AEROSPACE
Assuring Space Mission Success

Architectural Design Patterns for Flight Software

Julie Fant[1], Hassan Gomaa[2], and Robert Pettit[1]
The Aerospace Corporation[1] and George Mason University[2]

Computer and Software Division
The Aerospace Corporation
March 2011

GEORGE MASON UNIVERSITY

Outline

- **Introduction**
- **Related Works**
- **Research Approach**
  - ***Selecting Patterns for FSW***
  - ***Creating Design Pattern Templates for FSW***
  - ***Capturing Software Performance in Design Pattern Templates***
- **Real world case study**
- **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
  - *Platform and domain independent*
  - *Can be applied at several different layers of abstraction*

- Taking advantage of design patterns is particularly import for the flight software (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 architecting [2]

Related Works

- Several notable approaches and patterns for building real time software architectures from design patterns
  - *Only provide high level guidance applying design patterns*
  - *Do not take the additional step of providing domain specific executable design pattern templates to make applying design patterns*

- Less research in applying design patterns to the FSW domain
  - *Herrmann and Schöning use abstract factory and façade design patterns for telemetry processing*
    - Do not address how design patterns can be used for other FSW features
  - *Several reference architectures for FSW that can be used as a starting point for FSW*
    - Not design pattern based therefore they do not guarantee that the benefits of design patterns will be leveraged in the architectures produced using them

- Mission Data System (MDS) project provides a system level control architecture, framework, and systems engineering methodology for developing state-based models for planning and execution.
  - *Our research complements and supports this work*

Research Approach

- ***Systematic approach for designing common functionality in FSW architectures from software architectural design patterns***
  - Select Patterns for FSW
  - Create Design Pattern Templates for FSW
  - Capture Software Performance in Design Pattern Templates
  - Build FSW from design patterns

Emphasis on common features that are seen on a wide variety of spacecraft

Selecting Patterns for FSW

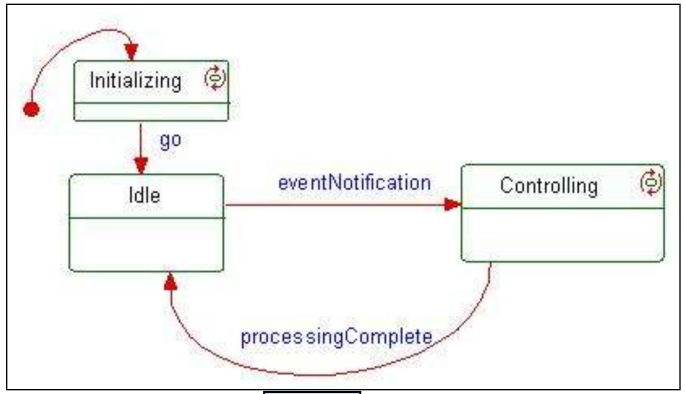
- Select existing design patterns from the DRE domain that support FSW functionality
  - *This can be accomplished because FSW is a type of DRE software*
- Emphasis on common features across the FSW domain
  - *Command execution*
  - *Uplink/downlink telemetry*
  - *Others*
- Example : <u>Command Execution</u> involves determining the order in which spacecraft commands are executed
  - *Example patterns that can be used to support this feature*
    - Centralized control
      - *Single control component that conceptually executes a state machine*
      - *Benefits: control logic contained in single component therefore easier to maintain and understand*
      - *Well suited for small spacecraft*
    - Hierarchical control
      - *Multiple control components that control some part of the system by conceptually executing a state machine*
      - *Single coordinator that orchestrates overall control by determining next job and sending it to controller for executing*
      - *Benefits: overall control handle by single component, but several controllers to execute the work to avoid bottlenecks*
      - *Suited for larger spacecraft*

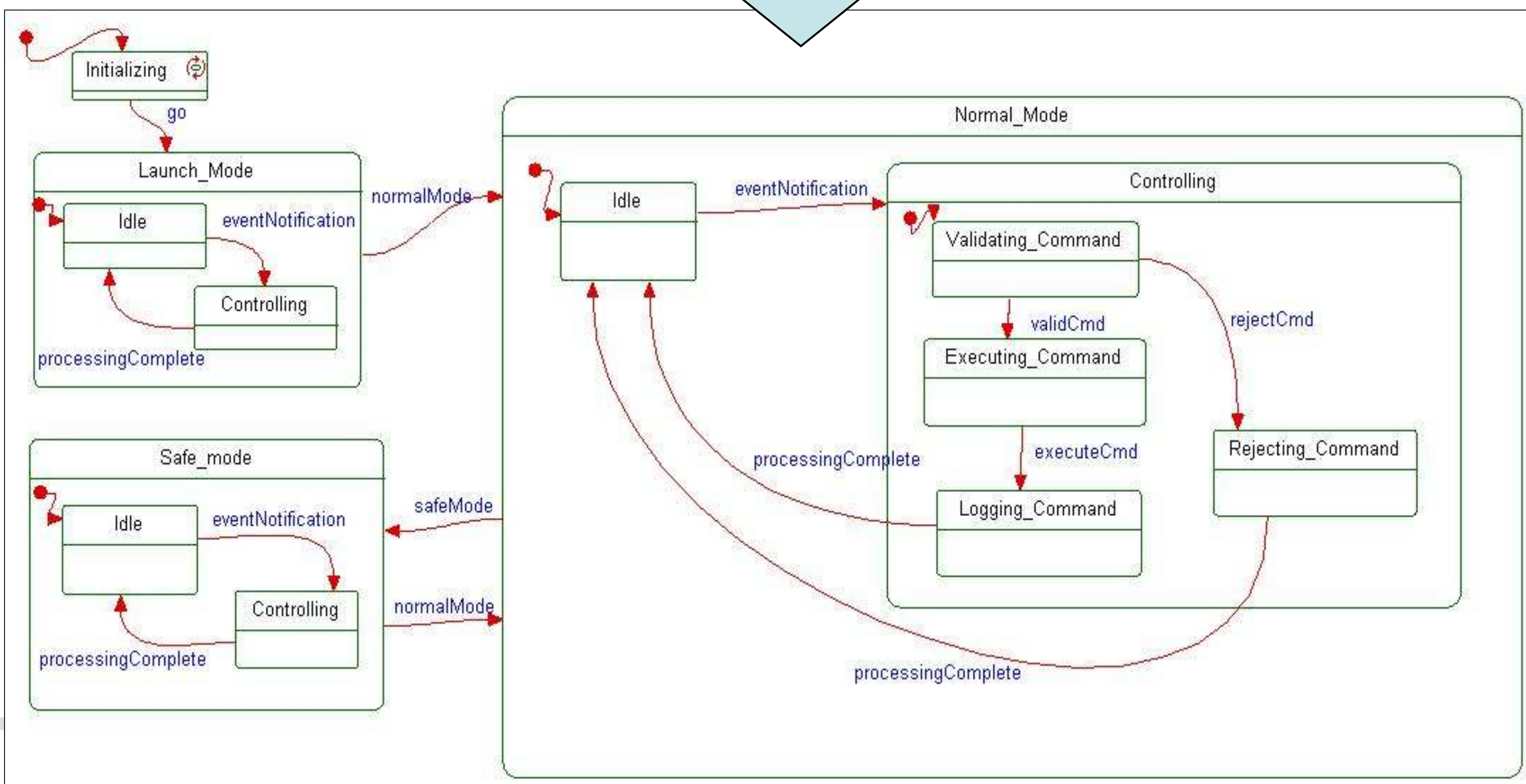
Creating Design Pattern Templates for FSW

- Create executable design pattern templates for the FSW domain
  - *Makes the design patterns more directly applicable to FSW architectures*
  - *Provide structure for design patterns*
  - *Save time when instantiating the design patterns*

- Executable design pattern templates
  - *Captured using the UML*
    - Both static and dynamic architectural views
  - *State machines used to capture the internal behavior of each concurrent component in the design pattern*
    - Executed using Harl's executable statechart semantics

Creating Design Pattern Templates for FSW Example





Capturing Software Performance in Design Pattern Templates

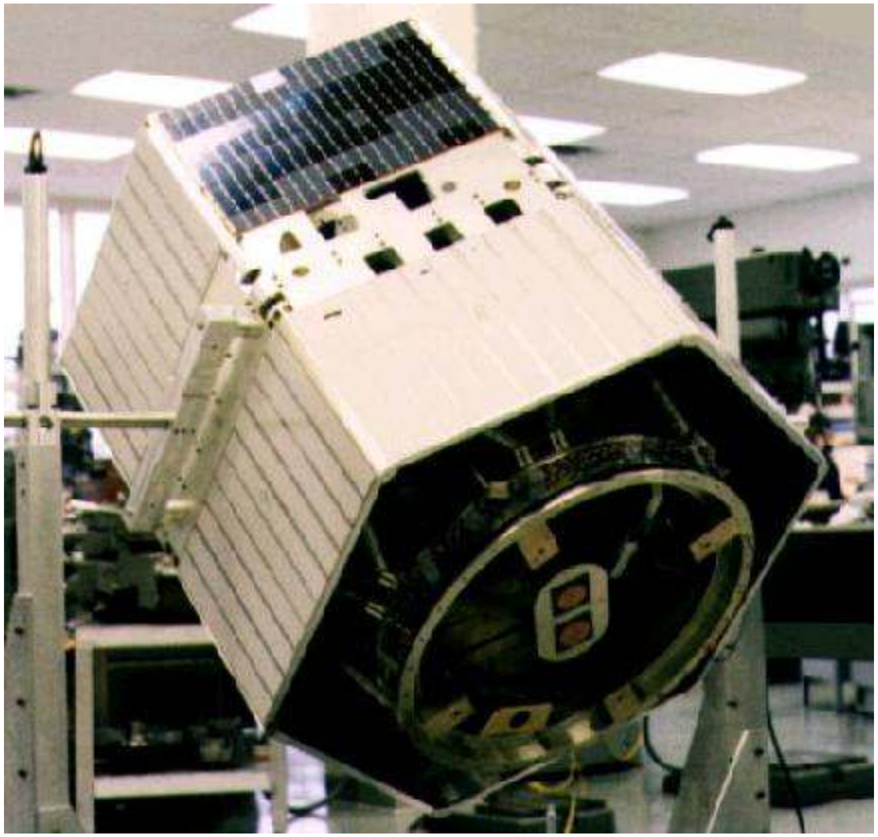
- Platform independent software performance information captured with the MARTE Profile

- MARTE annotations are used in the sequence diagrams
  - *MARTE stereotypes used depending on the type of performance analysis*
  - *For example, if the sequence diagram lends itself to analyzing response time*
    - «GaWorkloadEvent» stereotype is used to denote an event that triggers the scenario on the sequence diagram.
    - «PaStep» stereotype is used on any step that is involved in the scenario

- Contain platform independent software performance estimates
  - *Captured in the tags of the MARTE stereotypes*
  - *Platform independent estimates are captured using comparative parameters*
    - Example: 2t where t represents a platform specific multiplier relative to a benchmark
  - *When the design pattern templates are applied to a specific FSW architecture, these parameters will be substituted for the platform specific values*

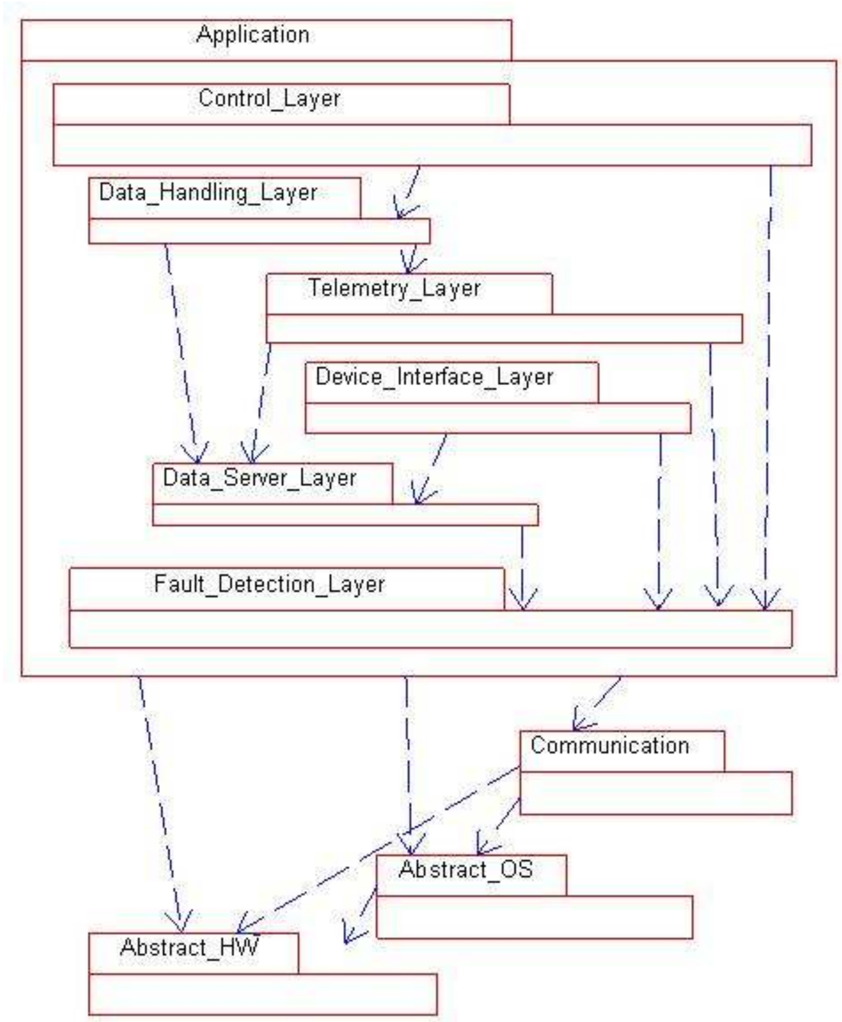
SNOE Command and Data Handling (C&DH) Case Study

- **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)
    - mircoGPS 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



Building SNOE C&DH from Design Patterns

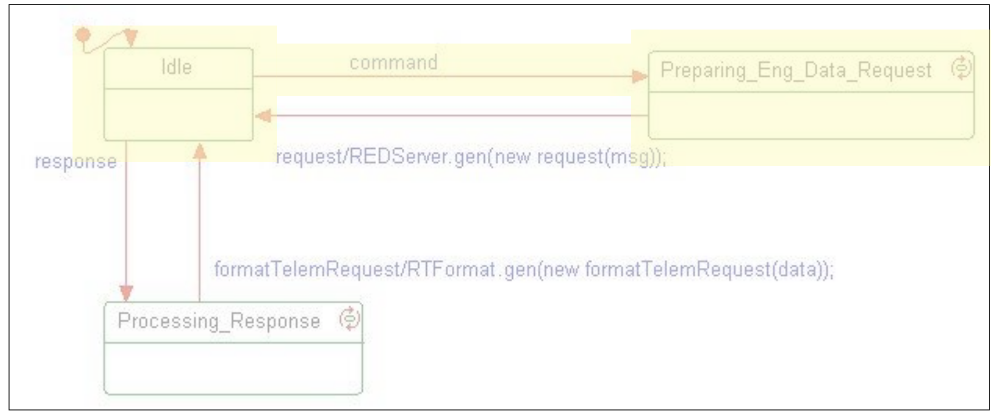
- Selecting design patterns for SNOE
  - *SNOE's C&DH subsystem uses 11 patterns*
  - *Example: Command execution*
    - SNOE controls a relatively small number of hardware devices
    - Payload instruments require minimal commanding from FSW
    - Centralized Control good match!
- Executable templates are instantiated for SNOE
  - *Example: Modified 5 Layer Pattern for FSW and Layers Pattern*
- SNOE specific information is added to the templates
- Finally, interconnect design pattern templates with the rest of the architecture
  - *Resulting software architecture can then be validated using executable statechart semantics*



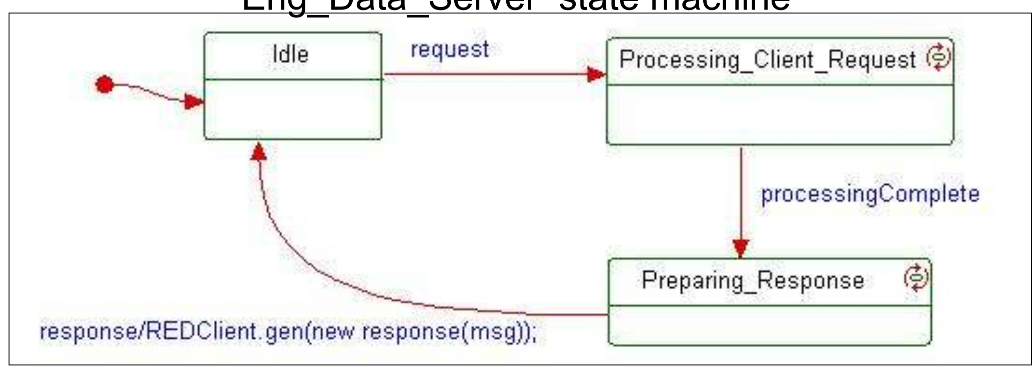
SNOE Functional Validation

- Example: Collect engineering data scenario
  - *Centralized_Controller receives, validates, and determines response to a ground command to collect the spacecraft engineering data*
  - *Centralized_Controller sends this command to the Eng_Data_Client to execute*
  - *When the Eng_Data_Client receives the command it moves into the Preparing_Eng_Data_Request state*
    - Prepares a request for the Eng_Data_Server to get the current engineering data

Eng_Data_Client state machine



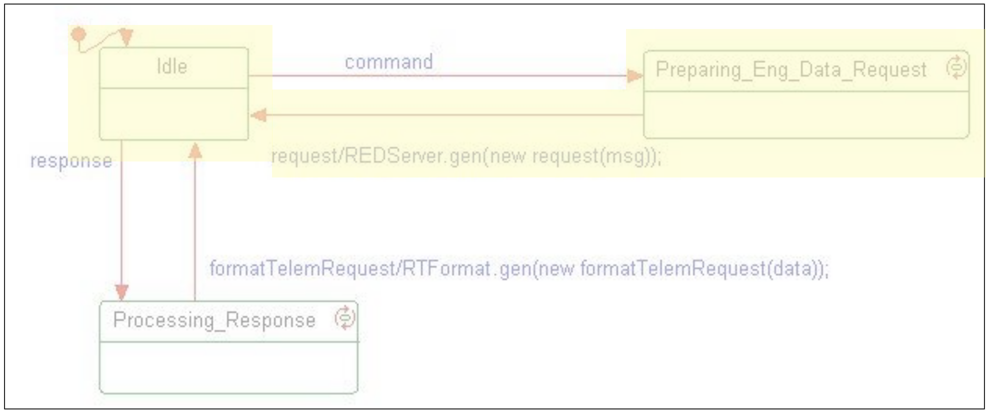
Eng_Data_Server state machine



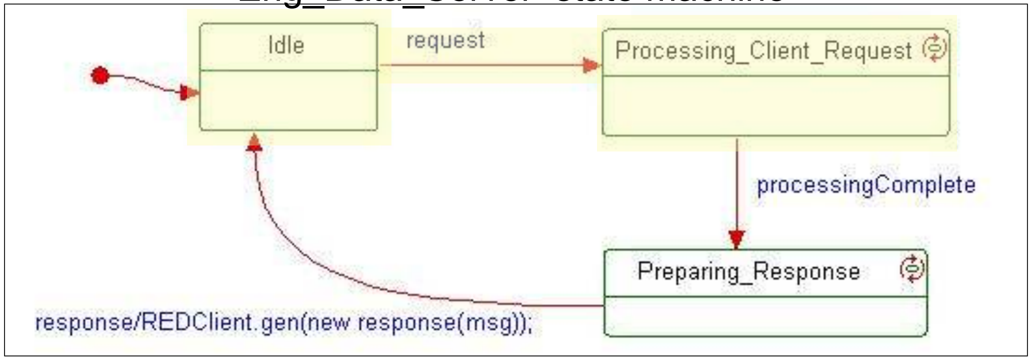
SNOE Functional Validation (cont)

- Example: Collect engineering data scenario (cont)
  - *Eng_Data_Client then sends the new request message to the Eng_Data_Server through its required port called REDServer*
    - Eng_Data_Client transitions back to the Idle state
    - Eng_Data_Server transitions into Processing_Client_Request state
  - *Eng_Data_Server processes the request*
    - Transitions to the Preparing_Response state to format a response message

Eng_Data_Client state machine



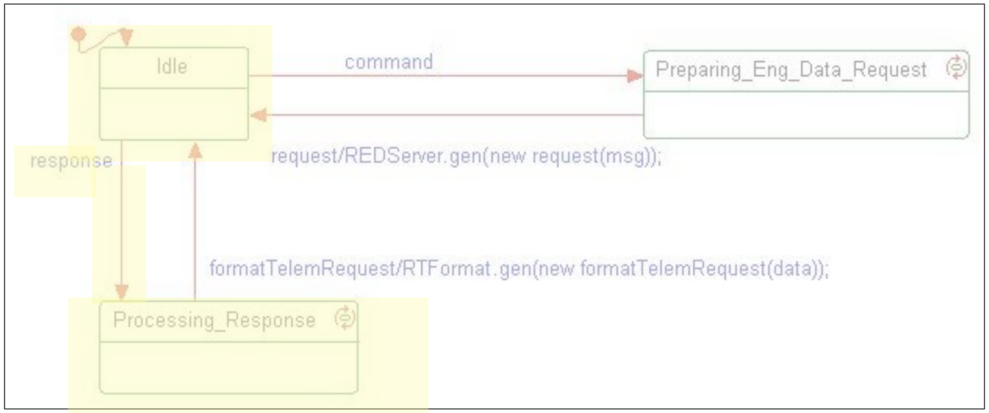
Eng_Data_Server state machine



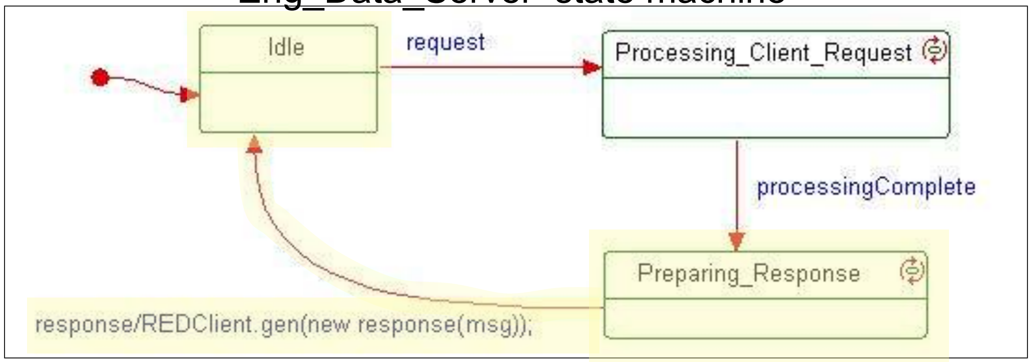
SNOE Functional Validation (cont)

- Example: Collect engineering data scenario (cont)
  - *Eng_Data_Server sends the response to the Eng_Data_Client through its required ported called, REDClient*
    - Eng_Data_Server transitions back to the Idle state to wait for the next request
  - *Eng_Data_Client receives the response message and transitions into Processing Response State*
    - Processes the response and performs checks on the data

Eng_Data_Client state machine



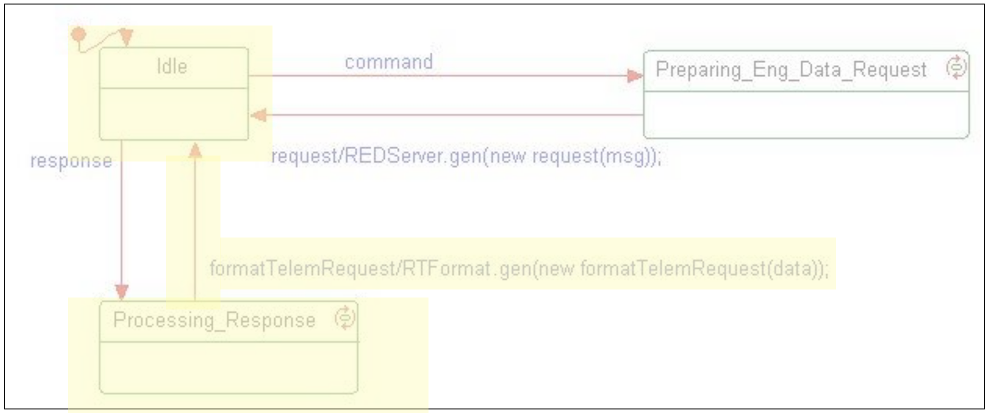
Eng_Data_Server state machine



SNOE Functional Validation (cont)

- Example: Collect engineering data scenario (cont)
  - *When processing is complete Eng_Data_Client then sends the data to the Telemetry_Formatter to format that data into telemetry packets for transmission through the required port call RTFormat*
    - Eng_Data_Client returns to the Idle state
- Process is repeated for other scenarios

Eng_Data_Client state machine



The Collect Engineering Data scenario executed as expected therefore it is validated!

Conclusions and Future Work

- Conclusions
  - *Presented an approach to building FSW from software architectural design patterns*
    - Based on DRE software architecture patterns
    - Leverages the UML software modeling language
  - *Using this approach will lead to*
    - Better quality software architectures
    - Reduced number of onboard anomalies related to software design flaws
- Future Work
  - *Expand case study to include performance validation*
  - *Apply patterns to additional case studies*
  - *Look for ways to address feature variability in the FSW domain*
  - *Look for areas to automated the application of the executable design pattern templates*
  - *Expand research to other DRE domains*
  - *Explore state machine based code generators for rapid prototyping and software performance benchmarking*