A Core Flight Software System

Jonathan Wilmot
Flight Software Branch
Goddard Space Flight Center
Greenbelt, Maryland 20771 USA
Jonathan.J.Wilmot@nasa.gov

ABSTRACT
No two flight missions are alike, hence, development and on-orbit software costs are high. Software portability and adaptability across hardware platforms and operating systems has been minimal at best. Standard interfaces across applications and/or common applications are almost non-existent. To reduce flight software costs, these issues must be addressed. This presentation describes how the Flight Software Branch at Goddard Space Flight Center has architected a solution to these problems.

Categories and Subject Descriptors
D.2 [**Software Engineering**]: General – *software architectures.*

General Terms
Design, Performance, Standardization.

Keywords
Software, Operating System, Interfaces.

1. INTRODUCTION
The Flight Software Branch, at the Goddard Space Flight Center (GSFC), has been working on a solution to one of the major cost drivers for Real-Time Embedded Flight software, **adaptability**, as it applies to both development and on orbit software costs.

Because each mission may have different hardware platforms, software must be customized during development. As processors and interfaces change from project to project, software developers are required to adapt and retest many of the applications at great expense in both time and resources. Even the choice of operating systems can cause significant modification and adaptation as operating system vendors can't agree on common functions such as what a simple open file system call should look like.

On orbit, the software must also be adaptable. Unlike terrestrial systems, flight systems must work around hardware faults and support system degradation due to the harsh space environment without the benefits of having the repairman available. In this environment, the software modification process must be easy and robust to reduce impacts to science or mission operations.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
CODES+ISSS'05, Sept. 19–21, 2005, Jersey City, New Jersey, USA.
Copyright 2005 ACM 1-59593-161-9/05/0009...$5.00.

To help reduce both the development and the on-orbit maintenance costs, GSFC has been working on an evolvable software architecture that supports a run-time plug and play paradigm along with a layered abstraction. One probably takes this for granted on desktop computers, where you can start and stop applications at will, but in the flight software world this has been a long time in coming as the flight hardware platforms are now powerful enough to support this.

2. The Architecture
This architecture, as shown in Figure 1, is built on the concepts of standard interface layers and messaging middleware. Each layer hides the details of the lower layer, promoting portability across the inevitable hardware and interface changes that come with each new flight mission. The messaging layer guarantees that applications can communicate regardless of the underlying platform and even across multiple processors on multiple spacecrafts in larger missions. In this architecture, messaging uses a publish and subscribe model. Applications simply subscribe at run time to the data they need and the messaging middleware delivers it to them. For example, the attitude control software says, send me all sensor data, and the data starts flowing.

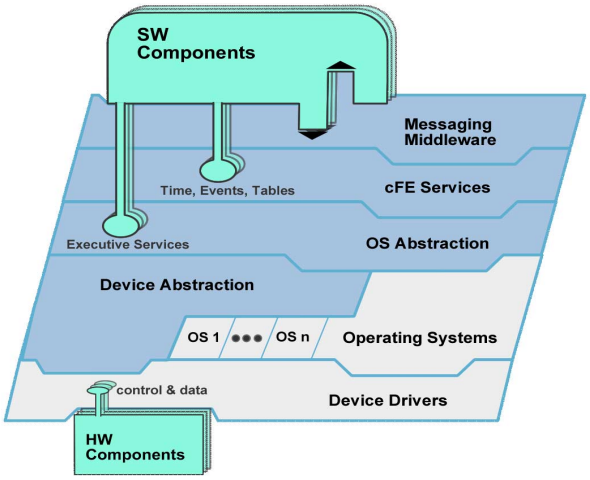

Figure 1. Core Flight Software System Architecture

The plug and play flight system, known as the Core Flight System (CFS), consists of three major parts, the core Flight Executive (cFE) which includes an OS Abstraction, a software component library, and an Integrated Development Environment (IDE) based on the open source Eclipse IDE. The cFE contains the run time services, messaging, and operating system. On top of that sits the standard flight components and mission applications. To help put it all together, the IDE enables users to configure and build the system using a graphical desktop interface.

3. ADVANTAGES of the CFS

This architecture will speed the development process. Using the IDE, engineers will configure the cFE, select some standard applications and load this to the spacecraft processor, allowing a base system to be up and running in a few days. Engineers can then start developing the new software unique to their mission much earlier in the schedule.

This architecture also offers a potential for much greater on orbit flexibility. New science algorithms can be created and uploaded to the flight file system using one of several protocols, started/stopped, tried out, all without impacting other operations. When this is coupled with the protected mode process model of newer operating systems and processors, individual applications can even crash and be restarted automatically without affecting other flight critical systems like communications or attitude control, much unlike previous implementations that needed to restart the entire flight system most likely causing the spacecraft to enter a safe mode and upsetting some of the scientists and mission operators.

4. SUMMARY

We have every expectation that the CFS will help reduce development schedules by reusing existing components, reduce risk by reusing **tested** components, and add flexibility during development and on orbit. This technology was first prototyped and successfully demonstrated in the lab for the Global Precipitation Measurement (GPM) mission in 2004. However, the Lunar Reconnaissance Orbiter (LRO) will be the first mission to capitalize on the advantages of the CFS, enjoying the ease of software integration and in-orbit flexibility..