedures that are performed on the FSW, the C&DH system, and the integrated satellite are described herein. These procedures include traditional software tests, such as black box, glass box, and unit testing, as well as formal spacecraft tests, such as Command Execution and Day-in-the-Life (DITL) tests. The design of the flight software and associated hardware are integral components of the current missions at the TSL which, when flown, will be some of the most operationally complex CubeSat missions attempted to date.

1. Introduction

As the CubeSat industry grows, there is a larger demand for CubeSats to handle more complex mission and operational requirements. These requirements flow down to affect the Command and Data Handling (C&DH) subsystem of the satellite, which acts as the brain of the spacecraft, consisting of the hardware, including the main flight computer, and the software that controls the operations of the satellite.

The Texas Spacecraft Lab (TSL), a university satellite lab at the University of Texas at Austin (UT-Austin), is currently performing several different CubeSat missions. More complex mission requirements are placing increased emphasis on the design of the C&DH system. The experience gained by past missions has also made obvious the need to create a re-usable C&DH system for the lab's CubeSats. This paper aims at describing this effort and promoting the reuse of the C&DH system through its design elements.

Corresponding Author: Shaina A. M. Johl, shaina.johl@gmail.com

2. Background

The objective of developing a C&DH system that can support multiple missions was motivated by several factors, including experiences and lessons learned from both past and current satellite missions at the TSL.

2.1. Past TSL C&DH Architectures

As with most aerospace projects, the development process of this C&DH system did not begin from scratch. The TSL has designed, implemented, and validated C&DH systems for several past missions, some of which have been flown. The TSL has built three satellites that have flown previously: Bevo-1, and two nano-satellites launched together as part of the FASTRAC mission. The knowledge, design work, and most importantly, the lessons learned, inherited from these past missions were extremely helpful in the development of the C&DH module for the current missions. The initial step involved in designing the current system was an analysis of the past C&DH hardware architectures and flight software (FSW) design for these missions. This article presents an overview of the C&DH systems used for Bevo-1 and FASTRAC and the lessons learned that significantly influenced the current system's development.

2.1.1. Bevo-1

The first spacecraft built by the TSL for launch was Bevo-1. The Bevo-1 satellite was built as the first of a series of four missions known as the LONESTAR (Low Earth Orbiting Navigation Experiment for Spacecraft Testing Autonomous Rendezvous and docking) program. The mission objective for Bevo-1 was to collect and downlink two orbits of GPS data to validate NASA JSC's DRAGON (Dual RF Astrodynamic GPS Orbital Navigator) GPS receiver (Johl and Imken, 2013). The current C&DH architecture mimics that of Bevo-1. The design of having one main flight processor that manages all the operations of the satellite is reused, as well as the C&DH software in C++ and in a Linux environment.

The design goal for the Bevo-1 FSW to be reusable between missions was considered a partial success, due to the fact that the benefits of reuse of the previous system were not maximized. This result stems from inefficiency in the practices involved in the development process. Successful reuse is not simply restricted to a source code-level, but applies to other technical levels, such as system architecture, as well as non-technical factors, such as personnel knowledge and documentation (Fortune and Valerdi, 2008). These issues in the organizational structure's development process were identified, however, and new procedures have been established to avoid the practices that act against reuse.

First, there was a lack of available documentation from all steps of the C&DH development process, including design, implementation and testing procedures, test methodologies, and results. This made the task of reusing the software from Bevo-1 difficult by reducing the benefits that should result from reuse, such as shortened development effort and time. Proper and clear documentation is necessary to aid in adopting reuse as a common practice in an organization (Gerard et al., 2007). Therefore, more emphasis has been placed on documenting and logging design decisions, troubleshooting methods, and software bugs.

Second, the FSW on Bevo-1 was programmed to be inherently tailored to the specific hardware of the satellite. The initial goal of implementing the software with the design goal of reusability in mind was abandoned later in the software development due to tight scheduling constraints that required the need to complete the FSW implementation as quick as possible. As the delivery deadline approached, there was less focus on implementing the flight software in a reusable manner, and it became specifically coded to be functional for just the Bevo-1 spacecraft (Imken, 2011).

2.1.2. FASTRAC

FASTRAC (Formation Autonomy Spacecraft with Thrust, Relnav, Attitude, and Crosslink) was a satellite built by the TSL that began in 2003. It was the winning entry of the U.S. Air Force's University

Nanosat-3 Competition in 2005, and was launched in November 2010. The satellite was in operation for more than two years. The FASTRAC project consisted of two nearly identical nano-satellites. FASTRAC had three primary mission objectives, the first of which entailed establishing an autonomous crosslink between the two satellites. The second objective involved performing on-orbit GPS relative navigation, and the final objective demonstrated autonomous thruster firing logic based on an on-orbit single antenna GPS attitude determination solution (Muñoz et al., 2012).

The FASTRAC C&DH system encompassed a distributed software architecture based on a microcontroller design developed by Santa Clara University (SCU). A distributed architecture involves having several processors that divide the computing responsibilities of the satellite and communicate with each other. Each microcontroller manages a separate subsystem of the satellite: communications, power, GPS, and the thruster or inertial measurement unit (IMU), depending on the satellite. This distributed C&DH system architecture adopted from SCU was proven to be effective for the FASTRAC mission. In this case, using a system designed by a third party and customizing it to the specific missions needs reduced the mission development costs. This approach also saved in component costs, as the SCU system is comprised of commercial off-the-shelf (COTS) parts.

However, in practice, the FASTRAC team found that the distributed architecture did not prove to be the most effective choice for re-use on future spacecraft. The main disadvantage of this architecture was in terms of internal data sharing for the satellite. The subsystems required information from each other to perform their respective functions. The software design was complicated by the fact that any data flow from subsystem to subsystem for even the simplest functions had to be sent and received over the common I2C bus, where each AVR-SAT was designated a master. This characteristic introduced timing and sharing complexities, causing non-received data and lockup. Because of this subsystem interdependence, contrary to prior belief, this design did not promote quick, parallel development of the subsystems software.

The selection of this architecture for the current 3U CubeSat missions in the TSL would have presented significant challenges. The subsystems and mission phases for Bevo-2, RACE, and ARMADILLO are more complex than those of FASTRAC, thus the data sharing between modules would have been even more difficult to manage. In addition, the hardware based on the selected C&DH architecture of multiple AVRs required for the separate processors would make it more difficult for the C&DH system to fit within its volume constraints based on the small size of a 3U CubeSat. As a result, the design team did not select a distributed architecture for future missions, although this decision will be re-examined periodically.

2.2. Overview of Current Missions

The TSL is currently developing the spacecraft bus for two 3U CubeSat missions simultaneously, known as Bevo-2 and ARMADILLO. The TSL has also recently developed the RACE satellite, which was onboard the Antares during the launch accident in October 2014. The C&DH system design presented in this paper was used for these missions. A brief overview of each of the missions is presented below, to provide context and motivation for the reusable C&DH design.

2.2.1. RACE

RACE (Radiometer Atmospheric CubeSat Experiment) was a 3U CubeSat developed in collaboration with NASA's Jet Propulsion Laboratory (JPL), which provided the radiometer payload. The system included a 35 nm Indium Phosphide low noise amplifier (LNA) at the front-end. The primary mission objective for RACE was to advance the technology readiness level of the radiometer instrument, thereby reducing the risk of use in future missions (Arastu, 2014).

The TSL's involvement in the RACE mission began in April 2013. The TSL was responsible for building and testing the CubeSat bus, overseeing the payload integration, and managing the ground segment, including data collection. Although RACE was intended for launch to the International Space Station, the Antares rocket that was carrying RACE exploded shortly after liftoff on October 28, 2014, resulting in the loss of all cargo, including RACE.

2.2.2. Bevo-2

Bevo-2 is UT-Austin's satellite that is part of the second mission of the LONESTAR program. The technolo¬gies to be evaluated as the primary objective for Bevo-2 are featured in the spacecraft's Guidance Navigation and Control (GNC) module, including an in-house developed miniaturized star-tracker, an in-house designed cold gas thruster, and an in-house integrated, three-degree-of-freedom Attitude Determination and Control (ADC) module.

2.2.3. ARMADILLO

ARMADILLO (Atmosphere Related Measurements And Detection of submILLimeter Objects) is the winning CubeSat from the 2013 University Nanosatellite Program (UNP) competition sponsored by the U.S. Air Force. The primary objective of this mission is to characterize submillimeter diameter dust particles that are present in low Earth orbit. ARMADILLO features a Piezoelectric Dust Detector (PDD) under construction by Baylor University that will detect particles upon impact with the instrument. The PDD will record the electric charge produced by the impact, storing the measurement data until the C&DH computer queries the instrument. The data is then post-processed and provided to atmospheric models to improve the knowledge of the sub-millimeter space debris environment (Brumbaugh, 2012).

2.3. Spacecraft Modular Design

One of the strategies used by the TSL for developing multiple spacecraft with varied mission objectives is the use of a modular spacecraft structure bus design (Figure 1). The TSL implemented the concept of modularity in several layers of its hardware design. The hardware architecture of the ARMADILLO, RACE, and Bevo-2 structures are very similar; all three are organized with three modules, known as the

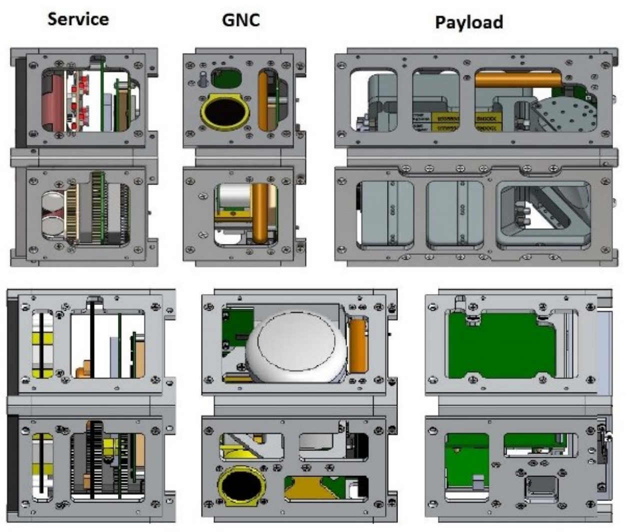

Figure 1. Modular view of RACE (top) and ARMADILLO (bottom) satellites.

Service module, the GNC module, and the Payload module. The modules are functionally similar, although they differ in capabilities, depending on the mission requirements. For Bevo-2 and ARMADILLO, each module is approximately 1U, whereas RACE had 1.5U dedicated to the Payload module.

Each module of the satellite serves a specific purpose. With this design, each module can be developed simultaneously with minimal dependence on the other modules. Two of these, the Service and the GNC modules, are used for each satellite with only minimal necessary changes. It was decided to mimic this concept of modularity in the software architecture as well, so that subsystems as a whole, both hardware and software, could be outright replaced by an entirely different version of the subsystem on another mission. Since all control and data transfer is routed through the C&DH system, a functioning subsystem comprised of the hardware components and the software necessary to operate it accordingly can then be completely replaced without affecting the rest of the satellite.

3. C&DH System Architecture Objective

For current missions, the objective is to develop a C&DH system that can be applicable to university pro-grams involved in concurrent satellite missions.

Therefore, this subsystem not only meets the current demands required by the current spacecraft, but also provides a reference for best practices and design choices based on constraints and qualities that are common for university programs. More importance was placed on developing a system that can act as a base for future missions with the expense of more time and effort invested in the initial system architecture and design. This decision was made based on the analysis of the previous missions, and a comparison with the missions and functionality required by the spacecraft currently in development, as well as for future missions. To achieve this objective, a list of qualities and constraints to guide the design and implementation decisions is presented.

3.1. Reusable Architecture and Portable Software

As the objective of the C&DH development is to construct a system that can be reused for future missions, both the architecture and software design should be generated with this as the primary focus. The three current missions that will be flying the C&DH system are highly diverse in terms of their mission objectives, subsystems, and mission complexity. With each spacecraft having unique payloads, the expected spacecraft mission performance metrics, such as data generation rates and expected telemetry, vary. For the developed C&DH system to act as a reusable system for spacecraft, it must be designed to handle this diverse set of requirements and instrument capabilities. The software should be designed such that it is not hardware specific, allowing for future component replacements. Therefore, the flight software should be portable, to minimize changes required to run for each of the three current missions, and to be reused for future missions.

3.2. High Reliability

The design goals of achieving high reliability and minimizing risk are common to most components, either hardware or software, used in space applications. The main responsibility of the C&DH system is to execute all of the operations that control the spacecraft. The C&DH system has the responsibility of managing all other subsystems to execute the mission successfully. It must interface with the various hardware components of the satellite by sending commands and receiving acknowledgement of the requested actions, as well as recording health and scientific data. The C&DH system is relied on to execute the spacecraft's tasks appropriately and in a timely manner, and to be able to handle run time errors gracefully, with a worst case scenario of resetting the spacecraft to return it to an operational state. Strong emphasis should therefore be placed on selecting the hardware architecture and designing the FSW for high reliability and low risk.

3.3. Simple Design

Working in a university setting, constraints that are present for most spacecraft missions, such as manpower and scheduling, become limiting factors that affect system development. There is a very high turnover in personnel for a university lab, for example; it is not uncommon for the TSL to lose five to ten students every semester. Their exit results in a loss of knowledge and expertise on their aspect of the project, and additional time and effort must be spent on training students to take on their tasks. Personnel flux is a significant constraint on the schedule, since there is a high learning curve before new members can start to make an impactful contribution to the project. Therefore, to keep the impact of training new members as low as possible, the C&DH system architecture and design should consider design simplicity.

Another major constraint for university programs and CubeSat programs is budget. CubeSat programs present the opportunity for lower development, build, and launch costs as compared to larger spacecraft, which is one of the major attributes that make this type of spacecraft favorable. CubeSat missions often demonstrate novel science instruments and technologies on a smaller and more cost effective scale. Therefore, each system of the spacecraft, including the C&DH, should be designed with the goal of low cost, to fit within the tight budgetary constraints.

3.4. Maintainable Software

Maintainable software is also an important attribute to strive for when facing university lab constraints. As there will be many new team members involved with developing the FSW over the years of spacecraft development, keeping the software maintainable and the design simple allows for more rapid comprehension of how the system works. Maintaining the software continuously also promotes software reuse for current and future missions.

4. C&DH Architecture

A centralized architecture was selected for the current missions' C&DH system, based on both the lessons learned from previous missions and the desired qualities and constraints to achieve the desired system properties. A centralized architecture involves all subsystems of the satellite with a point-to-point interface with only the C&DH subsystem. Therefore, all data and commands are sent only between the C&DH system and one other subsystem. This architecture is suitable for satellite systems with a small number of distinct subsystems, and is reliable in the sense that if one system fails during operations, the effect of the failure is minimized, as there is no direct interface with the other systems other than C&DH (Larson and Wertz, 2006). Thus, the integrity of the separate interfaces between the C&DH and the other subsystems remains intact. Furthermore, the employment of this architectural style allows for code reuse that was flown on Bevo-1. The FSW of Bevo-1 was fully functional and tested, and therefore provided a good starting point for the software development for the current missions.

For the flight software, the centralized architecture is employed with a component-based architectural style by treating each subsystem as a component, or module. There is one module per subsystem, and the FSW is then composed of all the modules compiled together into one executable. Figure 2 illustrates the different modules that comprise the FSW for Bevo-2 and ARMADILLO. The modular composition of RACE's FSW is the same, except that it excludes the GPS and NVS components.

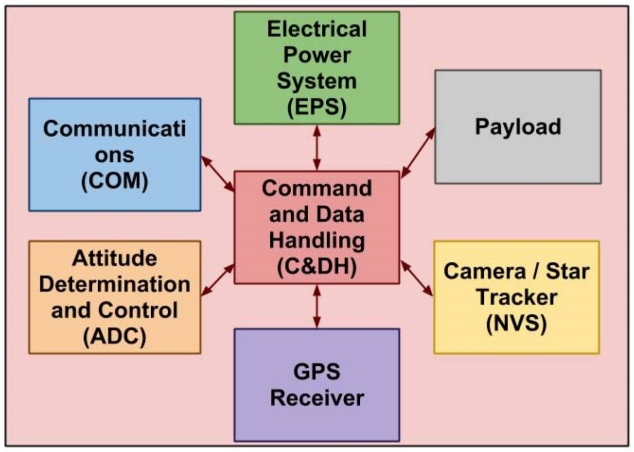
Electrical Power System (EPS)
Communications (COM)
Payload
Command and Data Handling (C&DH)
Attitude Determination and Control (ADC)
Camera / Star Tracker (NVS)
GPS Receiver

Figure 2. Software modules within the FSW.

This figure shows the C&DH subsystem software in the center, interacting with the software module for all other subsystems of the satellite, as indicated by the bi-directional arrows between modules. Both the ADC and GPS subsystems include a processor, but still act as slave components to the C&DH by continuously listening for commands to execute. Interface control documents for each subsystem are created to govern their software interface and functional requirements as needed by the C&DH system. Since each subsystem only interacts with the C&DH system, subsystems can then be developed in parallel with minimal interdependencies. This approach aids in dealing with constraints associated with a university lab, such as reduced manpower. Subsystems can be developed at different rates, allowing re-allocation of resources, such as assigning people to a different subsystem that requires more manpower for completion.

The C&DH subsystem is housed within the Service module, along with the UHF/VHF radio and the Electrical Power System (EPS). Two major components comprise the C&DH subsystem: the flight computer and the interface board. The C&DH system incorporates a System-On-Module (SOM) flight computer, which is the central processor for the entire satellite. A SOM, also known as a computer-on-module, is a sub-type of an embedded computer contained on a single circuit board that can be plugged into a carrier board. SOMs come in different configurations, but generally consist of a processor and standard input/output (I/O) capabilities, which can be configured and connected to other peripheral devices through the carrier board.

4.1. Advantages of Selected Hardware

Starting with a SOM as the processor of the C&DH system instead of designing the flight computer in-house has several advantages, particularly for CubeSat missions. The main advantage is that it simplifies the design of the C&DH hardware, significantly decreasing the effort for this part of the system development. This allows for more time to be spent on developing well-written and well-tested operational FSW. Most SOM computers also run using a Linux distribution. Therefore, the software can be written with portability in mind, and can be reused on other SOMs, independent of the chosen manufacturer.

A university lab has to mitigate constraints on manpower and time. Therefore, taking the approach of using an off-the-shelf embedded computer system for the C&DH computer saves time and effort that would be required for electronic design at the processor level. As SOMs already include the hardware and software necessary to provide computing functionality from the processor, it saves time in not requiring the additional design of complex circuitry for proper computer interfacing (Johl and Imken, 2013), and provides a level of flexibility for multiple applications.

Having the flight computer professionally designed also improves the reliability of the entire C&DH system. It reduces the risk associated with improper design, which can lead to computer malfunctions in orbit and mission failure. As SOMs are mass-produced COTS hardware that are readily available at lower cost, they are a great option for university satellites that have budgetary constraints. Finally, processing and computation power is not compromised, as there are many SOM computers available on the market with sufficient performance characteristics to not restrict the capabilities of the current spacecraft.

4.2. Flight Computer

A trade study was performed to select the computer to be used for the current satellite missions. The principle criteria used in the trade study were storage capability, interface availability, power consumption, documentation, and support. Based on the trade study, the selected SOM that best matched the requirements in place for the flight computer is the Phytec's phyCORE LPC3250. The most important performance characteristics of the LPC3250 SOM are listed in Table 1.

Table 1. LPC3250 SOM Performance Characteristics

CPU Frequency	208 MHz
On-Chip Memory	32 KB L1, 256 KB
CPU Frequency	208 MHz
DRAM	64 MB
NAND	64 MB
NOR	2 MB
EEPROM	32 KB
Available SD/SDIO/MMC	2
UART	7
RS-232	2
RS-232	2
I2C	2
SPI/SSP	4
Power Consumption (nominal)	372 mW
Power Supply	3.15 V

This computer includes NXP's LPC3250 microprocessor consisting of a 266 MHz ARM926EJ-S CPU core, Vector Floating Point (VFP) coprocessor, and a large set of peripherals (NXP, 2011). The microprocessor is designed for low-power, high-performance applications, which is ideal for CubeSat flight computers. The LPC3250 allows for the use of Linux as the running operating system on the SOM, lending to a significant amount of customization in terms of the kernel and pre-existing software tools and libraries.

4.3. Interface Board

The use the LPC3250 for the TSL spacecraft required an interface board to be designed to provide connection to the on-board peripheral devices. The board was designed based on the needs of not only the C&DH system, but of the other subsystems on the CubeSat. The interface board uses a PC104 connector to connect the C&DH system with the electrical power system (EPS) and communication (COM) boards in a stack to comprise the Service module, shown in Figure 3.

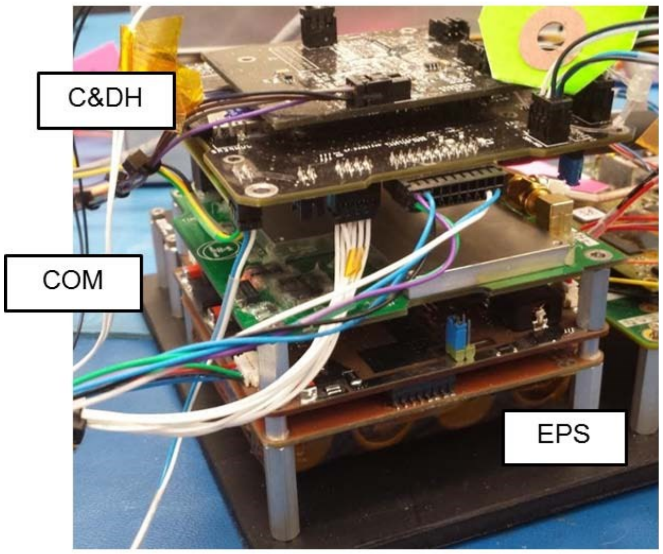

Figure 3. Hardware components of the TSL spacecraft service module.

The interface board houses the main on-board data storage device, a low-power real time clock (RTC), and switches that allow it to power off connected subsystems.

5. C&DH Flight Software Design

With three concurrent projects in the TSL that use the C&DH system, the driving goal behind the development of the FSW is to write code that can be used for separate missions with different payloads, but have a common implementation structure. The focus is to maximize the code reusability between these missions and future missions. This section describes the general software structure and general implementation details that enable code reuse across missions.

5.1. Architectural Approach to FSW

The object-oriented programming style has been used extensively in the architecture of the software. The C&DH software has been written in C++, which supports object-oriented programming. Each subsystem team has written their low-level code in either C or C++, but they are required to provide a high-level C++ wrapper acting as the interface between the two subsystems. Each wrapper includes the commanding functions that the C&DH calls for that subsystem. Therefore, each subsystem is represented as an object that interacts with the C&DH software. Figure 4 illustrates the FSW structure in a layered architectural view to highlight the reusability inherent in its design.

As shown in the figure, there are four software layers that comprise the FSW for the spacecraft. The top layer is the Application Layer, and includes the main FSW application executed on-board the current spacecraft. This application is comprised of the compilation of the processes, functions, and drivers that are contained in the lower layers of the FSW structure. The Software Process Layer provides the processes and the means for storage of mission data, health data, and generated command and error messaging. The High-Level Subsystem Interface Layer is comprised of the components that are used to provide the functionality from the various subsystems and devices of the spacecraft. Each block in the High-Level Subsystem Interface Layer provides the complete functionality for the appropriately named subsystem. The blocks in this layer represent the subsystem modules, as shown in Figure 2, which act as black boxes to the C&DH FSW, except for the defined interface that allows access to the main functionalities required from each subsystem. The lowest level of the figure represents the device drivers that implement the protocols, such as I2C or SPI, and are used by the software components from the upper levels to communicate with the spacecraft hardware.

The application named hookEm is executed on the C&DH computer, and is comprised of all

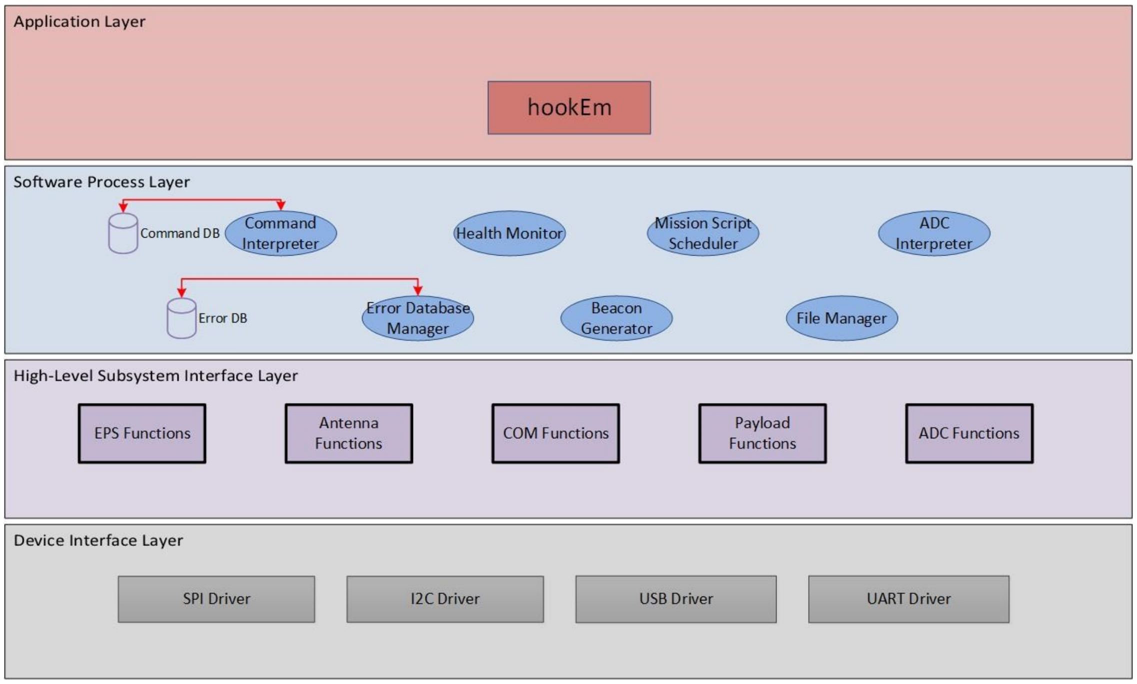

Figure 4. Layered view of FSW structure.

processes and interface functions listed underneath its block in the figure. The motivation behind structuring the FSW using this architectural pattern is to minimize the amount of software modification required for use for the different missions. With each mission having varying payloads and subsystems that comprise the spacecraft, it is inevitable for there to be mission-specific software. However, with this layered approach, the objective is to restrict the required changes in software to only the High-Level Subsystem Interface Layer, and maintain the structure of the C&DH FSW and its components as mission-independent.

5.2. FSW State Machine

All of the spacecraft actions and operations are governed by the FSW implemented by the C&DH system. The FSW is designed as a state machine, with its modes and activities involved in the C&DH FSW depicted in Figure 5.

The state machine is implemented as a class called ModeManager. The ModeManager governs the switching of states between satellite software states, called modes. For each mode, there are certain activities that are allowed to run. There are four modes defined for the FSW that the satellite can transition into: Startup (SU), Automatic Command Execution (ACE), Low Power (LP), and Fail Safe (FS). The switching between modes occurs according to specific flight rules, and is managed by a change in another entity of the software not shown in the figure, the transition variable. The value of the transition variable is altered when certain conditions are met, causing the ModeManager to de-activate the current mode that the satellite software is running, and activate the new mode based on the transition variable's value.

An activity is a process that occurs in the context of one or more modes; an activity may be continuous (LoopedActivity), or may run finitely (Activity). Activities are actions that occur in the flight software, implemented as software threads within modes to perform specific tasks. The activities shown in Figure 5 implement the processes in the Software Process Layer in the C&DH FSW shown in Figure 4. These process functions include major C&DH responsibilities, such as collecting and storing health data, interpreting and executing uplinked commands received

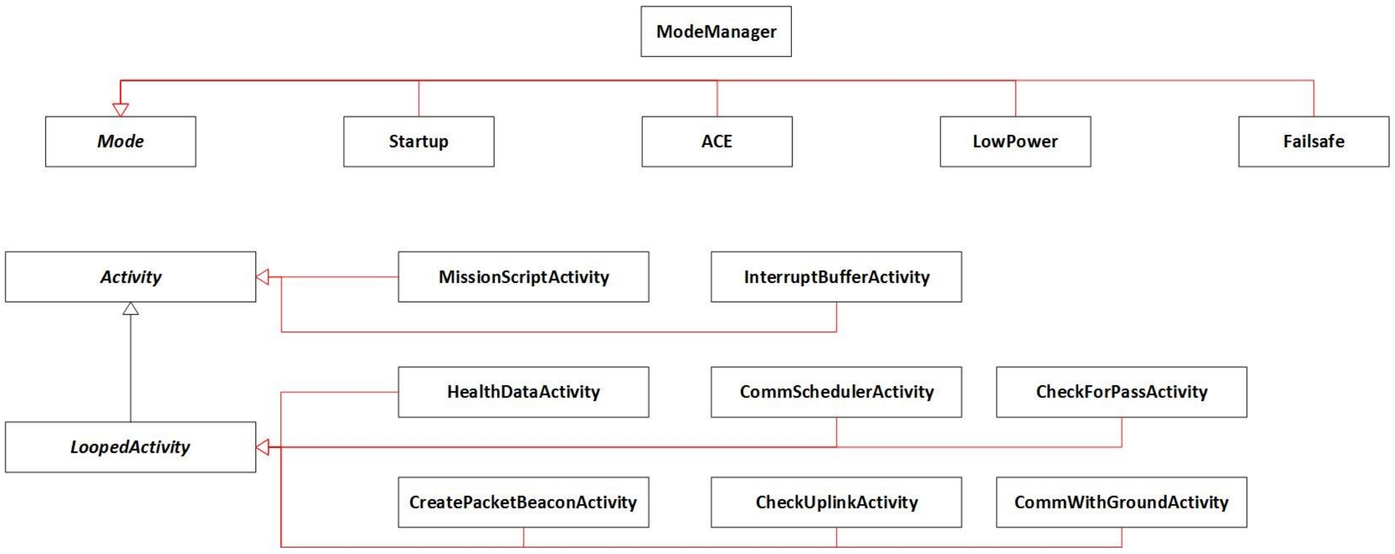

Figure 5. Modes and activities of C&DH FSW.

from ground, and periodically generating and transmitting packet beacons.

This C&DH FSW design is integrated into the FSW of the current missions. All of the missions therefore have the same operational software modes, transition definitions, and the majority of the same activities that are commenced and halted within each mode.

5.3. C&DH FSW Features

In the following text, some of the main functionalities that define the C&DH flight software are described. These functionalities involve the high level responsibilities of the C&DH subsystem, to maintain the operations of the satellite. The implementation of the main functionalities in the FSW that allow for completion of the mission operations of the current satellites includes (but is not limited to): software integrity checking, and error and fault detection.

5.3.1. Software Integrity Checking

One major concern of having the FSW encompassed in one program is possible corruption of the executable file. Single-event effects can be caused by ionizing radiation damage to the flight computer and other electronic components of the satellite due to the space environment, and must be considered when developing the C&DH subsystem. The events include single-event upsets (SEU), single-event latchups (SEL), and single-event burnouts (SEB) (Larson and Wertz, 2006). Redundancy of the FSW has been used to mitigate the effects of program corruption. Two redundant copies of the FSW executable are stored onboard. Upon boot-up of the flight computer, a script will be executed to check the integrity of the primary executable against the two backup copies. The validation of the FSW integrity is performed through a cryptographic hash function to generate a cryptographic token for each of the three executables.

5.3.2. Error and Fault Detection

The satellite is expected to experience errors during on-orbit operations. During software development, the C&DH team is responsible for maintaining the master document that lists all pre-defined errors that can occur in the FSW. These errors will be automatically logged onboard, if they occur during flight. Examples of pre-defined errors include the spacecraft not being able to open a file for reading or writing, or the spacecraft receiving a parameter associated with a command that is out of bounds. Two onboard error databases were created to keep track of any such errors that the satellite experiences throughout the mission. The first database placed onboard contains all the possible error IDs, and any subsequent autonomous actions that the satellite should take. The second error database is a log that is populated as errors occur. This second log can be requested by the ground station to determine if any FSW errors have occurred, so that appropriate ground actions for resolution can be taken.

5.3.3. Omitted C&DH Functionalities

There are also some C&DH functionalities that have been purposefully omitted for the current TSL spacecraft. These omitted functionalities may be common for other larger spacecraft missions, or even other CubeSat missions. The decisions on which features to include as part of the C&DH software functionalities were based on maximizing the level of reliability of the software. This is the direct result of implementing the FSW with the effort to obtain the desired qualities of a simple and robust design. However, omitting certain functionalities present in other C&DH systems forced the team to accept a higher level of risk as a tradeoff for maintaining a simple design. Two of the most common C&DH functionalities featured in other spacecraft that were omitted for the TSL's current missions include autonomous error handling and in-flight software re-programmability.

As discussed previously, the FSW has the capability to identify run-time errors that occur, and store error messages into a database that is maintained on the spacecraft. However, autonomous error handling based on error recognition is extremely limited. Currently, the spacecraft is restricted to executing soft resets for a small subset of the identified errors. The addition of more autonomous capabilities, especially when the spacecraft is already in a state of fault due to the occurrence of an error, would render the software more complex and thereby decrease the overall reliability of the spacecraft to successfully complete its mission objectives. It would also increase testing complexity and time. As a consequence, the responsibility to solve issues attributable to run-time errors is placed on the ground station operators.

Another omitted feature for this version of the FSW is in-flight software re-programmability. This feature was first listed as a subsystem requirement for the ARMADILLO mission, but then was later removed and re-identified as an extended objective. Preliminary work has been performed for implementation of this functionality; however, this effort was abandoned, due to time and manpower constraints. It was decided to focus resources on developing FSW that is reliable and robust, so that this capability would not be required for the current missions, recognizing that it could be selected on future missions.

6. FSW Testing

The FSW has completed the testing phase of the development cycle for validation and verification of the C&DH system. The lab implemented the practice of performing tests on the satellite software and hardware throughout the development process. The tests range from subsystem-specific functional tests to Day-in-the-Life simulations on both the "flat sat" engineering test bed and the fully integrated satellite. Working with the limited resources of university satellite programs, the FSW must be executed in an efficient and thorough testing process. Efforts for testing should never be shortchanged in any flight program; however, the test methodologies and test suites that are executed must be carefully selected in order to maximize software fault identification, and to ensure that testing covers all steps in the development process, from unit level to end-to-end system performance. This section describes the types of tests that were selected to validate the FSW for the current missions.

6.1. Unit Testing

The first type of testing performed on the C&DH software was unit testing. Unit testing involves testing each separate unit of a software program on its own to ensure that it meets its specification. For the C&DH software, each class was tested as a separate unit. Black box and glass box testing are common test techniques employed for testing engineering software products, and these were performed on each class. For the majority of the classes, the correct functionality was confirmed through the performance of these tests. In some cases, when defining the test cases, the C&DH team could recognize an error in the method before running the test.

6.2. Command Execution Testing

The next major step performed in the FSW testing process is the Command Execution Test (CET), the purpose of which is to validate every command that can be uplinked to the spacecraft (Air Force Research Laboratory Space Vehicles Directorate, 2013). This is to help prevent the ground station from sending a command that can place the satellite into an unknown state and jeopardize its integrity. The results generated in this test can then be used to identify and compare with satellite actions that are taken during flight. The test is designed to be executed with the FSW running on the integrated satellite and with the hardware reacting to the uplinked commands. Therefore, the CET not only tests for any bugs or defects in the FSW, but it also tests the various software and electrical interactions between all the components of the integrated satellite.

6.3. Day-in-the-Life (DITL) Testing

Upon completion of the CET, the next tests that are performed on the integrated satellite are scenario tests and Day-in-the-Life (DITL) tests. These tests are designed to simulate in-flight events and activities the satellite will experience. Similar to the CET, these tests only pass data in and out of the FSW by means that will be used during flight operations. The purpose of these tests is to verify the functionality of the fully integrated satellite while it is performing various sequences of flight operations. The DITL tests are the last type of tests to be run on the satellite before it is delivered for launch vehicle integration. Each mission phase for the particular satellite under test must be fully executed in the correct order. As the on-orbit lifetime of the current missions ranges from six months to two years, the DITL tests are an abbreviated simulation of the mission. The test should exercise all of the satellite's FSW modes and transitions. For example, the test should simulate situations where the satellite does not have enough power to accomplish the commands included in the current mission script or command file, or where the satellite experiences a critical failure when attempting to execute a command.

6.4. Recommendations

As the lab is currently involved in the software testing phase for the Bevo-2 and ARMADILLO missions, and completed this phase for RACE, there are some recommendations and improvements identified by the authors concerning the testing process. If applied, these recommendations could prove useful to other small satellite programs as well.

First, the lesson learned from Bevo-1 of the importance of having formalized documentation that describes testing methodologies, procedures, and results was also recognized for the current missions. In addition to thorough documentation, the lab should implement a formalized process for validation of all FSW, such as a formal software acceptance process. One modification to the existing practices is to have different team members act as the software testers than those responsible for implementation of the software under test. This ensures that more than one person understands and tests all software before it is accepted as ready to fly. Formalized testing should occur as early as possible in the development process. This ensures that software defects will be recognized before more components are integrated and it becomes harder to identify the source of an error. Test suites could be generated even before implementation occurs to help ensure that this practice is followed. Alternative test methods that can replace or add to the current executed tests should also be considered, as these new methods may be more efficient in identifying software defects. For example, formalized processes that test multithreaded program attributes such as timing and concurrency should be included.

7. Conclusion

This research presents the development of a reusable C&DH system that is applicable to university small satellite programs working on multiple missions. One area of interest for spacecraft software development is how to design and implement FSW that minimizes mission-specific code. Based on past missions, it was found that more time should be allotted to developing a solid C&DH system architecture that can act as the foundation for future missions. Specific architectural and design choices were made to meet this objective for the current lab missions. Designing the system around a COTS SOM as the flight computer running a Linux environment has mitigated risk that would have been associated with designing the computer from the processor level. Implementing the FSW in C++ using OO techniques allowed for a software architecture using a component-based architectural style, promoting software reuse and portability. Structuring the FSW in a modular manner in which each subsystem is treated as a component that interacts with the central flight computer has also had advantages, such as subsystem upgradability and interchangeability. As with any new system, thorough and well-planned testing is an integral step in the development process, requiring just as much effort as the implementation. Hence, selecting the proper set of test procedures and test suites to maximize FSW validation is integral for providing a reliable C&DH system within program constraints such as manpower and tight scheduling. Proper documentation of the test procedures and results allows for more maintainable and reliable FSW, leading to an overall higher probability of a successful mission.

References

Air Force Research Laboratory Space Vehicles Directorate (2013). NanoSat-8 User's Guide. Technical Report, April. Air Force Research Laboratory, Kirtland AFB, NM.

Arastu, S. H. (2014): Space, Stars, Mars, Earth, Planets and More – NASA Jet Propulsion Laboratory. Available at: http://phaeton.jpl.nasa.gov/external/projects/race.cfm (accessed June 5, 2014).

Brumbaugh, K. M. (2012): The Metrics of Spacecraft Design Reusability and Cost Analysis as Applied to CubeSats. Masters Thesis, University of Texas at Austin, Austin, TX.

Fortune, J. and Valerdi, R. (2008): Considerations for Successful Reuse in Systems Engineering, in *AIAA SPACE 2008 Conference & Exposition*, pp. 1–8.

Gerard, R. et al. (2007): The Software Reuse Working Group: A Case Study in Fostering Reuse, in *2007 IEEE International Conference on Information Reuse and Integration*, pp. 24–29.

Imken, T. K. (2011): Design and Development of a Modular and Reusable CubeSat Bus. Engineering Honors Undergraduate Thesis, University of Texas at Austin, Austin, TX.

Johl, S. A. M. and Imken, T. K. (2013): Design of the Command and Data Handling System in the UT Austin Satellite Design Laboratory. Technical Report, University of Texas at Austin, Austin, TX.

Larson, W. J. and Wertz, J. R. (2006): *Space Mission Analysis and Design*, 3rd ed. Torrance, CA: Mircrocosm Inc.

Munoz, S., Hornbuckle, R. W., and Lightsey, E. G. (2012): FASTRAC Early Fight Results. *Journal of Small Satellites*, Vol. 1, No. 2, pp. 49–61.

NXP (2011): LPC3220/30/40/50 16/32-bit ARM Microcontrollers; Hardware Floating-point Coprocessor, USB On-The-Go, and EMC Memory Interface. Technical Report, October. NXP.