Designing Command and Data Handling (C&DH) Subsystems from Software Architectural Design Patterns

Julie Fant\textsuperscript{1}, Hassan Gomaa\textsuperscript{2}, and Robert Pettit\textsuperscript{1}
The Aerospace Corporation\textsuperscript{1} and George Mason University\textsuperscript{2}

Computer and Software Division
The Aerospace Corporation
December 7, 2010
Outline

• Motivation for this research

• Research approach

• Distributed, real-time and embedded (DRE) software design patterns

• Deriving DRE software design patterns for flight software

• Real world case study example

• Conclusions and future work
Motivation for this research

- Software design patterns are best practice solutions to common software problems
  - Avoid reinventing the wheel
  - Improvement in the -ilities

- However, software design patterns can be difficult to apply in practice
  - Abstract and generic descriptions
  - Can be applied at several different layers of abstraction

- Taking advantage of design patterns is particularly import for the FSW domain
  - Increased FSW responsibilities has led to additional complexity and a greater number of software related anomalies.
    - “In the period from 1998 to 2000, nearly half of all observed spacecraft anomalies were related to software” [1]

  - NASA’s Study on Flight Software Complexity Report examined flight software complexity and provided a series of recommendations to better manage the associated challenge.
    - This presentation aligns with their recommendation to perform early analysis and architesting [2]
Research Approach

• Select and categorize software design patterns for common DRE features
  – A feature is a requirement or characteristic of the system that is provided by some DRE systems
  – Assuming DRE systems are built on top of RTOS therefore patterns for RTOS features are not included

• Create executable UML design pattern templates for each pattern
  – Validated for functional correctness using executable statecharts

• Select and augment DRE design patterns for the FSW domain

• Customize the executable UML design pattern templates for FSW

Flight Software is a specialized type of DRE software therefore design patterns for flight software can be derived from DRE design patterns
DRE Common Features

- Architecture Structure
  - Decompose DRE software from a high-level perspective
  - Used early on in the architecting process since they provide the structural foundation of the software

- Control Feature
  - Controlling the system, subsystems, or external devices
  - Control can be handled and structured in variety of different ways depending on the requirements

- Data Processing Feature
  - Performing a single or series of transformational steps on a data
  - Examples include converting a set of data from one format to another and converting an analog signal into digital data

- Data Collection Feature
  - Gathering data from various sources
  - This includes pulling from the sources or pushing data to the receivers
  - Examples include monitoring system health and gathering data from input devices

- Data Sharing Feature
  - Strategies for ensuring data integrity when multiple concurrent components need to read and write data

- Fault Detection Feature
  - Detecting when a fault has occurred
  - Particularly important for DRE systems with autonomy because it needs be able to determine that a fault has occurred before it can execute the proper response

- Fault Response Feature
  - Ability to respond to a fault
  - Particularly important for DRE systems with autonomy because it needs to know how to respond to a fault when a human is not in the loop

- Communication Feature
  - Specifics of how components exchange information with each other
  - Cross cut the other features and therefore these patterns are intended to be used combination with the other feature groups
Design Patterns for DRE Common Features

- **Architecture Structure**
  - Layers
  - 5 Layer Architecture
  - Whole-Part

- **Control Feature**
  - Centralized Control
  - Hierarchical Control
  - Distributed Control
  - State
  - Command Processor
  - Command Dispatcher

- **Data Processing Feature**
  - Pipes and Filters
  - Master Slave
  - Strategy

- **Data Collection Feature**
  - Pulling Data from the Source
    - Client Server
    - Multiple Client Multiple Server
    - Multitier Client Server
  - Pushing Data to the Receiver
    - Publish Subscribe
    - Broadcast
    - Multicast

- **Data Sharing Feature**
  - Two Phase Commit
  - Compound Commit
  - Long-Living Transaction

- **Fault Detection Feature**
  - Watchdog
  - Single Protected Channel
  - Homogeneous Redundancy
  - Heterogeneous Redundancy
  - Triple Modular Redundancy
  - Monitor Actuator
  - Sanity Check

- **Fault Response Feature**
  - Safety Executive

- **Communication Feature**
  - Asynchronous
  - Bidirectional Asynchronous
  - Asynchronous with Callback
  - Synchronous with Reply
  - Synchronous without Reply
  - Broker Forwarding
  - Broker Handle
  - Discovery
  - Publish Subscribe
  - Broadcast
  - Multicast
Executable UML Design Pattern Template Example

- **Centralized Control Pattern**
  - *One control component that provides overall control by conceptually executing a state machine*

- **How it can be used to achieve Control feature**
  - *It provides a control structure*
  - *All control logic is captured in a single component, which makes it easy to understand and maintain*
  - *Suitable for small applications because the centralized control can become a bottleneck*

- **Collaboration Diagram**
  - *Shows the components roles involved in the design pattern and their high-level interactions*
Executable UML Design Pattern Template Example (cont.)

- State machines
  - Captures the internal behavior of a component
    - State machine for each active component
  - Used to validate a design pattern template by simulating the flow of events and messages through the design pattern
• Sequence Diagram
  – *Illustrates interactions between the components*
  – *Sequence diagram covers a sample set of component interactions*
C&DH Common Features

- Command execution feature
  - Executing the command or distributing the command to the appropriate subsystem for execution

- Mode management feature
  - Maintaining and changing the current mode of the spacecraft (e.g. Normal Mode, Safe Mode)

- Collect engineering data feature
  - Receiving health and monitor data from other subsystems

- Collect payload data feature
  - Receiving payload data from the payload hardware

- Format telemetry feature
  - Formatting the engineering or payload data into a telemetry format

- Store telemetry feature
  - Storing the telemetry in memory

- Retrieve telemetry feature
  - Retrieving telemetry from memory

- Local fault detection feature
  - Performing various checks on local operations and devices to ensure they are functioning correctly

C&DH Common Features are features that are seen on a wide variety of spacecraft.
Deriving C&DH Patterns from DRE Patterns (cont.)

- **Format Telemetry Feature**
  - Specialized version
  - DRE Data Processing Feature
  - Starting Point

- **Store (or Retrieve) Telemetry Feature**
  - Specialized version
  - DRE Data Sharing Feature
  - Starting Point

- **Local Fault Detection Feature**
  - Specialized version
  - DRE Fault Detection Feature
  - Starting Point

**DRE Data Sharing Feature**
- **DRE Data Processing Feature**
  - Pipes and Filters
  - Master Slave
  - Single Protected Channel
  - Homogeneous Redundancy
  - Heterogeneous Redundancy
  - Triple Modular Redundancy
  - Monitor Actuator
  - Sanity Check
  - Strategy

**DRE Fault Detection Feature**
- **DRE Data Sharing Feature**
- **Two Phase Commit**
- **Compound Commit**
- **Long-Living Transaction**
- **Strategized Locking**
- **Scoped Locking**

**Watchdog**
Augment C&DH Design Patterns with C&DH specific patterns and customizations

- **Command execution and mode management features**
  - Centralized Control
  - Hierarchical Control
  - Distributed Control
  - State
  - Command Processor
  - Command Dispatcher

- **Collect engineering data feature**
  - Client Server
  - Multiple Client Multiple Server
  - Multitier Client Server
  - Publish Subscribe
  - Broadcast
  - Multicast

- **Collect payload data feature**
  - Client Server
  - Multiple Client Multiple Server
  - Multitier Client Server
  - Publish Subscribe
  - Broadcast
  - Multicast

- **Format telemetry feature**
  - Pipes and Filters
  - Master Slave
  - Strategy
  - Abstract Factory with Facade

- **Store telemetry feature**
  - Two Phase Commit
  - Compound Commit
  - Long-Living Transaction

- **Retrieve telemetry feature**
  - Two Phase Commit
  - Compound Commit
  - Long-Living Transaction
  - Strategized Locking
  - Scoped Locking

- **Architecture Structure**
  - Layers
  - 4 Layer Architecture
  - Whole-Part

- **Local fault detection feature**
  - Watchdog
  - Processor Restart Watchdog
  - Single Protected Channel
  - Homogeneous Redundancy
  - Heterogeneous Redundancy
  - Triple Modular Redundancy
  - Monitor Actuator
  - Sanity Check

- **Communication**
  - Asynchronous
  - Bidirectional Asynchronous
  - Asynchronous with Callback
  - Synchronous with Reply
  - Synchronous without Reply
  - Broker Forwarding
  - Broker Handle
  - Discovery
  - Publish Subscribe
  - Broadcast
  - Multicast
Customizing Executable UML Template Example

- Centralized Control Pattern
  - *One control component that provides overall control by conceptually executing a state machine*
- Customizations
  - Add C&DH generalized logic to Centralized_Controller statechart
  - Add Ground Commands since the controller must balance when to execute ground commands & when to respond to onboard events
  - Update sequence diagram options to include Ground_Command scenario
SNOE Case Study

- **Student Nitric Oxide Explorer (SNOE)**
  - *Real world, small satellite program from NASA*
  - *Mission involves using a spin stabilized spacecraft in a low earth orbit to measure thermospheric nitric oxide and its variability*
  - *The spacecraft instruments*
    - ultraviolet spectrometer (UVS)
    - auroral photometer (AP)
    - solar soft X-ray photometer (SXP)
    - microGPS Bit-Grabber Space Receiver
  - *All the science and engineering data collected is downlinked to the ground for processing*
  - *The ground station is responsible for attitude determination and monitoring long term health and safety for the spacecraft and instruments*
  - *All data and commands are formatted using Consultative Committee for Space Data Systems (CCSDS) standards*
  - *Thermal control is passive and is handled solely by the hardware*
  - *Limited hardware redundancy*
  - *One SC4A Single Board Spaceflight Computer*
    - Five I/O blocks on two daughter boards that handle interfacing to all subsystems
SNOE C&DH Pattern Selection

• Command execution and mode management features
  – Centralized Control
  – Hierarchical Control
  – Distributed Control
  – Command Dispatcher

• Collect engineering data feature
  – Client Server
  – Multiple Client Multiple Server
  – Multitier Client Server
  – Publish Subscribe
  – Broadcast
  – Multicast

• Collect payload data feature
  – Client Server
  – Multiple Client Multiple Server
  – Multitier Client Server
  – Publish Subscribe
  – Broadcast
  – Multicast

• Format telemetry feature
  – Pipes and Filters
  – Master Slave
  – Strategy
  – Abstract Factory with Facade

• Store telemetry feature
  – Two Phase Commit
  – Compound Commit

• Retrieve telemetry feature
  – Two Phase Commit
  – Compound Commit

• Architecture Structure
  – Layers
  – 4 Layer Architecture
  – Whole-Part

• Local fault detection feature
  – Watchdog
  – Processor Restart
  – Single Protected Channel
  – Homogeneous Redundancy
  – Heterogeneous Redundancy
  – Triple Modular Redundancy
  – Monitor Actuator
  – Sanity Check

• Communication
  – Asynchronous
  – Bidirectional Asynchronous
  – Asynchronous with Callback
  – Synchronous with Reply
  – Synchronous without Reply
  – Publish Subscribe
  – Broadcast
  – Multicast
SNOE C&DH Software Architecture – Static View

*message queues not depicted
SNOE C&DH Software Architecture – Static View

4 Layers Architecture Pattern

- Control Layer
  - Centralized_Controller
  - RT_Command_Queue
  - ATT_Command_Queue

- Data_Handling_Layer
  - Payload_Data_Client
  - Eng_Data_Client

- Telemetry_Layer
  - Telemetry_Format_Server
  - Abstract_Transformation
  - Time_Tag_Transformation
  - Packetize_Transformation
  - Telemetry_Packet

- Device_Interface_Layer
  - Receiver_Input_Component
  - microGPS_IO_Component
  - Transmitter_Output_Component
  - SSP_IO_Component
  - SSR_IO_Component
  - HCL_IO_Component
  - PCU_IO_Component
  - AP_IO_Component
  - VGS_IO_Component
  - Torque_Rod_Output_Component

- Data_Sensor_Layer
  - Eng_Data_Server
  - Payload_Data_Server
  - Eng_Data
  - AP_Data
  - UGS_Data
  - SXP_Data
  - microGPS_Data

- Fault_Detection_Layer
  - Telem_Validation
  - AP_Watchdog
  - UGS_Watchdog
  - Receiver_Watchdog
  - Magnometer_Watchdog
  - HCL_Watchdog
  - Processor_Watchdog
  - PCU_Watchdog
  - microGPS_Watchdog
  - SXP_Watchdog
  - Transmitter_Watchdog
  - Torque_Rod_Watchdog

Communication

Abstract_Os

Abstract_Pw
SNOE C&DH Software Architecture – Static View

Client Server Pattern (collect eng data)
Client Server Pattern (collect payload data)
SNOE C&DH Software Architecture – Static View

Pipes & Filters Pattern
SNOE C&DH Software Architecture – Static View

Single Protected Channel Pattern
SNOE C&DH Software Architecture (cont.)

- Design patterns after instantiation provide the foundation for the C&DH software architecture

- To have a complete C&DH software architecture the design patterns need to be:
  - *Interconnected with the rest of the software architecture*
  - *Customized to meet the application specific requirements*
    - Adding application specific processing logic
    - Adding application specific control logic to state machines
    - Updating sequence diagrams to reflect the application specific interactions
Conclusions and Future Work

• Conclusions
  – **Presented an approach to building C&DH subsystem from software architectural design patterns**
    • Based on DRE software architecture patterns
    • Leverages the UML software modeling language
  – **Using this approach C&DH software will lead to**
    • Better quality C&DH software architectures
    • Reduced number of onboard anomalies related to software

• Future Work
  – **Apply patterns to additional case studies**
  – **Expand research to include other FSW subsystems (e.g. Fault Management Subsystem, Attitude Control Subsystem)**
  – **Augment executable design pattern templates with software performance characteristics to facilitate software performance validation**
  – **Look for areas to automated the application of the executable design pattern templates**
  – **Expand research to other DRE domains**
References

[14] Laboratory For Atmospheric and Space Physics, “STUDENT NITRIC OXIDE EXPLORER HOMEPAGE.”
Designing Command and Data Handling Subsystems from Software Architectural Design Patterns

Julie Fant\textsuperscript{1}, Hassan Gomaa\textsuperscript{2}, and Robert Pettit\textsuperscript{1}

The Aerospace Corporation\textsuperscript{1} and George Mason University\textsuperscript{2}

Computer and Software Division
The Aerospace Corporation
December 7, 2010
Julie.S.Fant@aero.org
571-307-3828