IAC-16-B6.2.6

INTEGRATED FLIGHT & GROUND SOFTWARE FRAMEWORK FOR FAST MISSION TIMELINES

R. Duke
Surrey Space Centre, United Kingdom, r.duke@surrey.ac.uk

C. P. Bridges, B. Stewart, B. Taylor, C. Massimiani, J. Forshaw, G. Aglietti
Surrey Space Centre, United Kingdom
{c.p.bridges, b.stewart, b.taylor, c.massimiani, j.forshaw, g.aglietti}@surrey.ac.uk

Flight and ground segment software in university missions is often developed only after hardware has matured sufficiently towards flight configuration and also as bespoke codebases to address key subsystems in power, communications, attitude, and payload control with little commonality. This bespoke software process is often hardware specific, highly sequential, and costly in staff/monitory resources and, ultimately, development time. Within Surrey Space Centre (SSC), there are a number of satellite missions under development with similar delivery timelines that have overlapping requirements for the common tasks and additional payload handling. To address the needs of multiple missions with limited staff resources in a given delivery schedule, computing commonality for both flight and ground segment software is exploited by implementing a common set of flight tasks (or modules) which can be automatically generated into ground segment databases to deliver advanced debugging support during system end-to-end test (SEET) and operations.

This paper focuses on the development, implementation, and testing of SSC's common software framework on the Stellenbosch ADCS stack and OBC emulators for numerous missions including Alsat-1N, RemoveDebris, SME-SAT, and InflateSail. The framework uses a combination of open-source embedded and enterprise tools such as the FreeRTOS operating system coupled with rapid development templates used to auto-generate C and Python scripts offline from 'message databases'. In the flight software, a 'core' packet router thread forwards messages between threads for inter process communication (IPC). On the ground, this is complemented with an auto-generated PostgreSQL database and web interface to test, log, and display results in the SSC satellite operations centre. Profiling is performed using FreeRTOS primitives to manage module behaviour, context, time and memory – especially important during integration. This new framework has allowed for flight and ground software to be developed in parallel across SSC's current and future missions more efficiently, with fewer propagated errors, and increased consistency between the flight software, ground station and project documentation.

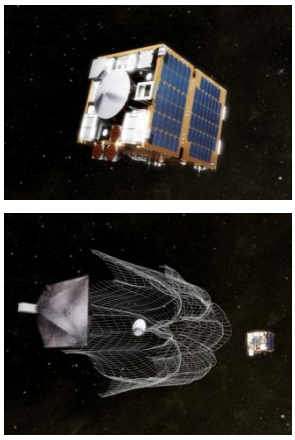
I. BACKGROUND

Nanosatellites in the form of the CubeSat have proliferated in educational institutions towards real-world, team-based and often time-restricted engineering projects that provide an excellent addition to students' careers. New industries and companies have been providing subsystems and services through to full missions for over a decade. Despite this, CubeSat missions have a statistical low success rate. The findings by Erlank et al[1] shows that there is no clear recipe for success. A key area investigated was the variation in collective team experience in designing, building, launching and operating of spacecraft. This inexperience often falls foul of the basics in requirement forming, following testing processes, and even in the general philosophy for justifying design decisions.

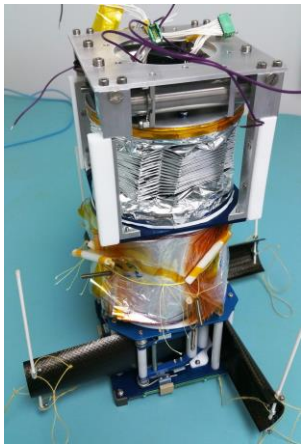
A number of specifications have developed over the last decade, including the CubeSat Design Specification which focuses on mechanical conformity but not regarding flight or ground software. In the U.K., agency and community consultations have taken place regarding the use of standardisation to aid in assessment when applying for space licenses. There was a positive yet mixed response when discussing how international standards could or should be employed to further provide safety assurances about the intrinsic hazards of CubeSat missions to other existing missions. Although SSC is involved in the definition of ISO standards specifically designed for 'lean' satellites, i.e. satellites that utilise non-traditional, risk-taking development and management approaches with the aim to provide value of some kind to the customer at low-cost and without taking much time to realise the satellite mission[2], involving small teams and restrictive budgets, those

[1] A. Erlank, C. P. Bridges, 'A Bio-Inspired Approach to Increased Reliability on SMESAT', 8th European CubeSat Symposium, Imperial College, London

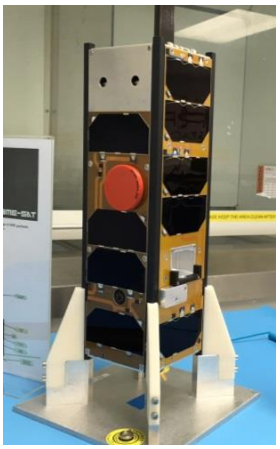
[2] ISO/TC20/SC14, or, specifically CubeSat (ISO/DIS/17770), Testing (ISO/DIS/19683) and Top requirements (ISO/CD/20991)



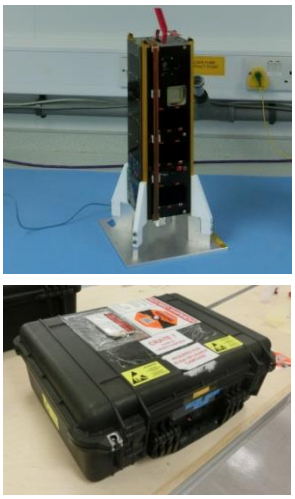
RemoveDebris DSATs
(Early AIT)



QB50 InflateSail
(Late AIT)



SME-SAT
(Awaiting EVT)



AlSat-1N
(Shipped for launch)

Figure 1. Surrey Space Centre Satellite Progress

standards are not mature yet. Therefore best practice will apply as suggested from the U. K. CubeSat and Nanosatellite Committee (formed of members from industry, academia, amateurs, and entrepreneurs) which found that[3]:

- U. K. developers should be free to build the CubeSat in any way possible so long as it can be reliably and safely ejected from a POD deployer.
- Reference to standards or guidelines applied in full or part during a CubeSat development can help provide extra assurance that processes undertaken by a developer are robust, and may help agency understanding of commonly used standards or guidelines and in assessing their worth.
- Existing ISO Standards, as well as ECSS and CCSDS specifications are designed for bigger teams; therefore they should be employed to specify standard aspects of CubeSats platforms that can provide safety assurances regarding the intrinsic hazards presented by the CubeSat satellite and its subsystems to other space systems.

Incorporating greater process leads to a more robust and reliable spacecraft, however, this also comes at greater financial and scientific risk to benefit business and academia so that developers will not be disadvantaged to international CubeSat developers.

The majority of education and research CubeSat missions tread this fine line in being financially viable and technologically acceptable; with relevant ECSS and CCSDS specifications and books often ignored; or best practice otherwise omitted on grounds of project resources. There are however some salient points to take from these books, particularly ECSS-E40[4] which details the entire software development lifecycle, from requirements capture right through to verification and validation testing. Bartram et al[5] discusses his work on the ESA ESEO mission, and highlights three areas needing to be addressed:

1. A low-cost satellite development process does not have the resources required as a typical ECSS project and therefore cannot rigorously follow all aspects of the recommendations.
2. The ECSS framework is designed for large teams of engineers all contributing to a single development, this has wide reaching implications for source code management.
3. The lack of peer review process and the implications of leaving all of the testing until the end of the development adds risk to the project and methodologies should be investigated to provide snapshot review alongside code generation.

The ultimate intention for low-cost software should be the creation of a new lightweight process incorporating the key features of specifications for rapid development whilst ensuring that software quality standards are throughout.

[3] U. K. CubeSat and Nanosatellite Forum, 'White Paper on U. K. CubeSat Regulation & UKSA CubeSat Consultation', September 2015, http://www.cubesatforum.org.uk

[4] European Cooperation for Space Standardization, "ECSS-E-ST_40C," 2009

[5] P. Bartram, C. P. Bridges, 'ESEO Satellite Communications Payload', MSc Dissertation, University of Surrey

I.I Multi-Mission Workplace

Like many institutions and companies, multiple simultaneous projects and satellite builds are common place – and at Surrey this posed a particular set of problems. As a university, the student and staff turnover on flight projects is high creating difficulties with project continuity. In this paper, we highlight four projects that are in varying points of development as shown in Fig. 1.

RemoveDebris

RemoveDebris is a low cost mission performing key active debris removal (ADR) technology demonstrations including the use of a net, a harpoon, vision-based navigation and a dragsail in a realistic space operational environment, due for launch in 2017[6,7]. For the purposes of the mission CubeSat debris-sats (DSATs) are ejected from the primary platform then used as targets instead of real space debris, which is an important step towards a fully operational ADR mission. The EC FP7 RemoveDebris mission, undertaken by a consortium of 10 partners, aims to be one of the world's first in-orbit demonstrations of key technologies for active debris removal and is a vital prerequisite to achieving the ultimate goal of a cleaner Earth orbital environment.

InflateSail

InflateSail[8] is a 3U CubeSat intended to demonstrate the effectiveness of drag deorbiting from low Earth orbit. The mission payload consists of a 1 m long 90 mm diameter inflatable mast and a 10 m^2 sail and the spacecraft is divided approximately into 1U for the avionics and 2U for the payload. The sail, the sail support booms and the deployer mechanism are based closely on previous sail designs at SSC, while the inflatable boom is completely new. The cool gas generators (to drive the inflation) are based on existing technology, but this particular version was developed especially for the InflateSail mission. The majority of the electronic components are COTS, with the exception of the motor controller which was developed in-house. The 'Jack-in-the-box' deployment of the inflatable and sail deployer means that the side panels are not required to open. Only the top 'lid' of the structure needs to be deployable. Four polymer clips hold the lid firmly to the body of the satellite and slot into grooves in the walls of the satellite.

SME-SAT

This spacecraft is the culmination of 3 years work under the FP7 SME-SAT project where the consortium has been able to "design, build, integrate and test platform and payload subsystem hardware and software into a new technology demonstrator mission based on the 3U CubeSat standard". It contains either highly advanced or science-grade attitude sensors such as the Sensonor inertial measurement unit, LEMI magnetometer, ISIS star-camera and Theon/ESS accelerometers. In addition to this, SSC built a new control moment gyro (CMG) system for actuation and SystematIC built a new EPS. Although the project is now closed, it is currently in soft-stack configuration and awaiting the environmental test campaign.

AlSat-1N

The AlSat-1N[9] spacecraft is being designed, built and launched as part of the AlSat-Nano programme for Algerian students as an international collaboration between the UK and Algeria, supported by the UK Space Agency. SSC were responsible for the platform hardware design, development and construction during AIT, and final system testing. The Algerian Space Agency (ASAL) is responsible for launch and operations with expert guidance from UK. Approximately half of the AlSat-1N platform was been made available to host self-funded payloads from the U.K. CubeSat community as a free flight opportunity.

With all these missions having similar timeframes to completion, a new structure was put in place at the beginning of 2016 to pool the staff/student resources into our SSC Engineering Team. Those involved in software were two primary developers to write the code, numerous internal and external users providing feedback and requirements as satellites progressed, and an academic member of staff for steering and mentoring. These limited resources are not uncommon in many nanosatellites missions and are a significant constraint.

However, for all the SSC CubeSat missions, there are a number of common hardware elements: primarily the SSC/ESL Stellenbosch Attitude Determination and Control System (ADCS) stack for the QB50 project.

[6] J. L. Forshaw, G. S. Aglietti, N. Navarathinam, H. Kadhem, T. Salmon, A. Pisseloup, E. Joffre, T. Chabot, I. Retat, R. Axthelm, S. Barraclough, A. Ratcliffe, C. Bernal, F. Chaumette, A. Pollini, W. H. Steyn, RemoveDEBRIS: An in-orbit active debris removal demonstration mission, Acta Astronautica 127 (2016) 448 – 463. doi:10.1016/j.actaastro.2016.06.018

[7] Forshaw, J. L., Aglietti, G. S., Salmon, T., Retat, I., Roe, M., Burgess, C., Chabot, T., Pisseloup, A., Phipps, A., Bernal, C., Chaumette, F., Pollini, A., and Steyn, W. H. Review of final payload test results for the RemoveDEBRIS active debris removal mission. In 67th International Astronautical Congress, Guadalajara, Mexico, 2016

[8] Viquerat, A., Schenk, M., Lappas, V. and Sanders, B., 2015. Functional and Qualification Testing of the InflateSail Technology Demonstrator. In 2nd AIAA Spacecraft Structures Conference (p. 1627)

[9] C. P. Bridges for SSC Engineering Team, 'Surrey Space Centre Mission Update' 2016 AMSAT-UK Colloquium, https://www.youtube.com/watch?v=8gjx3d8Po8U

The QB50 stack[10] consists of 3 boards, the CubeComputer, CubeControl and CubeSense boards. Of key interest is the CubeComputer which performs the CubeSat processing and contains a 32-bit ARM Cortex-M3 including flash for in-flight reprogramming (dual redundant), an FPGA for flow-through error correction in case of a radiation upset on the memory and a MicroSD card for data storage. The CubeControl controls magnetorquers and samples connected sensors. The CubeSense contains both Sun and nadir sensors.

This common platform can be exploited towards writing homogenous flight software and ground software making use of open-source professional web tools for logging and management. As such, the rest of this paper is devoted to the philosophy and implementation of the Surrey Common Software Framework and its use in SSC flight projects.

II. SURREY COMMON SOFTWARE FRAMEWORK

At the beginning of this software effort, a number of C-based libraries were available. These were available from sources including the Energy Micro Processor Board Support Package (BSP), the Stellenbosch / ESL BSP, and also the Surrey QB50 Flight software. In addition to this, there were DeorbitSail and STRaND1 code bases, along with open-source operating systems. The vision for the flight software was to be:

- Modular: 'Plug and play' with common core applications and optional modules for hardware and mission specific functions. Improvements made to the software for one project could be easily shared amongst the others.
- Mission Independent: Aligning software and reducing differences between missions that results in better sharing of developer resources and operator training making the software flexible enough to handle multiple missions also results in more flexibility when in orbit.
- Rapid Development: Auto code generation (AGC) is used to build all telemetry and telecommands handlers and provide the developers with data structures to rapidly code.
- Maintainable: Using programming languages common to the university environment, standard code and code structures reduced the time for new developers to get up to speed in an environment where short term research contracts are in use.

To be able to create this flexible architecture, allowing for multiple simultaneous developments, FreeRTOS[11] was chosen and a hardware abstraction layer (HAL) was developed. The FreeRTOS operating system was chosen as it has a wide user base, flown on previous missions, and would be simple to port to the Stellenbosch OBC computer and memory architecture. The HAL ensures that any I2C, CAN, FLASH, and UART interfaces employ appropriate mutex handling, context switching, and priority inheritance to prevent multiple access and to not block high priority tasks as shown in Fig. 2.

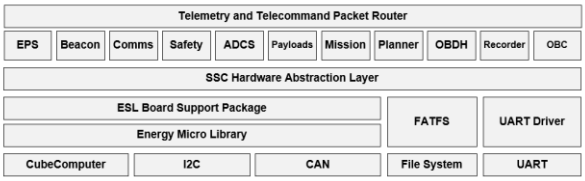


Figure 2. OBC layer Diagram

The general philosophy of the software design reduced down into three options:

1. Fully Custom System where all code is custom, can interface with everything but is extremely labour intensive.
2. Convert to a Standard Router model where all data packets must adhere to strict formats (i.e. OBDH requirements pushed onto other subsystems).
3. Convert to a Flexible Router model where the OBC simply forwards on packets but must decode them at runtime to ensure compatibility with all subsystems.

Each task has a queue to accept incoming packets as well as separate timing and memory allocation. Tasks will suspend automatically when awaiting incoming messages to minimise CPU consumption.

II.I Packet Routing and Formats

The core function of the flight software is to relay messages between the ground and various subsystems on the spacecraft. Therefore the key to making a system capable of spanning multiple missions is to create a flexible messaging format. This is especially important in the field of CubeSats. Often COTS payloads and interfaces are of different standards, and very little customisation from providers is possible to match internal standards and interfaces.

As a result of the flexible packet format, the design for the OBC was chosen to employ an internal router system between payloads and subsystems modules. Extending the FreeRTOS Queue Management

[10] ESL, Stellenbosch University , Cube ADCS Module, http://www.cubesatshop.com/product/cube-adcs/

[11] FreeRTOS 'Quality RTOS & Embedded Software', http://www.freertos.org/

functions[12], the packet router forwards all packets placed onto the 'TCT_Queue' to the relevant subsystem where is then decoded for use as shown in Fig. 3.

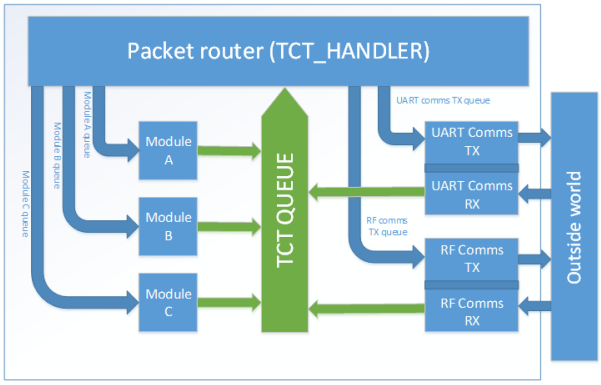


Figure 3. Packet Router Architecture

II.I Autocode Generation (AGC)

A major challenge facing CubeSats, especially across multiple missions, is synchronising of telemetry and telecommand formats and information. Information across subsystems and payload providers, Interface Control Documents (ICD), operator documentation, and especially source code and mission control systems must all be kept in sync. This is solved by using auto code generation (ACG) for all telemetry and telecommand functionality. Using AGC also helps reduces coding errors as all inputs can be checked as the code is generated and common hand-written errors flagged to the developer. As the majority of code is focused around the AGC-based telemetry and telecommands, this vastly reduces the numbers of issues that occur.

The prototype ACG was and developed for the QB50 Surrey / Stellenbosch ADCS system and also used for the DeorbitSail[13] spacecraft. This generated basic telecommands handling. The most recent version developed at the SSC for the common framework expands this by allowing automatic code generation of I2C and CAN bus forwarding, as well as telemetry/telecommand variable storage support resulting in over 50% of the spacecraft code being auto generated.

To begin this process, an Excel sheet is filled out containing the format and information for each telemetry of telecommand messages. Keeping this in a standard Excel format means that payload and subsystem providers can fill these in directly. This document then becomes the Software ICD between the provider and the software team.

The Excel sheets are read via a custom Python script. The script checks for errors and alerts the user if any are found. This eliminates many typical cases of user error. Next C code is generated to handle the telemetry commands and the mission control PostgreSQL database[14] is updated so that the new changes can be tested instantaneously with the groundstation. All telemetry and telecommands variables are automatically created and stored in internal structures so that developers writing code can be rapidly generated by entering parameters in the correct procedures and have access to all telemetry values and variables directly

Finally the script generates documentation in a user-level Word format that can be used by operators and incorporated into a design document. The result of this means that documents and code are always kept live and up to date with relatively little overhead. This process principle is shown in Fig. 4.

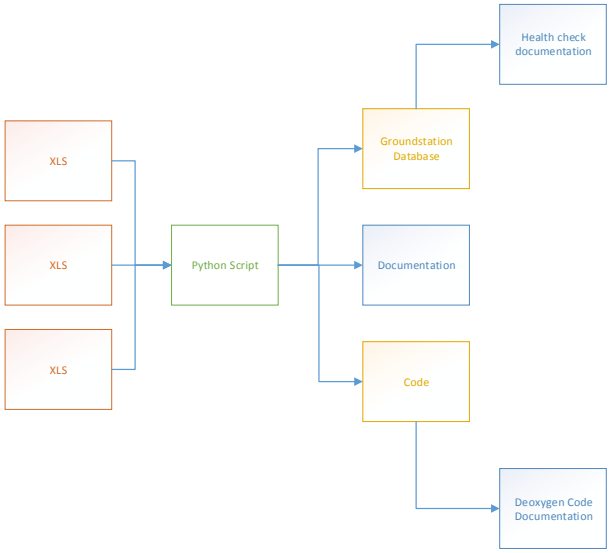


Figure 4. Autocode Generation Flow

III. MISSION CONTROL SYSTEM (MCS)

Many mission control systems are based around a single program (single executable). They are typically bespoke, used for limited amount of operations/missions and often just a basic interface for operators. At the SSC, we leverage modern web based interfaces and a highly reliable database to allow for multiple simultaneous, and multiple connections from a variety of systems.

[12] FreeRTOS 'Queue Management', http://www.freertos.org/a00018.html

[13] Olive R. Stohlman and Vaios Lappas. "Development of the Deorbitsail flight model", Spacecraft Structures Conference, AIAA SciTech, (AIAA 2014-1509), http://dx.doi.org/10.2514/6.2014-1509

[14] PostgreSQL Database, Open-Source, https://www.postgresql.org/

All data and commanding is centralised in a database for spacecraft both during testing and whilst in orbit. The MCS can support multiple groundstations as multiple incoming data sources. Currently the MCS supports multiple on orbit operations, monitors and interfaces with all spacecraft under test at the SSC and is also used as a developer and training tool via use of emulated groundstation and spacecraft hardware. Using the same interface and system for development and training through to on orbit operations reduces the testing needed when separate systems are needed and also reduces engineer training.

A benefit of basing the system around a multi-user database is to allow the separation of 'user interfaces' and 'mission critical interfaces' which allows for isolated coding and test.

Using a COTS database with standard SQL interface reduces the need for custom interface documentation. In addition, using COTS software as well as programming languages such as Python that are well used in universities allows the code to be maintainable with existing skill sets. Modularity of the system also helps with maintainability and upgrades, as well as tracing faults more easily.

A key aim of the MCS was to expand the user base from operators to all engineers involved in designing and building the satellites across the different disciplines. Specifically AIT engineers and systems designers are supported by the MCS by providing automated scripting and report to reduce testing time and effort.

In addition, the SSC groundstation software is based on similar concepts that allows operators to use a single web interface to command and view both groundstation and mission control consoles in a unique environment – see Fig. 5 and 6. However similar, both are isolated with a distinct API to either to be developed or used on its own, or with other systems.

III.I MCS Scripts

Two scripts are used for each connection to the MCS from outside sources such as ground stations, via UART's, email, etc. All incoming data to the MCS are currently connected via UDP ports with basic API consisting of Spacecraft ID, Unix Time, Data, and additional parameters.

The incoming script is considered the most vital and therefore is kept as simple as possible. It receives incoming data and stores it directly in the database. It also forwards on the UDP packet onto the outgoing script so that the outgoing commander script is able to respond to incoming telemetry in real time. The outgoing script can call different custom scripts depending on the action needed to be performed – such as change of transmitted telecommands. In most cases, this is as basic as checking the database for any commands flagged as ready to transmit, forwarding the hex code via the UDP and awaiting the response, and finally acting accordingly on any result.

To reduce system downtime, all scripts are monitored constantly by ensuring each script regularly updates the database with a latest 'heartbeat' time. If a script crashes for any reason, the heartbeat time is not updated and the operators can be automatically alerted.

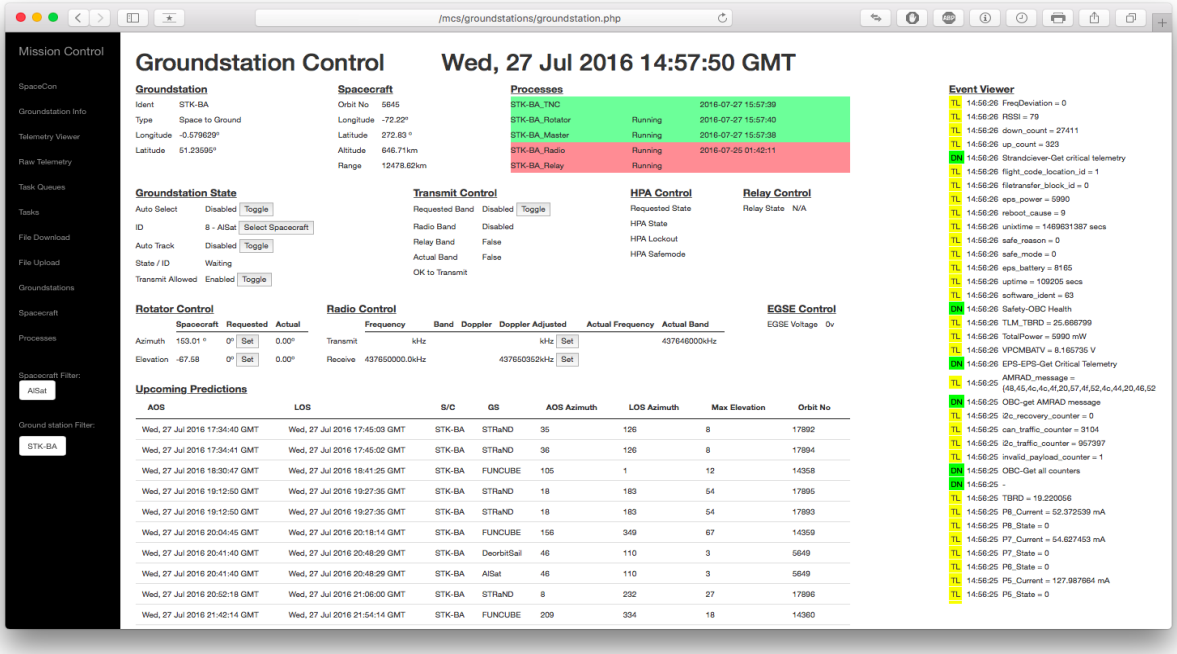


Figure 5. MCS Groundstation User Web Interface

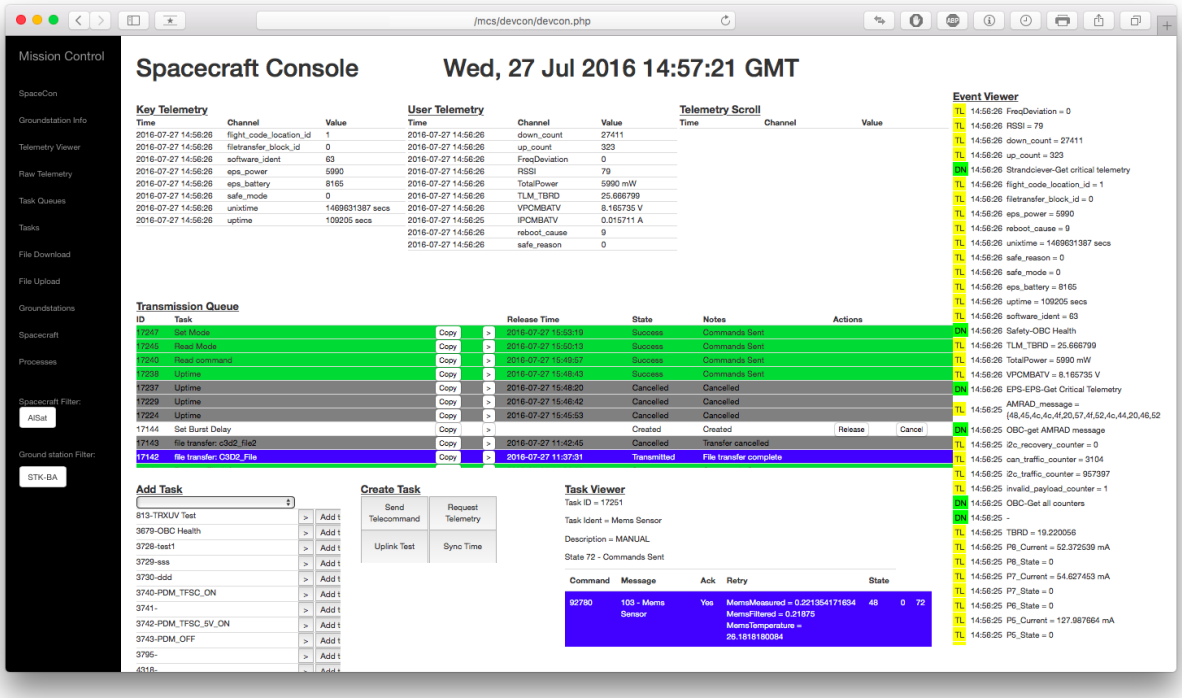


Figure 6. MCS Spacecraft Control User Web Interface

III.II Tasks & Scripting

New commands can be added or removed from the command stack using the command generator which constructs the packet via a graphical interface. The basis for commanding the spacecraft is setting up tasks generally in the form of command stacks, or scripts. The task page allows these to be set up in advance of a pass, minimising to minimise any human errors during a short pass window. Command stacks may be cancelled by operators at any time.

When a task is run, each message is sent in turn. The command is logged as failed if the command did not acknowledge, or if validation criteria is not met. If a command fails validation, then task is state Successful completion (with errors) and flagged. Tasks are designed for multiple runs during development, functional testing, end to end testing, and on-orbit commissioning. For testing of our SSC flight missions, scripts, formed of multiple telecommand and telemetry requests were used many times for common check-out procedures, proving very useful during EVT. The common user interfaces are shown for configuring the groundstation console scripts for tracking and Doppler control, as well as the spacecraft 'data path' console in Fig. 5 and 6.

IV. NANOSATELLITE TEST CASE

The recently completed AlSat-1N satellite has been logging data since March 2016 and has been using the MCS for incoming platform and payload hardware tests, verification of functional interfaces, and exercising of key telemetry points as the spacecraft is integrated. As intended, the MCS was used extensively with the express purpose of being able to quickly translate, view, and log for long-term future storage and analysis. The key library used for this functionality was HighCharts[15] allowing for sliding windows views of varying time periods. The following screenshots highlight some of the key images allowing end-users to write scripts, execute procedures and ensure spacecraft safety. Figures 7 and 8 shows the result from data collection of the batteries and electrical system to quickly view trends in temperature changes during thermal vacuum testing.

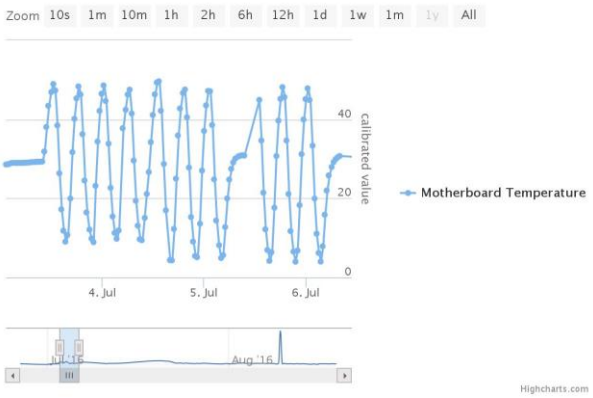


Figure 7. EPS Motherboard Temperature Log

[15] Highcharts, http://www.highcharts.com/

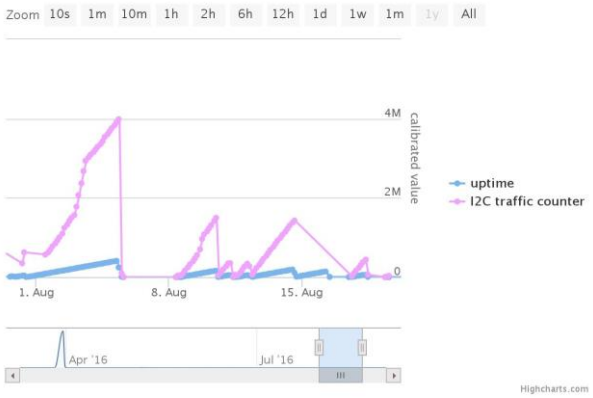


Figure 8. OBC Uptime & I2C Traffic Counter Logs

Further to these functions, the database can be interrogated directly. Using these modern tools, standard SQL statements can be used to provide development statistics during a mission. Of particular interest is the transition between systems testing of the satellite over the UART to when the RF groundstation in the loop testing began (shown in Fig. 9).

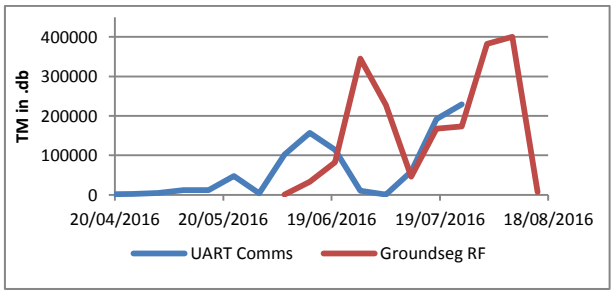


Figure 9. Spacecraft TM Packets (UART to RF Testing)

Further to this, the team can investigate how each subsystem (or thread) is added into the database. Raw statistics are discussed rather than the data content itself. We can see from Figures 10 and 11 when subsystems are delivered, their hardware checked out, and then developed further in software. We can see that Payload 2 was developed during the April-May '16 period and Payload 1 in June '16. A clear ramp up is shown in threads over time; beginning with the EPS, through to ADCS, File Transfer, RF, and finally overall OBDH threads. This analysis can highlight where current or future effort is needed in future missions even in regarding simple items such as naming conventions, to the development order, to the impact of addition of late subsystems in the satellite design. In this case, it is clear where the critical platform subsystems underwent extended test before delivery and separated from payload development.

V. SUMMARY

When comparing commercially available flight and ground software systems with our own implementations, we found that many key features introduced have accelerated test procedures and provided a transparent way for developers and users to engage with software as it develops. These include the combined use of AGC for flight and groundstation software, FreeRTOS adoption, and the internal packet router. This new architecture has been built and validated through test using a small team with close feedback from users – in under 6 months.

New user interfaces have been built using the latest web practices with individual scripts to control the complex needs of multi-mission systems – for developer and user requirements. SSC will continue this development for our current missions at Surrey as we progress into operational flight missions.

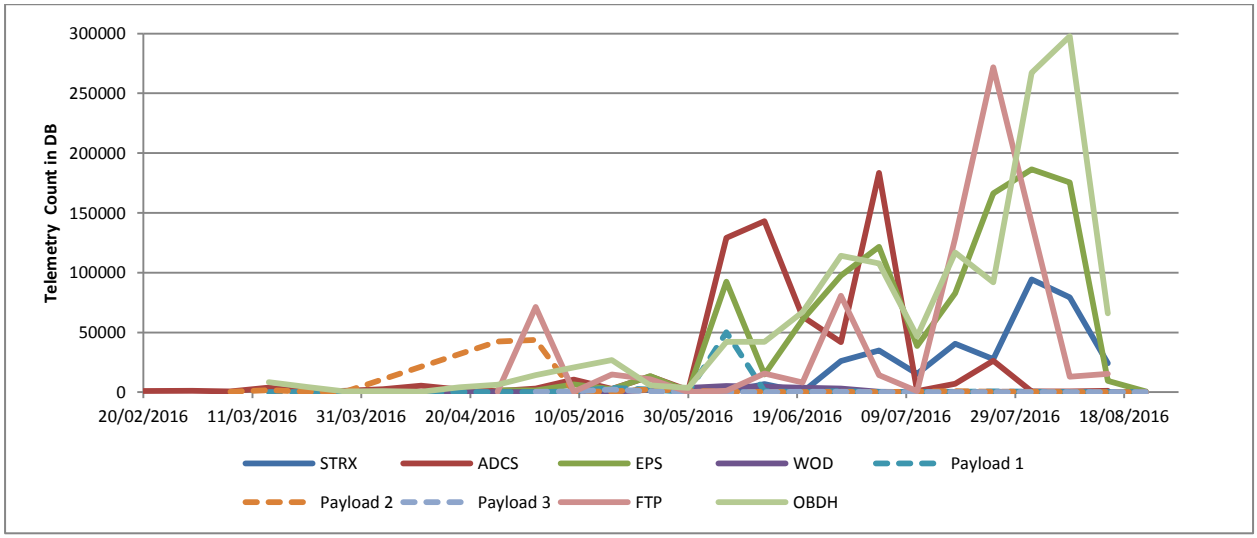


Figure 10. PostgreSQL Database TM Entries over Time per Thread (i.e. subsystem)

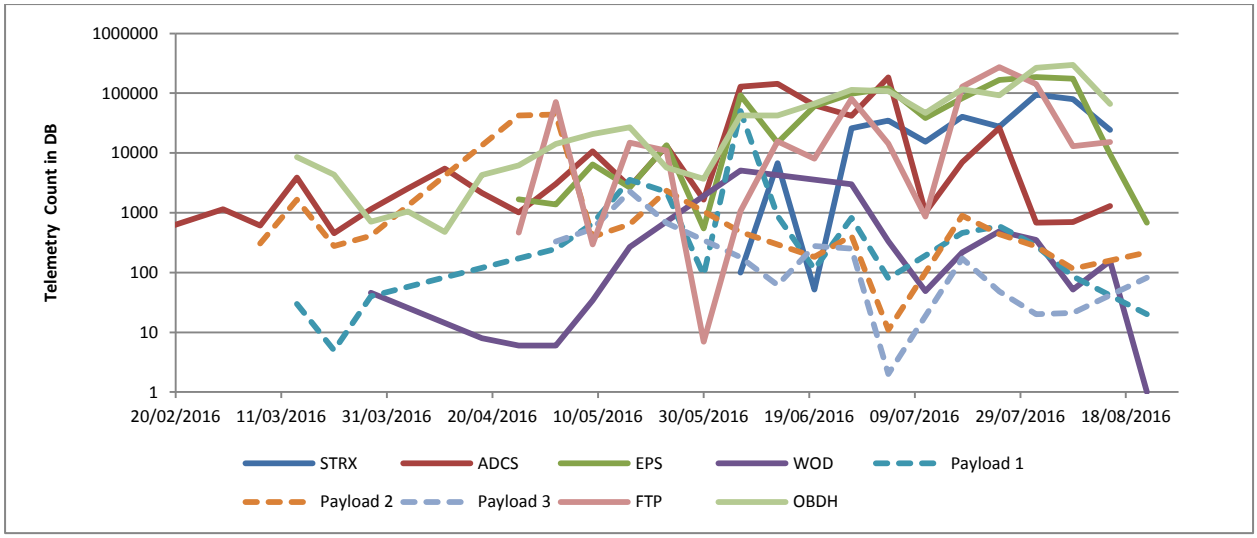


Figure 11. Log plot of PostgreSQL Database TM Entries over Time per Thread (i.e. subsystem)

ACKNOWLEDGEMENTS

The authors would like to acknowledge the following contributors to this paper: Lourens Visagie for the prototype AGC work during his time at SSC, numerous flight and ground station code developers including Brian Yeomans and Oliver Launchbury-Clark, our project consortium partners and sponsors on SSC's missions, and insights from our industrial partners.