Fault-Tolerant and Deterministic Flight-Software System For a High Performance CubeSat

Ilia Benson, California State University Northridge, Los Angeles, CA, USA
Adam Kaplan, California State University Northridge, Los Angeles, CA, USA
James Flynn, California State University Northridge, Los Angeles, CA, USA
Sharlene Katz, California State University Northridge, Los Angeles, CA, USA

ABSTRACT
We describe the design and implementation of a complete flight-software operating system (OS) for a high-performance CubeSat carrying a third-party payload. This CubeSat’s mission is to carry out research experiments on this payload while in low earth orbit. These experiments may be specified and prepared on the ground while the CubeSat is already in flight, and later uploaded via communication link. Experimental results collected by the CubeSat may likewise be downloaded to the ground. The CubeSat must collect and respond to sensor telemetry every second, and respond to ground communication on demand. To survive an adversarial space environment, the CubeSat OS is implemented as a deterministic state machine, storing state in a fault tolerant global memory structure. We validate our system via an end to end test of the CubeSat with its ground station, and demonstrate its capability to tolerate and even actively mitigate potential faults resulting from space radiation.

KEYWORDS
CubeSat, Determinism, Fault Tolerance, Operating System, State Machine, Triple Modular Redundancy

INTRODUCTION
This paper documents the design and implementation of a complete flight software system for a CubeSat (Heidt et al., 2000) housing a 16-bit Microchip dsPIC-33F microcontroller with a mere 32 kB of on-chip static RAM. This CubeSat, CSUNSat1, is the product of a four-year collaboration between CSUN and the Jet Propulsion Laboratory (JPL), and was developed by a team of over 70 students (Benson et al., 2016; Bolotin et al., 2015; Chin et al., 2014-15; Kaplan, Flynn and Katz, 2014). This CubeSat’s mission is to carry out one or more research experiments while in low earth orbit. These experiments may be specified and prepared on the ground as a scripted sequence of steps while the CubeSat is already in flight, and later uploaded from ground to the CubeSat via communication link. Experimental results collected by the CubeSat (hereafter referred to as telemetry readings) may likewise be downloaded from the CubeSat to the ground.

CSUNSat1 (shown in Figure 1) must function under extraordinary design constraints, including the aforementioned RAM capacity. Moreover, equipment budget and physical space limitations prevent us from employing radiation-hardened memory, forcing us to consider the effects of space radiation on our system state and data integrity. Finally, the satellite must be able to sample 44 independent hardware sensors every second and record this data for later transmission to ground on-demand. Such
a performance demand is met by a fully custom-written flight software OS, designed specifically for this mission with the goal of providing maximum availability in the presence of space effects. CSUNSat1 has no attitude control, and is therefore tumbling in space. Additionally, CSUNSat1 will be launched into low earth orbit (LEO). Thus, communication between satellite and ground is opportunistic and limited to small windows of time, forcing CSUNSat1 to perform independent self-maintenance and reliably store multiple weeks of experimental data.

The CSUNSat1 flight software is designed to maximize predictable satellite behavior in an adversarial space environment. This software is not programmed using a conventional off-the-shelf embedded real-time operating system (RTOS) such as VxWorks (Wind River Systems, 2016), Salvo (Kalman, 2004), FreeRTOS (Barry, 2010), or RODOS (Barschke, Großekatthöfer and Montenegro, 2014). Rather, we have designed and implemented the operating system (OS) ourselves, specified as a deterministic state machine (Schneider, 1990).

Important system state is maintained in a global data structure. This global structure is stored in triple-modular redundant (TMR) fashion for fault tolerance, allowing the satellite to tolerate a large degree of memory corruption and/or failure. Periodic checks on the integrity of attached storage, as well as an extensive Anomaly Resolution Subsystem, further enable us to tolerate and even actively mitigate potential faults resulting from space radiation.

BACKGROUND AND RELATED WORK

A CubeSat (Heidt et al., 2000) is a miniature satellite (20 x 10 x 10 cm) capable of carrying an onboard experiment into space. The NASA CubeSat Launch Initiative (CSLI) (National Aeronautics and Space Administration, 2016) provides a launch free of charge for some educational and/or non-profit CubeSats. These CubeSats are “piggybacked” on government and commercial satellite launches around the world and have a potential lifetime of 25 years, orbiting the earth approximately every ninety minutes. For this reason, CubeSats and nanosatellites in general are drawing interest as a low-cost platform for space experimentation, particularly in educational environments (Straub et al., 2013; Tsuda et al., 2001).

Due to development cost and complexity, many CubeSat missions are implemented upon flight software systems which employ an open-source OS or RTOS. The Cal Poly SLO PolySat project has launched eight missions using a software architecture based on the UNIX process model. The PolySat employs a Linux distribution for its architecture middleware (Manyak and Bellardo, 2011). The AAUSAT from Aalborg University has employed FreeRTOS as the basis of their flight software (Nielsen et al., 2009; Pedersen, Sorensen and Larsen, 2009). The Salvo RTOS (Kalman, 2004) is included with the Pumpkin CubeSat Kit, and has been used by university nanosatellite projects such as Delft’s Delfi-C3 (Ubbels et al., 2005), which has a command response subsystem like CSUNSat1, but designed atop this RTOS.

Process-based or task-based scheduler systems are undesirable in the case of CSUNSat1, as they are expected to add randomness, and therefore non-determinism, to flight software execution. Like CSUNSat1, Waydo, Henry and Campbell (2002) designed a CubeSat for limited LEO missions using a simple bare-metal loop architecture rather than an RTOS. Although their design is also able to detect single event upsets using telemetry monitoring, their CubeSat is built for far simpler missions, and does not support the rich command parser, the experimental sequence specification, and the anomaly resolution and mitigation subsystem that we have built into our flight software.
Migrating Android Applications to the Cloud
[www.igi-global.com/chapter/migrating-android-applications-cloud/64526?camid=4v1a](www.igi-global.com/chapter/migrating-android-applications-cloud/64526?camid=4v1a)

Internet of Things Applications: Current and Future Development
[www.igi-global.com/chapter/internet-of-things-applications/159055?camid=4v1a](www.igi-global.com/chapter/internet-of-things-applications/159055?camid=4v1a)