NASA’s Core Flight Software -
a Reusable Real-Time Framework

Topics:
• Core Flight Software (CFS) Overview
• Case Study: Morpheus Lander
• JSC CFS Development Efforts
• CFS Training Slides

Lorraine Prokop, Ph.D.
Advanced Exploration Systems Core Flight Software Project Manager
NASA – Johnson Space Center (JSC)
November, 2014
What is CFS?
- NASA Agency Asset for Spacecraft Flight Software Reuse (http://cfs.gsfc.nasa.gov/)
  - Productized real-time flight software developed over several years by Goddard Space Flight Center to serve as reusable software framework basis for spacecraft missions, test missions, real-time systems
- Fully tested, documented, operational with LRO spacecraft, several other operational missions since
- Published Service Layer (cFE) and open source Operating System Abstraction Layer (OSAL) for common services
  - Pub/sub message bus, time services, events, tables, file, task execution (http://sourceforge.net/projects/coreflightexec/files/cFE-6.4.0/)
  - Runs on multiple platforms and with several operating systems (http://sourceforge.net/projects/osal/)
- Apps or “bubbles” for common spacecraft functions provided as government open source reuse (available source forge shortly)
  - Scheduler, commanding, telemetry, communication, data recording, limits, system health, sequences

Why use it?
- Proven rapid deployment -- Saves software development/test time, costs, skilled resources
- Provides up-front architectural framework and services needed commonly across spacecraft/real-time embedded command/control applications
  - Don’t have to “reinvent the wheel” every spacecraft for common functions
- Allows ease of development and integration by supporting multiple OS’s and Platforms

In-house experiences with CFS software development
- High software productivity achieved starting with solid architecture (~15+ SLOC/day)
- Ease of application and hardware/software integration
- Decreased verification needed – mature code and architecture – Test Readiness Level (TRL9)
- Excellent product line support from Goddard
CFS Project Use History – Non Exhaustive

Johnson Space Center CFS Usage Timeline

CFS Use in Some Current Spacecraft

Goddard Missions:
- Lunar Reconnaissance Orbiter (LRO) (2009)
- Solar Dynamics Observatory (SDO) (2010)
- Magnetospheric Multiscale Mission (MMS) (2014)
- Global Precipitation Measurement (GPM) (2014)

Ames Research Center Missions:
- Lunar Atmosphere and Dust Environment Explorer (LADEE) (2013)

Applied Physical Lab (APL) Missions:
- Radiation Belt Storm Probes (RBSP) (Aug 2012)
Core Flight Software (CFS) Architecture Overview

Core Flight Software Framework-Architectural Layers

- Mission Specific CFS Apps
- CFS Reusable Apps
- cFE (core Flight Executive) Services API
- Operating System Abstraction Layer (OSAL) API
- Platform Specific Package (PSP)

Notional CFS Application Software Architecture

- Mission Specific Apps - Components
- Hardware Specific Device I/O Apps - Components
- Inter-task Message Router (Software Bus – Publish/Subscribe)

Services:
- Scheduler
- Telemetry Output
- Command Ingest
- CFDP
- File Manager
- Mass Storage Device
- Data Storage
- Limit Checker
- Health & Safety Manager
- Software Bus
- Time Services
- Executive Services
- Event Services
- Table Services
## CFS Supported Platforms

*(non-exhaustive)*

<table>
<thead>
<tr>
<th>Platform</th>
<th>OS</th>
<th>Project</th>
<th>Status / Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAD750</td>
<td>vxWorks 6.4</td>
<td>LR0, RBSP, GPM</td>
<td>Project tested.</td>
</tr>
<tr>
<td>RAD750</td>
<td>RTEMS 4.10</td>
<td>ICESat-2/ATLAS</td>
<td>Early in instrument test program</td>
</tr>
<tr>
<td>Rad Hard Coldfire (S208)</td>
<td>RTEMS 4.10</td>
<td>MMS</td>
<td>Project tested.</td>
</tr>
<tr>
<td>LEON3</td>
<td>RTEMS 4.10</td>
<td>Solar Probe Plus</td>
<td>In Development for SPP mission</td>
</tr>
<tr>
<td>MCP750 PPC</td>
<td>vxWorks 6.4</td>
<td>cFE/CFS Project</td>
<td>Tested. Used as baseline CFS development platform.</td>
</tr>
<tr>
<td>PC / x86</td>
<td>Linux</td>
<td>n/a</td>
<td>Not formally tested. Used by JSC.</td>
</tr>
<tr>
<td>Coldfire MCF5235 board</td>
<td>RTEMS 4.10</td>
<td>n/a</td>
<td>Not formally tested. Used for RTEMS Development, and MMS board.</td>
</tr>
<tr>
<td>LEON3 – generic – (simulator, multiple COTS boards)</td>
<td>RTEMS 4.10</td>
<td>n/a</td>
<td>Not tested. Not in CFS CM. Used for LEON3 development. Can be used on LEON3 Simulator.</td>
</tr>
<tr>
<td>Coldfire Simulator (qemu 68k)</td>
<td>RTEMS 4.10</td>
<td>n/a</td>
<td>Not formally tested. Used for OSAL / cFE development</td>
</tr>
<tr>
<td>TILERA</td>
<td>Linux</td>
<td>Maestro IRAD (FY12)</td>
<td>Not formally tested. Compatible with Desktop PC linux version.</td>
</tr>
<tr>
<td>MCP750 PPC</td>
<td>vxWorks 6.x</td>
<td>Memory Protection IRAD (FY11)</td>
<td>Adds memory protection to standard cFE. Not formally tested. Not integrated with cFE repository.</td>
</tr>
<tr>
<td>PC x86</td>
<td>Linux</td>
<td>Multi-Core IRAD (FY12)</td>
<td>Adds multi-core CPU capability to cFE. Not formally tested. Not integrated with cFE repository.</td>
</tr>
<tr>
<td>Leon3</td>
<td>PikeOS</td>
<td>Virtualization IRAD (FY12)</td>
<td>Adds ability to run in partitioned OS. Prototype. Not integrated with cFE repository.</td>
</tr>
</tbody>
</table>

### Platform Details

<table>
<thead>
<tr>
<th>Platform</th>
<th>OS</th>
<th>Project</th>
<th>Status / Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>AiTech S950 (PPC750FX)</td>
<td>vxWorks 6.7</td>
<td>Morpheus</td>
<td>In JSC CM. Integration tested on real Morpheus Vehicle hardware. Flown on Morpheus test vehicle.</td>
</tr>
<tr>
<td>RTD pc386-IDAN, PC104, Pentium M</td>
<td>RTEMS 4.10</td>
<td>ISS Downmass/ Micro Capsule</td>
<td>In JSC CM. Integration tested on real Micro Capsule hardware.</td>
</tr>
<tr>
<td>Acro Virtex 5</td>
<td>VxWorks 6.9</td>
<td>AEMU</td>
<td>In development.</td>
</tr>
<tr>
<td>Space Micro Proton P400k</td>
<td>VxWorks SMP 6.8</td>
<td>MMSEV, AAE</td>
<td>In JSC CM. In development for MMSEV FY13 work.</td>
</tr>
<tr>
<td>Maxwell SC575</td>
<td>VxWorks 6.9</td>
<td>EAM, AAE</td>
<td>In JSC CM. EAM about to start using.</td>
</tr>
<tr>
<td>787FCM</td>
<td>Integrity ARINC</td>
<td>AES CFS</td>
<td>In development, producing ARINC653 cFE, OSAL.</td>
</tr>
<tr>
<td>OrionSCP</td>
<td>Integrity ARINC</td>
<td>AES CFS</td>
<td>In development, producing ARINC653 cFE, OSAL.</td>
</tr>
<tr>
<td>750FCR</td>
<td>VxWorks ARINC 6.8</td>
<td>AES CFS</td>
<td>In development, testing FTSS SW fault containment with a voting quad architecture.</td>
</tr>
<tr>
<td>Trick (simulation environment)</td>
<td>Linux</td>
<td>AES CFS</td>
<td>In development, for multi-project use.</td>
</tr>
<tr>
<td>LEON3</td>
<td>VxWorks 6.7</td>
<td>BFS</td>
<td>In JSC CM. BFS prototype.</td>
</tr>
<tr>
<td>AiTech SP0</td>
<td>VxWorks 6.7</td>
<td>RPM?</td>
<td>In JSC CM. RPM performance analysis.</td>
</tr>
</tbody>
</table>

Recently Developed largely in support of AES projects
Broad Awareness/Use of the CFS

- **ARC** - Using on LADEE (flight SW system on budget/schedule)
- **JPL** – Evaluating architecture for robotic missions and ESTO missions, DTN
- **Commercial** - Moon Express (Lunar X-Prize)
- **DOD**
  - Potential for standardization through Office of Director of National Intelligence (ODNI)
  - 2012- met w/ Space Universal MOdular Architecture (SUMO) team led by Office of Director for National Intelligence (James Afarin (HQ))
- **GRC** – Using on CPST, Suits
- **JSC** - Used Successfully on Morpheus. Using on Habitats and Suits (AES) – enhancing for human rated software.
- **APL** - Successfully used on RBSP. Proposing use on Solar Probe, DoD programs.
- **GSFC** - Successfully on LRO, using on MMS, GPM, instruments. Plans for NICER, several others.
- **MSFC** - Mighty Eagle Lander, prototyping for AES
- **KSC** - Evaluating for AES, sounding rockets and UAV’s
- **South Korea Aerospace Research Institute (KARI)** - Lunar Exploration Research Team

Case Study: Project Morpheus

Introduction

- Morpheus is a Full Scale Robotic Lander (500kg payload) built as a risk reduction test article
  - Morpheus system includes the vehicle, ground systems, operations
  - Developed, tested and operated in-house at Johnson Space Center and KSC

- Example Video: [http://www.youtube.com/watch?v=tdrSYP2gSbg](http://www.youtube.com/watch?v=tdrSYP2gSbg)

- Technologies:
  - Liquid oxygen/methane propulsion (cryogenic, green, safe for ground handling and crew)
  - Precision landing and hazard detection Sensors
  - Leverages GSFC’s modular, reusable Core Flight Software
  - Technology incubator for advanced development efforts

While technologies offer promise, capabilities offer potential solutions with application for future human exploration beyond LEO. Morpheus provides a bridge for evolving these technologies into capable systems that can be demonstrated and tested – in a relevant flight environment.

- Tests complete: 12 hot fire, 34 tethered, and 14 free flights to date

- Lean Development Approach
void XXX_AppMain()
{
    /* Perform application initializations */
    if (XXX_InitApp() != CFE_SUCCESS)
    {
        g_XXX_AppData.uiRunStatus = CFE_ES_APP_ERROR;
    }

    /* Application main loop */
    while (CFE_ES_RunLoop(&g_XXX_AppData.uiRunStatus) == TRUE)
    {
        XXX_RcvMsg(CFE_SB_PEND_FOREVER);
    }

    /* Exit the application */
    CFE_ES_ExitApp(g_XXX_AppData.uiRunStatus);
}

int32 XXX_InitApp()
{
    int32 iStatus=CFE_SUCCESS;
    g_XXX_AppData.uiRunStatus = CFE_ES_APP_RUN;

    iStatus = CFE_ES_RegisterApp();
    if (iStatus != CFE_SUCCESS)
    {
        CFE_ES_WriteToSysLog("XXX - Failed to register the app (0x%08X)\n", iStatus);
        goto XXX_InitApp_Exit_Tag;
    }

    if ((XXX_InitEvent() != CFE_SUCCESS) ||
        (XXX_InitPipe() != CFE_SUCCESS) ||
        (XXX_InitData() != CFE_SUCCESS))
    {
        iStatus = -1;
        goto XXX_InitApp_Exit_Tag;
    }

    /* Install the cleanup callback */
    OS_TaskInstallDeleteHandler((void*)&XXX_CleanupCallback);

    XXX_InitApp_Exit_Tag:
    if (iStatus == CFE_SUCCESS)
    {
        CFE_EVS_SendEvent(XXX_INIT_INF_EID,
        CFE_EVS_INFORMATION,
        "XXX - Application initialized");
    }
    else
    {
        CFE_ES_WriteToSysLog("XXX - Application failed to initialize\n");
    }
}
```c
int32 XXX_RcvMsg(int32 iBlocking)
{
    int32 iStatus=CFE_SUCCESS;
    CFE_SB_Msg_t* MsgPtr=NULL;
    CFE_SB_MsgId_t MsgId;

    /* Wait for WakeUp messages from scheduler */
    iStatus = CFE_SB_RcvMsg(&MsgPtr, g_XXX_AppData.SchPipeId, iBlocking);

    /* Start Performance Log entry - create initial entry */
    CFE_ES_PerfLogEntry(XXX_MAIN_TASK_PERF_ID);

    if (iStatus == CFE_SUCCESS)
    {
        MsgId = CFE_SB_GetMsgId(MsgPtr);
        switch (MsgId)
        {
        case XXX_WAKEUP_MID:
            XXX_ProcessNewCmds();
            XXX_ProcessNewData();
            /* TODO: Add more code here to handle other things when app wakes up, like any cyclic processing */

        /* The last thing to do at the end of this Wakeup cycle should be to automatically publish new output. */
        XXX_SendOutData();
        break;

        /* TODO: Add code here to handle other command IDs, if needed. Normally, other app commands are added as command codes to the app's CMD_MID and processed in XXX_ProcessNewCmds(). Adding another CMD_MID would also require adding another command pipe. */

        default:
            CFE_EVS_SendEvent(XXX_MSGID_ERR_EID, CFE_EVS_ERROR, "XXX - Recvd invalid SCH msgId (0x%08X)", MsgId);
        }
    } else if (iStatus == CFE_SB_NO_MESSAGE)
    {
        /* If there's no incoming message, you can do something here, or do nothing */
    } else
    {
        /* This is an example of returning on an error. ** Note that a SB read error is not always going to result in an app quitting, depends on the app. Changing the run status to ** CFS_ES_APP_ERROR will cause the app's main loop to exit and the ** app to exit. */

        CFE_EVS_SendEvent(XXX_PIPE_ERR_EID, CFE_EVS_ERROR, "XXX: SB pipe read error (0x%08X), app will exit", iStatus);
        g_XXX_AppData.uiRunStatus= CFE_ES_APP_ERROR;
    }

    /* Stop Performance Log entry */
    CFE_ES_PerfLogExit(XXX_MAIN_TASK_PERF_ID);

    return (iStatus);
}
```
Morpheus Simulation
Morpheus Ground Systems – ITOS Control Room
What is ITOS (Integrated Test and Operations System)?
• A low-cost, highly configurable, control and monitoring system

What are its current applications?
• Satellite development, test, & operations
• Science instrument development, test, & operations
• Ground station equipment monitoring & control

Who is using ITOS?
• SAMPEX, TRACE, FAST, SWAS, WIRE,
• Spartan 201, 251, 401, 402
• HESSI, Swift, ULDB, Triana
• PiVot GPS, CIRS, Mars Pathfinder

Who is commercializing ITOS?
• Universal Space Network
• the Hammers Company
• Omitron
• AlliedSignal Technical Services Corporation

From ITOS Promo Presentation: http://itos.gsfc.nasa.gov/
Core Flight Software

Lorraine Prokop, Ph.D. / JSC
Objectives

- Provide a *reusable* software architecture suitable for human-rated missions
  - Reduce/offset per-project software development, test, and certification costs by performing that work *once* serving multiple projects
  - Address software and hardware issues unique or typical to human-rated systems
- Provide reusable software products, tools, and artifacts directly usable by Class A projects/programs, and for general use across NASA
- Support Advanced Exploration Systems projects as they develop toward flight missions

The Core Flight Software Project’s objective is to evolve and extend the reusability of the Core Flight Software System into human-rated systems, thus enabling low cost, and rapid access to space.

Build upon reuse of existing TRL-9 uncrewed spacecraft software framework for utilization in human-rated programs.

Leverage platforms, resources and skills from synergetic programs/projects for development of next generation human-rated space software systems.

Utilize these products in direct support of development and certification of future manned programs.
CFS AES Project
Product Summary to Date

◆ FY13 Products
  • Quad-Voting CFS System – CFS on Partitioned VxWorks RTOS, synchronizing & voting 4 computers
  • CFS within Trick Simulation
  • Distributed CFS – network-based software bus
  • CFS on Orion/B787 Platform – CFS on Partitioned Green Hills RTOS
  • Reusable Certification Test Suite

◆ FY14 Products
  • Class A CFS Certification on Orion Platform
  • Performance Monitoring Tool Development
  • CFS Synch & Voting Software Development
  • Symmetric Multicore Processor (SMP) CFS Development
  • Product Line
  • Command & Data Dictionary Ground Database Tools
  • Education/Outreach
  • Orion Backup Computer Proof of Concept Demonstration
Flight Computer Architecture: CFS on ARINC and Voting Partition

- Four fault-containment regions (FCRs)
  - 4 Flight Critical Computers (FCC)
- Software voting
- Ethernet
- Will accommodate 2 arbitrary non-simultaneous faults
Synchronization & Voting
Embedded CFS-Trick Background
Flight Software - Simulation Philosophies

- **HWITL “Iron Bird”**
- **External Sim**
  - **Flight Software** (flight hardware)
  - **Flight Software** (non-flight hardware)
  - **Simulation Software** (test RIG)
  - **Simulation Software** (simulation computer)

**Increasing Fidelity**
- **Socket communication** (single computer, separate executable)
- **Flight Software** (single computer, single executable)

**Non-Flight I/O Interfaces** (typically Ethernet)
- **Simulation Software** (simulation computer)
- **Simulation Software** (simulation computer)

- Typically this flight software is not REAL, but an algorithmic prototype/analog
- Allows SAME source code to run in ALL configurations
- Allows analysis, faster-than-real-time execution, data inspection, debugging
Distributed CFS Demo Configuration

**CPU A**
- Scheduler: 40Hz
- Test App 1: 40Hz
- Telemetry Output: 5Hz
- Command Ingest: 5Hz
- Network Software Bus (sbn): 80Hz

**CPU B**
- Scheduler: 40Hz
- Test App 2: 40Hz
- Telemetry Output: 5Hz
- Command Ingest: 5Hz
- Network Software Bus (sbn): 80Hz

**CPU C**
- Test App 3: 20Hz
- Scheduler: 20Hz

**Ground Display Computer (ITOS)**
- CFS Distributed Demo - "sbn"
  - CPU A telemetry
  - CPU B telemetry
  - CPU C telemetry
  - Cmd Count A
  - Cmd Count B
  - CPU Indicator

**Ethernet**
- sbn over IP comm
- CCSDS over IP comm

**Wireless**
- 80Hz

**Local Display (Java)**
- CPU Count
- CPU Count
- CPU Count

AES Continuation Review - Sep 2013
CFS on Partitioned OS/B787
Class A Product Team

ut_cfe_evs PASSED 175 tests.
u_cfe_evs FAILED 0 tests.
Performance Monitoring Tool Screenshots
Voting System for Fault Tolerance

- **Description**
  - Provides CFS framework solution for synchronization/redundancy between flight computers

- **Accomplishments**
  - Designed System, held several design Inspections, held Demonstrations
  - Implementation underway
  - Supported Heterogeneous Voting Computer Demonstration 9/17/2014

- **Remaining Work (FY15)**
  - Continue development
  - Improve system robustness/reliability
  - Analyze/Improve Performance
  - Support Time Triggered Systems
Symmetric Multiprocessing (SMP) Support

- **Description**
  - Provide a generic SMP Operating System Abstraction Layer (OSAL) supporting multi-core processor architectures

- **Accomplishments**
  - Prototype implementation of CFS on dual core Space Micro Proton board and VxWorks SMP complete
    - Apps can be allocated to specific cores to deterministically balance processing load or to improve performance of certain apps

- **Remaining Work (FY15)**
  - Implement on SPARC LEON 4 quad-core, Tilera 36-core
  - Merge SMP support modifications into mainline CFS
Mobile Command and Telemetry System

- KSC developed general purpose data integration tool for managing command and telemetry metadata
- Intended to be generic in nature and applicable to any project using CFS or ITOS
- Web based interface built with Ruby on Rails
- Data can be ingested from a variety of formats including flat text files or Excel spreadsheets
- Imported into PostgreSQL relational database on which a wide variety of queries and reports can be run from MCTS provided GUI screens
- Currently capable of exporting data directly into ITOS compatible data record format
- Future enhancements include exporting data to XTCE format files as well as ‘C’ type data structure statements for compiling into CFS application code
- Demonstration held August 2014
Education/Course Idea: CFS on AR Drone Embedded with Trick Controls & Simulation
CFS Project “To Do List”
FY14 Work, FY15 Planned

- Class A Products, Human Ratable
  - Certify Class A on Orion primary Platform
  - Certify Class A on Orion backup (vxWorks/LEON3) Platform

- Testing
  - Reusable test suite additions for vxWorks
  - Cross-platform test framework
  - White-box testing of OSAL layer
  - Integrated unit test execution/post processing/reports
  - Build interface/instrument CFS code for performance testing, monitoring, display interface
  - Reusable performance test suite

- Human Spacecraft Support Activities
  - Support for Redundancy
    - Symmetric (same OS & shared mem) Multiprocessor Support (SMP)
      (Dual core, 4 core, 36 core)
    - Asymmetric Multiprocessor CFS support
      - Open source Quad CFS voting layer (continued in FY15)
  - VML – (virtual machine language) integration w/ CFS
  - Support for Distributed Systems (sbn additions)
  - User Interface Display Support – OpenGL Interface
  - Backup Flight Systems Architecture exploration

- Development Tools - Productivity / Interoperability
  - Performance Monitoring / Profiling Tool (Linux/Java)
  - Data Definition / Ground Integration Tools (continued FY15)
  - Autogeneration of application from a variety of tools - Matlab/Simulink/Rhapsody/sysML/Eclipse,
  - Matlab/Simulink simulation of CFS layers
  - Top-Coder effort to start with CodeReview Redmine Tool

- Additional Operating Systems / Hardware Platforms
  - iOS
  - Other real-time: real-time Linux, eCos
  - Additional Hypervisor prototyping- picos
  - FPGA with soft cores, PSP’s for hybrid chips with hard cores

- Specific Support Needed or AES Projects
  - DTN-CFS integration development
  - AMO-CFS integration
  - AAE project platforms / chosen architectures
  - RPM development
  - Exploration Augmentation Module development
  - Advanced EVA development support

- Outreach Maturation – Quad Copter
  - Develop Sim of Quad Copter, Basic GNC Apps
  - Develop product distribution for outreach (CFS, Apps & Trick)

- CFS Institutional Support/Infrastructure
  - Configuration Control, evolution, product planning
  - Website: how-to, wiki, FAQ, downloads
  - Product support & releases, training
  - SARB Recommended fixes

- Possible Flight Projects
  - ISS Flight Computer shadow
  - Orion Backup flight computer prototype, Leon3 processor
  - Software partition for Asteroid Retrieval Mission
Core Flight Software System (CFS)/Core Flight Executive (cFE) Training Material

Jonathan Wilmot
GSFC/Code 582
Jonathan.J.Wilmot@nasa.gov
301-286-2623
cFE - Overview

• **A set of mission independent, re-usable, core flight software services and operating environment**
  – Provides standardized Application Programmer Interfaces (API)
  – Supports and hosts flight software applications
  – Applications can be added and removed at run-time (eases system integration and FSW maintenance)
  – Supports software development for on-board FSW, desktop FSW development and simulators
  – Supports a variety of hardware platforms
  – Contains platform and mission configuration parameters that are used to tailor the cFE for a specific platform and mission.

• **cFE services include:**
  – Executive Services
  – Software Bus Services
  – Time Services
  – Event Services
  – Table Services

• **Layered on the Operation System Abstraction**
Motivation

• About six years ago GSFC was tasked two large in-house missions with concurrent development schedules (SDO, GPM)

• GSFC was to build the spacecraft bus, both avionics and software, and integrate the whole spacecraft

• Without the staff for both, we were directed to find a better way

• So management said, “you engineers figure out how to make the schedule and keep the cost in line”
  o We had about a year to figure it out before staffing up
    • This is before full cost accounting
Approach

• Formed a team of senior FSW engineers to strategize and develop a better way
• Each had experience on a few different missions and immediately saw all the commonality we could have had
• Team then decided to:
  – Determine impediments to good flight software reuse
  – Utilize best concepts from missions ranging from Small Explorer class to the Great Observatories
  – Design with reusability and flexibility in mind
  – Take advantage of software engineering advances
  – Be Composable

• Management helped isolate team engineers from short term mission schedules

• Team established architecture goals
Goals

1. Reduce time to deploy high quality flight software
2. Reduce project schedule and cost uncertainty
3. Directly facilitate formalized software reuse
4. Enable collaboration across organizations
5. Simplify sustaining engineering (AKA. On Orbit FSW maintenance) Missions last 10 years or more
6. Scale from small instruments to Hubble class missions
7. Build a platform for advanced concepts and prototyping
8. Create common standards and tools across the center
Heritage, what worked well

- **Message bus**
  - All software applications use message passing (internal and external)
  - CCSDS standards for messages (commands and telemetry)
  - Applications were processor agnostic (distributed processing)
- **Layering**
- **Packet based stored commanding (AKA Mission Manager)**
  - Absolute Time Sequence (ATP), Relative Time Sequence (RTP)
- **Vehicle FDIR based on commands and telemetry packets**
- **Table driven applications**
- **Critical subsystems time-triggered on network schedule**
  - 1553 bus master TDMA
- **Clean application interfaces**
  - Component based architecture (The Lollipop Diagram)
Heritage, what worked well

- **Lots of innovation**
  - Constant pipeline of new and varied missions
  - Teams worked full life cycle
    - Requirements through launch + 60 days
    - Maintenance teams in-house and in contact with engineers early in development
  - Teams keep trying different approaches
    - Rich heritage to draw from
Heritage: what didn’t work so well

- **Statically configured Message bus**
  - Scenario: GN&C needs a new diagnostic packet
    - Give the C&DH team your new packet definition file
    - Wait a week for a new interim build
    - Rinse and Repeat
  - How do I add a new one on orbit? (FAST mission example)

- **Monolithic load (The “Amorphous Blob”)**
  - Raw memory loads and byte patching needed to keep bandwidth needs down

- **Reinventing the wheel**
  - Mission specific common services (“Look, I’ve got a new and improved version!”)

- **Application rewrites for different OSes**
Re-use in the Past

- In the past, GSFC’s Flight Software Branch (FSB) has realized little cost savings via FSW reuse
  - No product line. Instead heritage missions were used as starting point
  - Changes made to the heritage software for the new mission were not controlled
    - New flight hardware or Operating System required changes throughout FSW
    - FSW Requirements were sometimes re-written which effects FSW and tests.
    - FSW changes were made at the discretion of developer
    - FSW test procedure changes were made at the discretion of the tester
    - Extensive documentation changes were made for style
  - Not all Products from heritage missions were available
  - Reuse was not an formal part of FSB development methods
  - Reuse was not enforced
Concepts and Standards

- Layered Architecture
- Standard Middleware/Bus
- Standard Application Programmer Interface for a set of core services
- Plug and Play Reusable Applications
- Command & Telemetry database
- Reuse Requirements Management
- Reuse Standards
- Reuse Repository
- Configuration Tool for Mission Users
- Development Tools

Core Flight Executive (cFE)

CFS Applications

Library & CM

Integrated Development Environment
Standard Middleware Bus

Publish/Subscribe

- Components communicate over a standards-based Message-oriented Middleware/Software Bus.
- The Middleware/Software Bus uses a run-time Publish/Subscribe model. Message source has no knowledge of destination.
- No inherent component start up dependencies

Impact:

- Minimizes interdependencies
- Supports HW and SW runtime “plug and play”
- Speeds development and integration.
- Enables dynamic component distribution and interconnection.

Legacy: Tightly-coupled, custom interfaces- data formats - protocols, internal knowledge, component interdependence

Publish/Subscribe: loosely-coupled, standard interface, data formats, protocols, & component independence
Application Programmer Interfaces

- CFS services and middleware communication bus has a standardized, well-documented API
- An abstracted HW component API enables standardized interaction between SW and HW components.

Impact:
- Allows development and testing using distributed teams
- With the framework already in place, applications can be started earlier in the development process
- Can do early testing and prototyping on desktops and commercial components
- Simplifies integration

API supplies all functions and data components developers need.
Plug and Play

- cFE API’s support add and remove functions
- SW components can be switched in and out at runtime, without rebooting or rebuilding the system SW.
- Qualified Hardware and CFS-compatible software both “plug and play.”

Impact:

- Changes can be made dynamically during development, test and on-orbit even as part of contingency management
- Technology evolution/change can be taken advantage of later in the development cycle.
- Testing flexibility (GSE, test apps, simulators)

This powerful paradigm allows SW components to be switched in and out at runtime, without rebooting or rebuilding the system SW.
Reusable Components

- Common FSW functionality has been abstracted into a library of reusable components and services.
- Tested, Certified, Documented
- A system is built from:
  - Core services
  - Reusable components
  - Custom mission specific components
  - Adapted legacy components

Impact:
- Reuse of tested, certified components supplies savings in each phase of the software development cycle
- Reduces risk
- Teams focus on the custom aspects of their project and don’t “reinvent the wheel.”
## Sample CFS Reusable Applications

<table>
<thead>
<tr>
<th>Application</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command Ingest</td>
<td>Reusable component for spacecraft commanding</td>
</tr>
<tr>
<td>Telemetry Output</td>
<td>Reusable component for sending and packaging telemetry</td>
</tr>
<tr>
<td>CFDP</td>
<td>Transfers/receives file data to/from the ground</td>
</tr>
<tr>
<td>Checksum</td>
<td>Performs data integrity checking of memory, tables and files</td>
</tr>
<tr>
<td>Data Storage</td>
<td>Records housekeeping, engineering and science data onboard for downlink</td>
</tr>
<tr>
<td>File Manager</td>
<td>Interfaces to the ground for managing files</td>
</tr>
<tr>
<td>GN&amp;C Framework</td>
<td>Provides framework for plugging in ACS models and objects</td>
</tr>
<tr>
<td>Housekeeping</td>
<td>Collects and re-packages telemetry from other applications.</td>
</tr>
<tr>
<td>Health and Safety</td>
<td>Ensures that critical tasks check-in, services watchdog, detects CPU hogging, and calculates CPU utilization</td>
</tr>
<tr>
<td>Limit Checker</td>
<td>Provides the capability to monitor values and take action when exceed threshold</td>
</tr>
<tr>
<td>Math Libraries</td>
<td>Scalar, vector, matrix and quaternion functions</td>
</tr>
<tr>
<td>Memory Dwell</td>
<td>Allows ground to telemeter the contents of memory locations. Useful for debugging</td>
</tr>
<tr>
<td>Memory Manager</td>
<td>Provides the ability to load and dump memory.</td>
</tr>
<tr>
<td>Scheduler</td>
<td>Schedules onboard activities (eg. hk requests)</td>
</tr>
<tr>
<td>Stored Command</td>
<td>Onboard Commands Sequencer (absolute and relative).</td>
</tr>
</tbody>
</table>
Health and Safety App / Housekeeping App

• Health and Safety App
  – Monitor Applications
    • Detect when defined applications are not running and take a defined action
  – Monitor Events
    • Detect table defined events and take a table defined action
  – Manage Watchdog
    • Initialize and periodically service the watchdog
    • Withhold periodic servicing of the watchdog if certain conditions are not met
  – Manage App Execution Counters
    • Report execution counters for a table defined list of Application Tasks

• Housekeeping App
  – Build combined telemetry messages containing data from applications
  – Notify the ground when expected data is not received
Data Storage App / File Manager App

- **Data Storage App**
  - Stores Software Bus messages (packets) to data storage files.
  - Filters packets according to packet filter table definition
  - Stores packets in files according to destination table definition

- **File Manager App**
  - Manages onboard files
    - Copy, Move, Rename, Delete, Close, Decompress, and Concatenate files providing file information and open file listings
  - Manages onboard directories
    - Create, delete, and providing directory listings
  - Device free space reporting
Limit Checker App / Memory Dwell App

• **Limit Checker App**
  – Monitors Table Driven Telemetry Watch points
    • Each watch point compares a telemetry data value with a constant threshold value
  – Evaluates Table Driven Action points
    • Each action point analyzes the results of one (or more) watch points

• **Memory Dwell App**
  – Samples data at any processor address
  – Augments telemetry stream provided during development and debugging
  – Dwell Packet Streams are Specified by Dwell Tables
  – Up to 16 active Dwell Tables
  – Dwell Tables can be populated either by Table Loads or via Jam Commands
Scheduler App / Stored Command App

- **Scheduler App**
  - Operates a Time Division Multiplexed (TDM) schedule of Applications via Software Bus Messages
  - Synchronized to external Major Frame (typically 1 Hz) signal
  - Each Major Frame split into a platform configuration number of smaller slots (typically 100 slots of 10 milliseconds each)
  - Each slot can contain a platform defined number of software bus messages (typically 5 messages) that can be issued within that slot

- **Stored Command App**
  - Executes preloaded command sequences at predetermined absolute or relative time intervals.
  - Supports Absolute Time Tagged Sequences
  - Supports Relative Time Tagged Sequences
Checksum App / Memory Manager App

- **Checksum App**
  - Monitors the static code/data specified by the users and reports all checksum miscompares as errors.
  - CS will be scheduled to wakeup on a 1Hz schedule
  - CS will be byte-limited per cycle to prevent CPU hogging

- **Memory Manager App**
  - Performs Memory Read and Write (Peek and Poke) Operations
  - Performs Memory Load and Dump Operations
  - Performs Diagnostic Operations
  - Provides Optional Support for Symbolic Addressing
Other CFS Apps

- **CFDP App**
  - Implements flight portion of CCSDS CFDP Protocol

- **Command Uplink App**
  - Implements flight portion of CCSDS Command uplink
  - Usually mission specific

- **Telemetry Output App**
  - CCSDS Telemetry downlink
  - Usually mission specific

- **Memory Scrub App**
  - Memory Scrub – Scrubs SDRAM check bits
  - Usually mission specific

- **CI Lab & TO Lab**
  - UDP sockets based uplink and downlink apps for lab testing
Component Example

• Interface only through core API’s.

• A component contains all data needed to define its operation.

• Components register for services
  • Register exception handlers
  • Register Event counters and filter
  • Register Tables
  • Publish messages
  • Subscribe to messages

• Component may be added and removed at runtime. (Allows rapid prototyping during development)
cFE Core - Overview

- A set of *mission independent, re-usable, core flight software services and operating environment*
  - Provides standardized Application Programmer Interfaces (API)
  - Supports and hosts flight software applications
  - Applications can be added and removed at run-time (eases system integration and FSW maintenance)
  - Supports software development for on-board FSW, desktop FSW development and simulators
  - Supports a variety of hardware platforms
  - Contains platform and mission configuration parameters that are used to tailor the cFE for a specific platform and mission.
cFE Core - Executive Services (ES)

- Manages the cFE Startup
- Provides ability to start, restart and delete cFE Applications
- Manages a Critical Data Store which can be used to preserve data (except in the case of a power-on reset)
- Provides ability to load shared libraries
- Logs information related to resets and exceptions
- Manages a system log for capturing information and errors
- Provides Performance Analysis support
cFE Core - Software Bus (SB)

- Provides a portable inter-application message service
- Routes messages to all applications that have subscribed to the message.
  - Subscriptions are done at application startup
  - Message routing can be added/removed at runtime
- Reports errors detected during the transferring of messages
- Outputs Statistics Packet and the Routing Information when commanded
cFE Core - Event Services (EVS)

- Provides an interface for sending asynchronous informational/error messages telemetry to ground
  - Provides a processor unique software bus event message containing the processor ID, Application ID, Event ID, timestamp, and the request-specified event data (text string including parameters)
- Provides an interface for filtering event messages
- Provides an interface for registering an application’s event filter masks, types, and type enable status
- Provides an interface for un-registering an application from using event services
- Provides an interface for enabling/disabling an application’s event filtering
- <optional> Provide an interface for logging event into a local event log
cFE Core - TIME Services

- Provides a user interface for correlation of spacecraft time to the ground reference time (epoch)
- Provides calculation of spacecraft time, derived from mission elapsed time (MET), a spacecraft time correlation factor (STCF), and optionally, leap seconds
- Provides a functional API for cFE applications to query the time
- Distributes a “time at the tone” command packet, containing the correct time at the moment of the 1Hz tone signal
- Distributes a “1Hz wakeup” command packet
- Forwards tone and time-at-the-tone packets
cFE Core - Table Services

- Manages all CFS table images
- Provides an API to simplify Table Management
- Table Registry is populated at run-time eliminating cross coupling of Applications with flight executive at compile time
- Performs table updates synchronously with the Application that owns the table to ensure table data integrity
- Shares tables between Applications
- Allows Non-Blocking Table updates in Interrupt Service Routines
- Provides a common ground/user interface to all tables
Operating System Abstraction Layer (OSAL) Overview

- A standalone project, separate from the cFE
  - The cFE is built on the OSAL to provide portability
- Available as Open Source on NASA’s Open Source Website
- Allows execution of FSW on multiple Real Time OSs
  - Build Verification testing done using VxWorks 6.4
- Allows execution of FSW on simulators and desktop computers
- Support three primary targets
  - POSIX
    - OSX
    - Linux
    - Cygwin
  - RTEMS 4.10
  - VxWorks 6.x
Platform Specific Package Overview

- **Supports the following Hardware Platforms/Operating Systems (non exhaustive)**
  - **Flight Hardware Environments**
    - MCP750/vxWorks 6.x
    - BAE RAD750/VxWorks 6.x
    - Coldfire/RTEMS 4.x
    - MCP405/linux (Spacecube)
  - **Desktop FSW Test Environments**
    - MAC/OSX
    - MAC/linux
    - PC(x86)/linux, Cygwin