



---

# **HAWC Science Instrument Software Subsystem**

## **Preliminary Design Review**

**NASA/GSFC  
Greenbelt, MD  
October 2, 1998**



# Agenda



- 
- **Context**
    - Participants
    - Observation Life Cycle
    - Software Conceptual Model
    - Requirements
  - **HAWC Specific Elements**
    - Hardware Control and Interfaces
    - Data Pipeline
  - **Analysis and Design Methodology**
    - Design Philosophy
    - Design Approach - Details
  - **Management Plan**
    - Design Schedule and Milestones
    - Test/Verification Plan
    - Risk Analysis and Mitigation



---

# Context

Participants  
Observation Life Cycle  
Software Conceptual Model  
Requirements



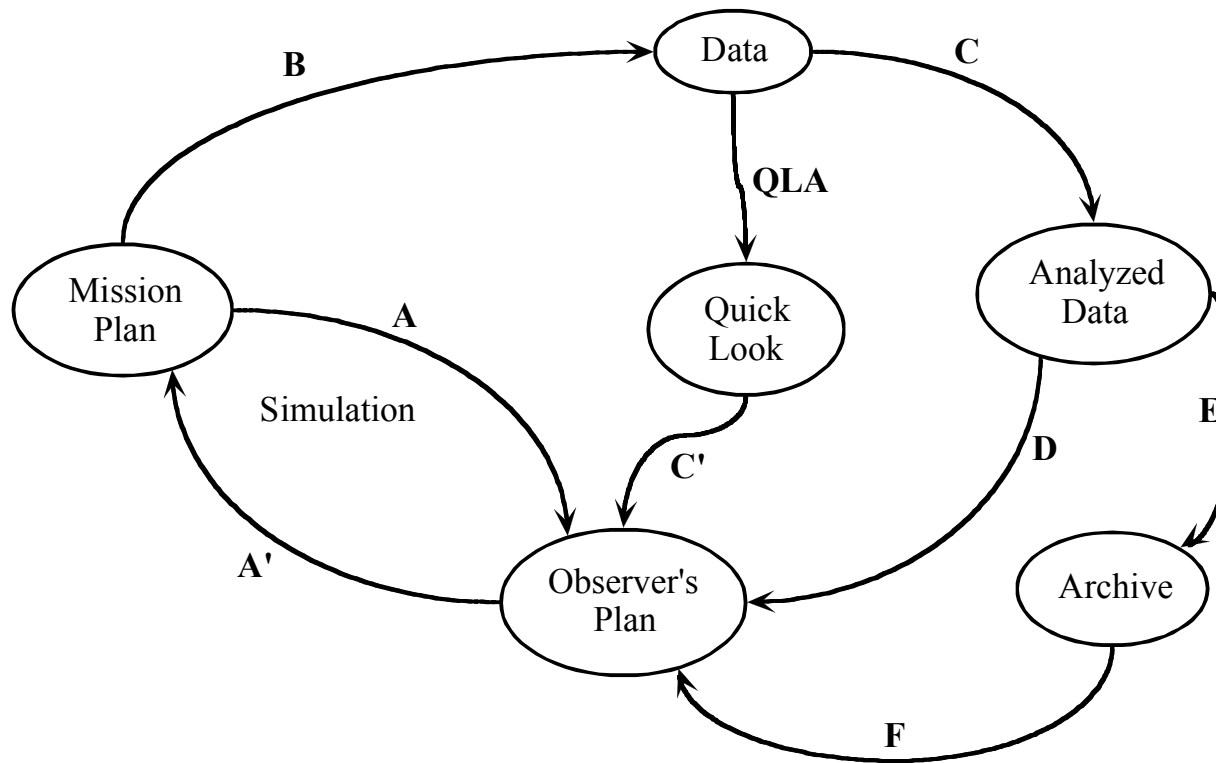
# Participants



- 
- University of Chicago, Yerkes Observatory
  - NASA/Goddard Space Flight Center
  - Rochester Institute of Technology



# Observation Life Cycle



**A** Convert flight, telescope and instrument parameters into a sequence of image locations and exposures.

**A'** Convert flight leg and sky position descriptions into flight, telescope and instrument command script. Capture Quick Look and Pipeline specifications.

**B** Execute mission plan and capture data, including housekeeping and logs.

**C** Perform data analysis using pipeline specification.

**D** Presentation of analyzed data

**E** Archiving of raw data, analyzed data, pipeline algorithms, housekeeping, observation log, analysis log.

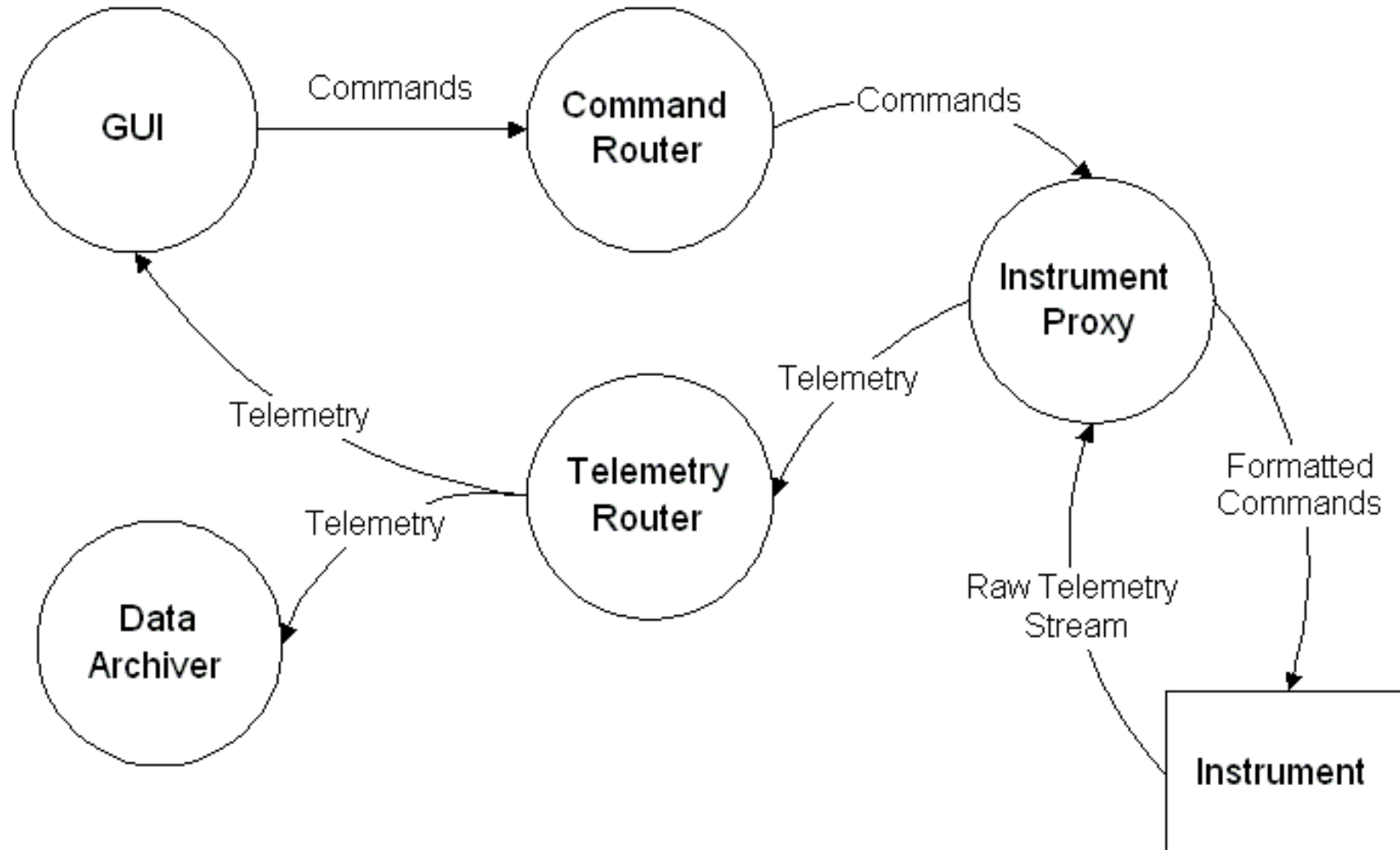
**F** Retrieval of archived information, including other sensors, flight plans, pipeline specifications, etc.

**QLA** Quick look analysis and presentation using planning baseline and scripts..

**C'**

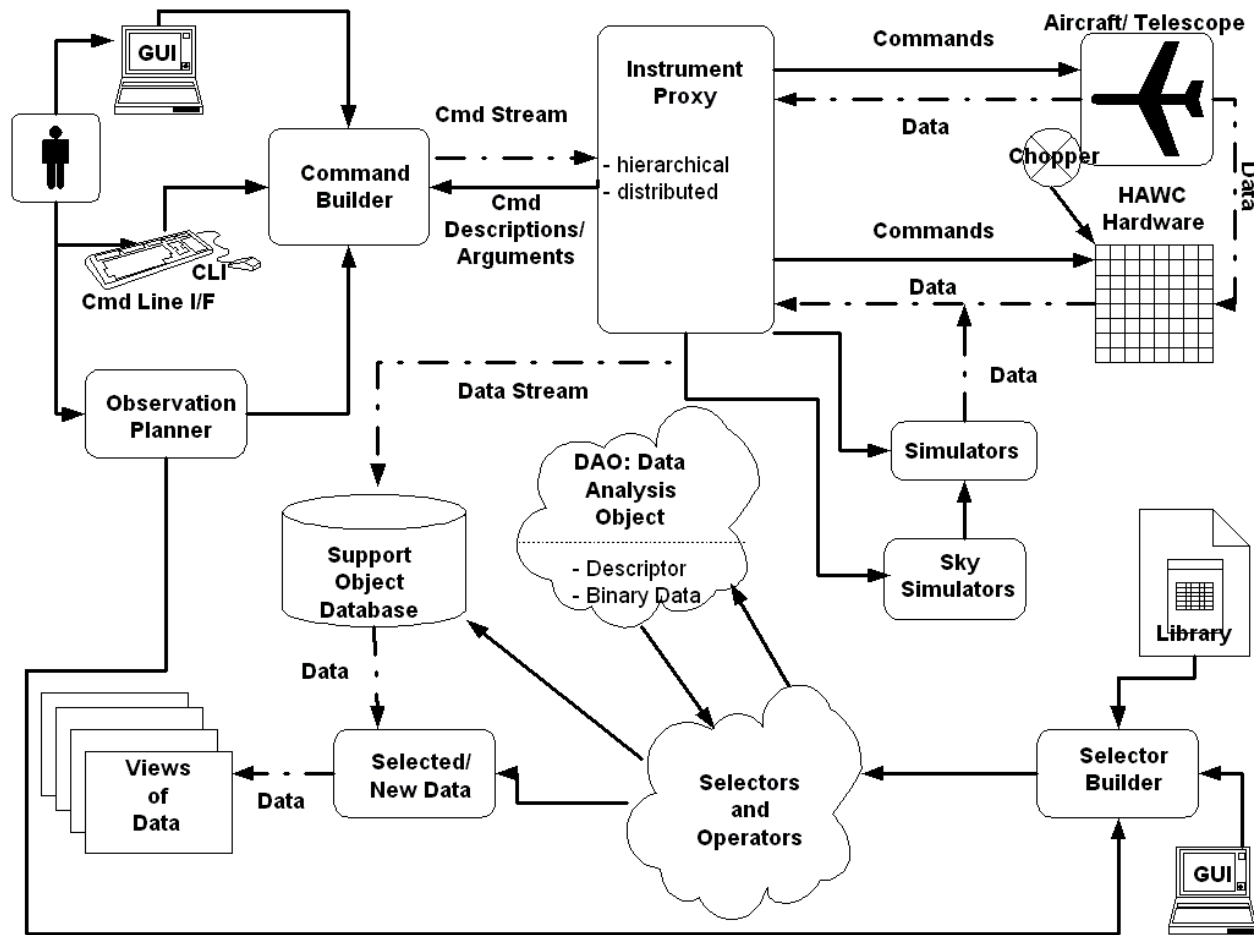


# Draft Elements of System Architecture

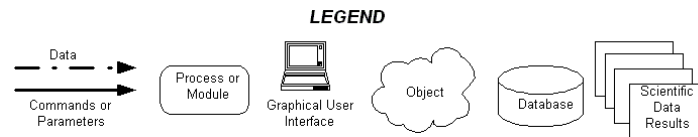




# Software Conceptual Model



HAWC Software Subsystem PDR





# User Requirements (Level 1)

---



- (1) Support full instrument operations in flight and on the ground
- (2) Maintainable by SSMOC staff
- (3) Easy to use by observer/user
- (4) Able to produce scientific quality data during the first year of operation
- (5) Support system integration and testing



# Design Requirements (Level 2)

---



- (1) “Turnkey” instrument operation (i.e. Comes up when switched on) - L1R3
- (2) User interface must pass defined usability tests - L1R3
- (3) Monitoring of all important instrument characteristics - L1R2,L1R3,L1R4
- (4) No false dependencies - L1R1,L1R4
- (5) Self documenting data throughout - L1R2,L1R3,L1R4
- (6) Platform independence - L1R2
- (7) Distributed architecture: Instrument controllable from any station on LAN(?) - L1R3
- (8) Preview of image to verify quality - L1R4



# Design Requirements (Level 2, cont.)

---



- (9) Post observation processing of data available within one week - L1R4
- (10) Ability to carry out an observation plan submitted by remote Investigator - L1R3
- (11) Instrument Software contains a comprehensive set of tools to support observations - L1R2,L1R3
- (12) Investigator can carry out an observation with (TBD) 2-4 hours of training - L1R3
- (13) All instrument states are accessible through the Investigator's interface - L1R3, L1R4
- (14) Software can function under sub-optimal instrument conditions - L1R1



# Design Requirements (Level 2, cont.)

---



- (15) Data collection and distribution will include all contextual information (house keeping, instrument and observatory environment, etc.) - L1R3,L1R4
- (16) Instrument controllable remotely given sufficient bandwidth - L1R3
- (17) Software contains a comprehensive set of tools to support instrument performance diagnostics - L1R2,L1R3
- (18) All instrument anomalies are reported to Investigator's interface and logged in a permanent archive - L1R3, L1R4
- (19) Software System must pass defined acceptance tests before instrument delivery – L1R5



---

# HAWC Specific Elements

Hardware Monitor and Control  
Interfaces  
Data Pipeline



# Hardware Monitor and Control

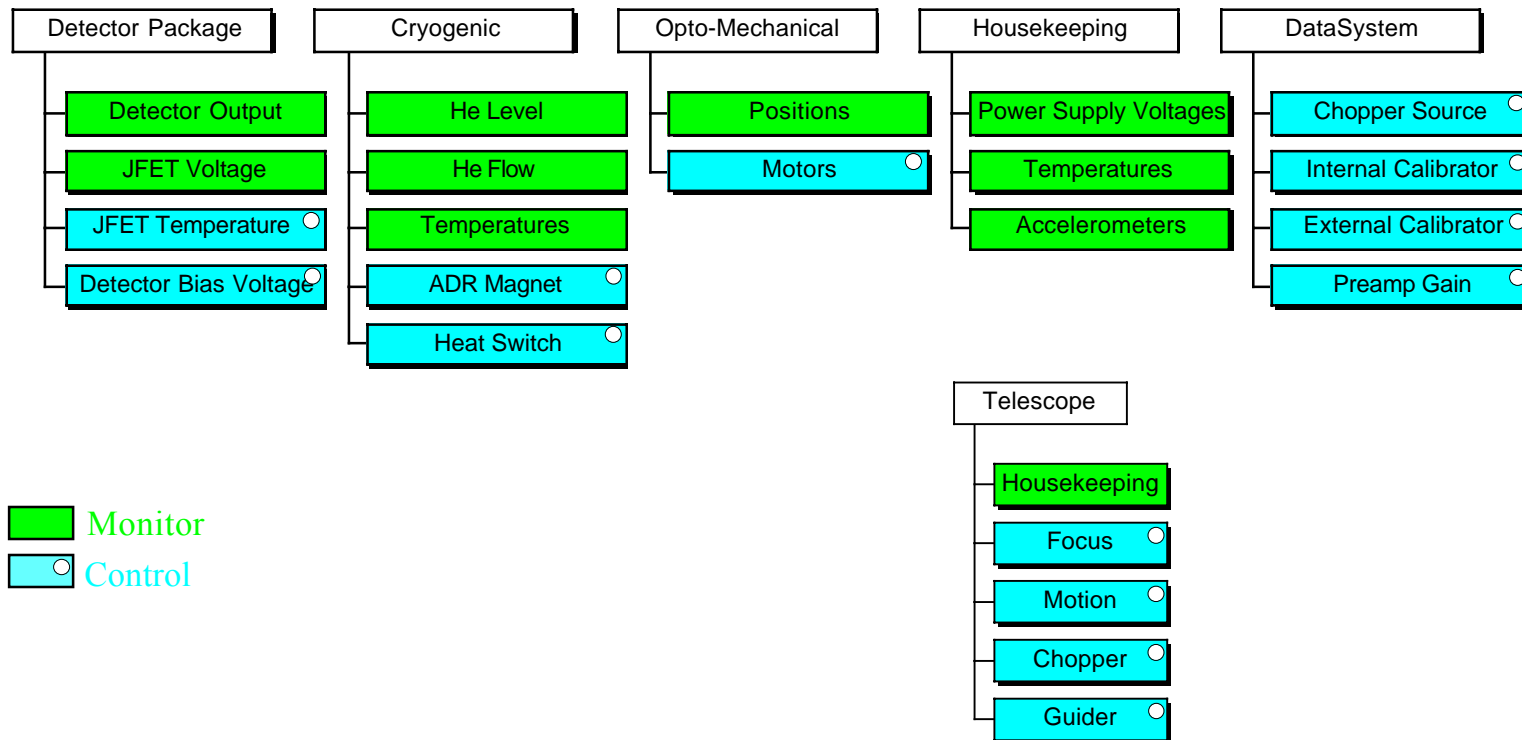
---



- Demodulate IR data stream
- Display data in various formats
  - Real Time (strip charts, images, etc.)
  - Partially reduced (Quick Look)
- Control various hardware systems
  - Instrument
  - Telescope
  - Test Equipment
- Monitor parameters
- Store data and housekeeping

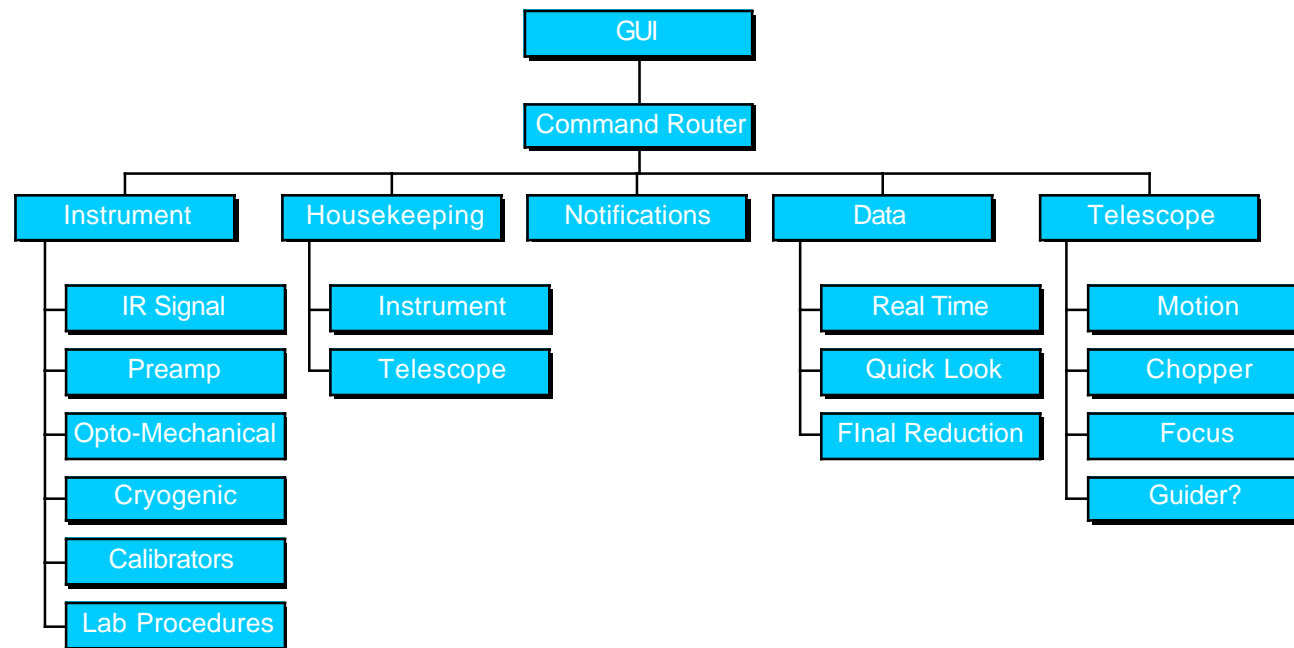


# Hardware Monitor and Control





# Functional Diagram





# Hardware Interfaces



---

## Data System

- Preamp gain
- Internal calibration source control
- Chopper position
- External calibrator
- Boresight box

## Detector Package

- Read detector output
- Monitor JFET supply voltage
- Monitor and control Bias voltage
- Monitor JFET temperature
- Control JFET temperature

## Cryogenic subsystem

- He level sense
- He flow sensor
- ADR magnet control
- Heat switch control
- Temperature sensors

## Opto-mechanical subsystem

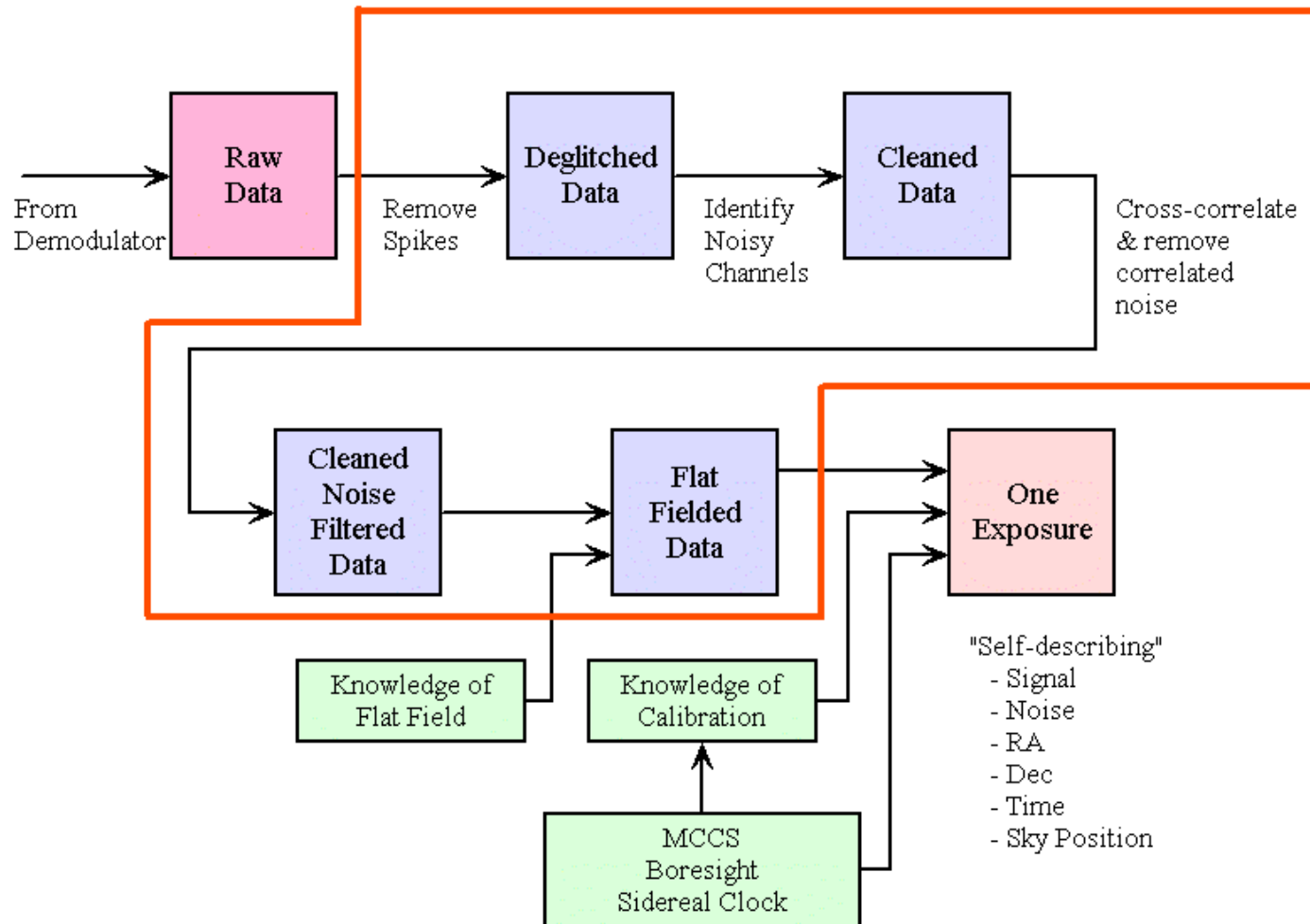
- Motor/actuator control
- Motor/actuator positional information

## Health/Safety/Environment

- Supply voltage monitors
- Temperature sensors
  - Dewar
  - External
- Accelerometers



# Data Pipeline Conceptual Model





# Data Pipelining



- 
- Need automated, script-driven data taking
    - Which means simply:
      - Go to object
      - Take a “single exposure”
      - Check the result for acceptability using quick-look
      - If no, repeat; if yes, start observing script
      - Go to next filter or next source
    - Observers will be encouraged to have their observing plan scripted, to allow an automated data pipeline



# Advantages of Scripting



- 
- Organizes and controls the data flow
    - “writes the log book”
  - Provides the instruction set for reduction
    - Script is the output from planning & simulation
  - A natural way to deal with interrupts
    - Such as turbulence and loss of track
    - Just pick up where you left off...
  - Frees the observer to concentrate on other things



# “A Single Exposure”



- 
- To produce a “single exposure” we need the data to be:
    - Cleaned:
      - Remove noise spikes,
      - Remove noisy channels,
      - Assess and remove any correlated sky noise
    - Flat-fielded
    - Calibrated
    - Positioned ( $\alpha, \delta$  of source and sky positions)



# Assembling Images from Multiple Exposures and Maps

---



- Compact source
  - Beamswitch, assume no flux in reference beam
- Extended source
  - Larger than the chopper throw
  - Demands iterative spatial deconvolution
- In either case
  - Stack pixels from single exposures



# Stacking the Exposures



- 
- Need to account for field rotation, particularly for longer integrations
  - Typically resample onto uniform grid (i.e. Image)
    - and rotate to North-up East-left
  - Output image as FITS file with standard header
  - Archive the results



# Knowing the Flatfield



- 
- The KAO 60 channel system
    - observed bright extended sources (e.g. M42, M17)
    - Method: “same sky position with different detector”
  - Archiving with Facility Class instrument:
    - Knowledge of (say) M42 continually improves
    - Long-term goal:
      - a single snapshot will both calibrate and flatfield, HAWC should have a “flatfielder/calibrator” which could be observed in every flight



---

# Analysis and Design Methodology

Design Philosophy  
Design Approach - Details



# Design Philosophy

---



- Facility instrument
- Object Oriented Analysis and Design
  - Factored architecture
  - Easier maintenance; incorporate changes quickly
  - Extensible to new technology
  - Re-useable elements (interface, graphical visualization tools, data selection tools)
  - Increased opportunity to integrate COTS elements
  - Translates to reduced lifecycle costs for system



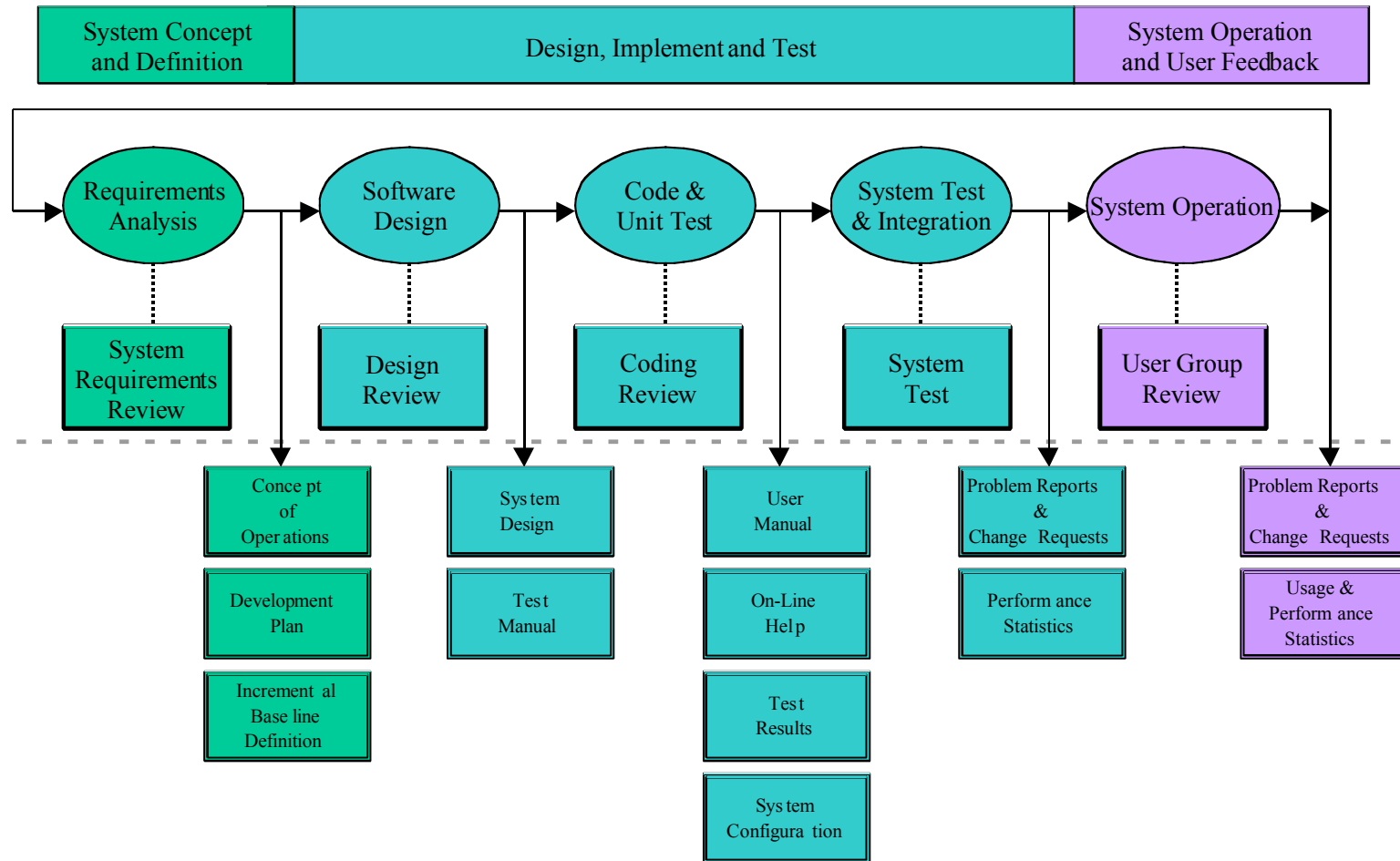
## Design Philosophy (cont.)



- 
- Distributed Architecture
  - Established Design Techniques and Tools (UML)
  - Iterative Design and Implementation (Continuous Improvement model)
    - Short intervals between iterations' deliveries
    - Risk oriented scheduling – assign high risk elements to early iterations
  - User Interfaces guided by Usability Engineering Practices
  - Provide Object level adapters and wrappers for legacy level procedural



# Software Life Cycle





# Design Elements and Modeling Techniques

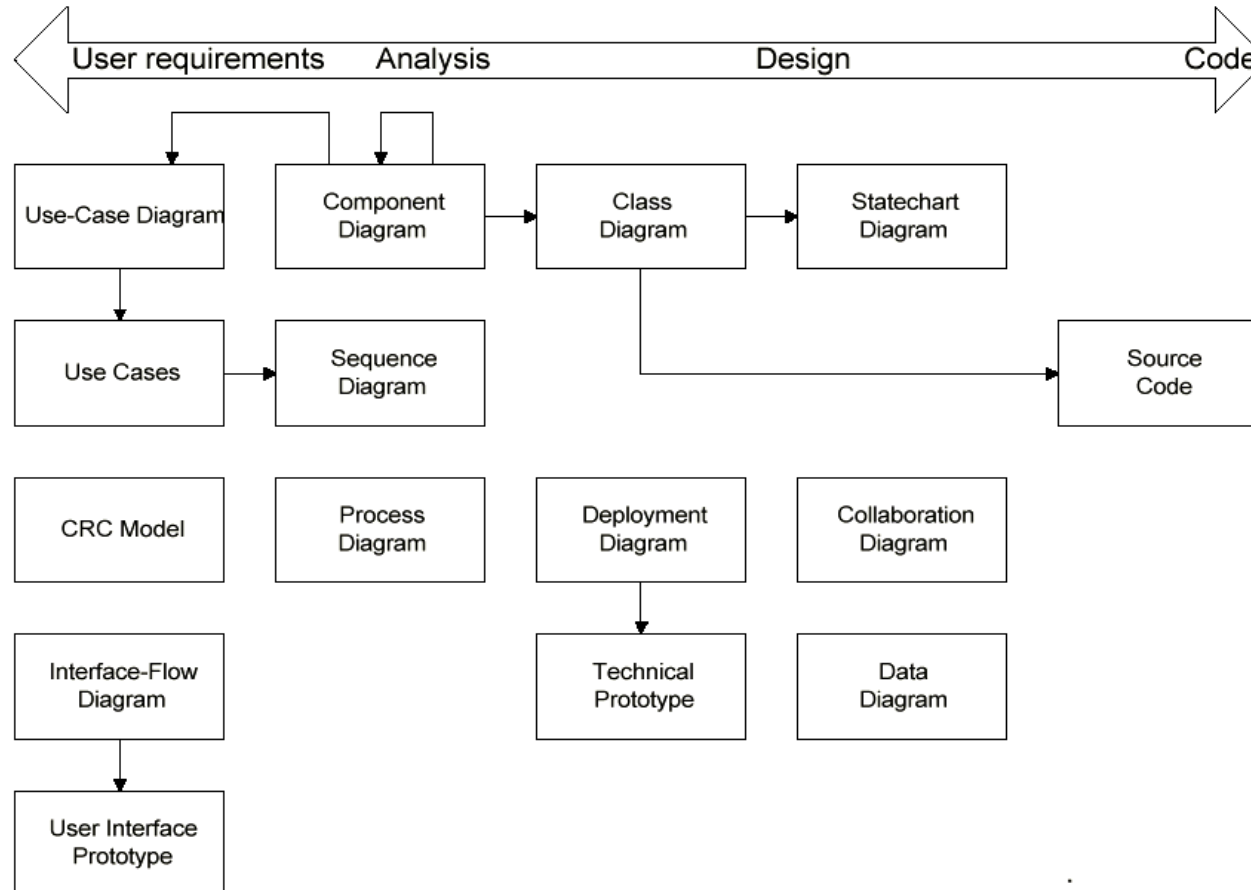
---



<b>Use Cases</b>	<b>Model of the dialogue between actors (scientist, technician, and engineers) and the system.</b>
<b>Use Case Scenarios</b>	<b>Instance of a use case – one path through the flow of events for the use case.</b>
<b>Class Responsibility and Collaborator cards</b>	<b>Software design technique for further refining the system requirement.</b>
<b>User Interface Prototypes</b>	<b>Means of describing the user interface and validating initial assumptions.</b>
<b>Class Diagrams</b>	<b>Show the classes of the system and their interrelationships (including inheritance, aggregation, and associations).</b>
<b>Sequence Diagrams</b>	<b>Diagram depicting object interactions arranged in time sequence.</b>
<b>Class/Collaboration Diagrams</b>	<b>Diagram representing object interactions.</b>
<b>Design Patterns</b>	<b>Application of proven design metaphors.</b>



# How the Modeling Techniques Fit Together





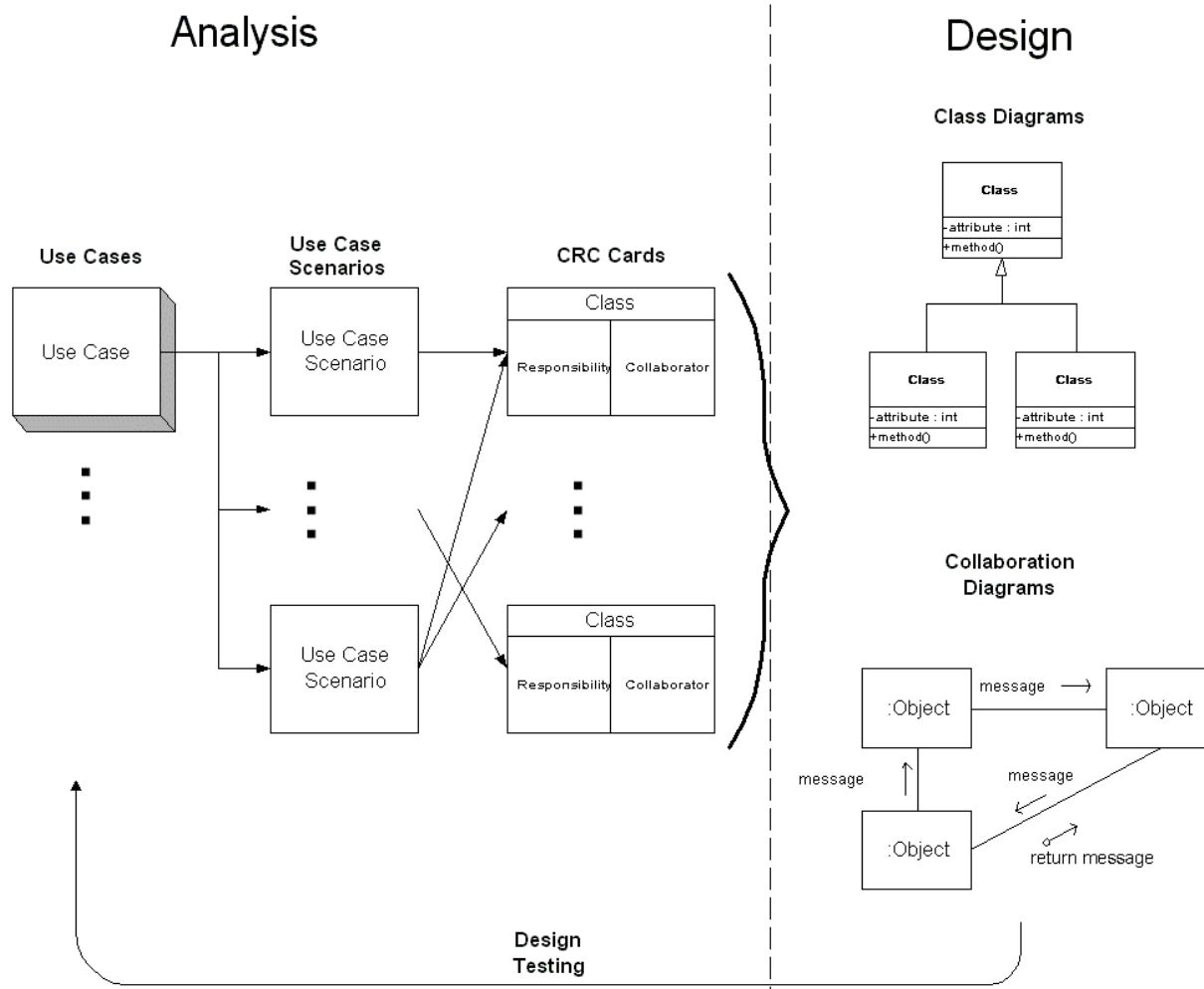
---

# Analysis and Design Methodology

## Design Approach - Details



# Initial Analysis and Design Steps





# Use Cases and Use Case Scenarios

---



- At highest level are system (not software) based
- Capture user expertise, extending requirements
- Provide test framework for design
- Define integration with observatory system
- Provide framework for user manual and maintenance procedures
- Drilling down to lower levels begin to define system elements and capabilities



# Use Case Examples



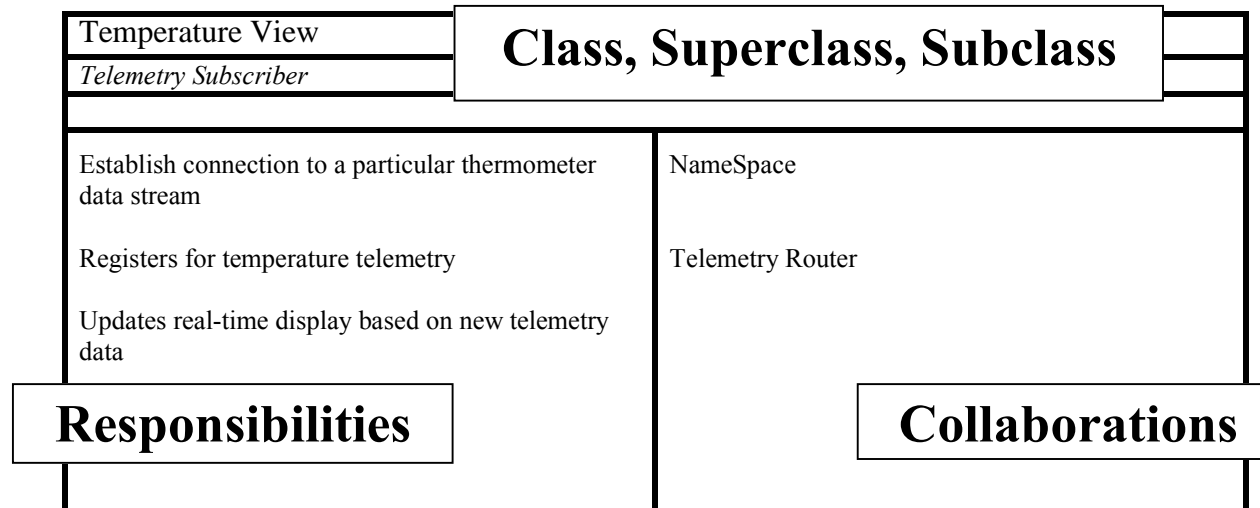
- 
- UC-1: Precommission: Scientist and Engineer build and deliver an integrated and tested facility instrument
    - UC-1.1: Engineers create and deliver subsystem
      - UC-1.1.1: Engineer tests Subsystem Command Interface
  - UC-2: Commission: HAWC Team and Observatory Staff perform initial checkout of in-flight performance of instrument and integrates instrument with the Observatory and System
    - UC-2.4: Technician installs Software System on Plane
      - UC-2.4.1: Technician performs Checkout on Plane
  - UC-3: Routine: Observers, supported by Observatory personnel, propose, plan, execute, and evaluate a scientific program using HAWC
    - UC-3.3: Technician checks out system in air
      - UC-3.3.1: Support Instrument Operator performs system/instrument Cold Start
  - UC-4: Maintenance: Observatory support personnel maintain and extend the HAWC instrument hardware and software.
    - UC-4.2: Technician installs upgrades
      - UC-4.2.1: Astronomer submits a new algorithm module to system library



# Class Responsibility and Collaborator Cards



- Identify classes - person, place, thing, event, concept, screen or report.
- Assign responsibilities - things that it knows and does, its attributes and methods.
- Identify collaborations - other classes it works with to fulfill responsibilities.





# Example Scenario

---



- Engineer Tests ADR Cycle Command Procedure
  - Class-Responsibility-Collaborator (CRC) cards
  - Class Hierarchy
  - Collaboration Diagram



# Use Case Context

---



- UC-1: Precommission
  - UC-1.1: Engineers Create and Deliver Subsystem
    - UC-1.1.7: Engineers Check-Out Subsystem
      - UC-1.1.7.1: Engineers Check-Out ADR Subsystem
        - » **UC-1.1.7.1.1 Engineer Tests ADR Cycle Command Procedure**



# Use Case: Engineer Tests ADR Cycle



<i>ID# UC-1.1.7.1.1</i>		<i>Engineer Tests ADR Cycle</i>		<i>1.0/09-16-98/RS/LCK</i>	
Goal	ADR Cycle Command Procedure is executed, resulting in the ADR being sufficiently tested.				
Preconditions	An ADR Cycle Command Procedure has been created to run against the ADR. The test environment for the ADR has been defined.				
Success End	Execution of ADR Command Procedure results in successfully testing the ADR.				
Failed End	Execution of ADR Command Procedure results in the engineer being unsuccessful in testing the ADR.				
Primary Actor	Engineer.				
Trigger Event	TBD.				
<i>Related Information</i>					
Due Date	TBD.				
Priority	TBD.				
Design Constraints	TBD.				
Parent use case(s)	UC-1.1.7.1: Engineer Checks Out ADR				
Child use case(s)	TBD.				
Scope Restrictions	TBD.				
Supporting actors	Instrument.				
Design Considerations	TBD.				
Dependent use cases	TBD.				
Information Sources	TBD.				
Open Issues	TBD.				



# Use Case: Engineer Tests ADR Cycle cont.



---

## Main Success Scenario

1. (OPTIONAL) Engineer reviews test plan for ADR.
2. Engineer establishes test environment (See <UC-TBD>).
3. Engineer begins execution of the named and saved ADR command procedure.
  - 3.1. System sends commands to ADR.
  - 3.2. System time-stamps and logs all commands/command procedure(s) triggered as result of main command procedure.
  - 3.3. Instrument sends telemetry to interested clients.
4. (OPTIONAL) Engineer terminates ADR Cycle testing.
  - 4.1. (OPTIONAL) Engineer executes a procedure to power down the instrument.

## Extension: Salt Pill Temperature Elevating Too Rapidly During Ramp Up

1. (OPTIONAL) Engineer notes the salt pill temperature concern by issuing a note in the log, including a snapshot of the data image (and a copy of the data selector).
2. (OPTIONAL) Engineer issues a manual command to adjust the current ramp up rate of the ADR.
3. (OPTIONAL) Engineer edits currently executing ADR Command Procedure, making modification based upon current state of ADR and what manual commands may have been sent to adjust the ADR state.  
(UNORDERED)
  - 3.1. (OPTIONAL) Modifies some "downstream" (not yet executed) commands.
  - 3.2. (OPTIONAL) Engineer modifies "up stream" (already executed) commands.
4. (OPTIONAL) IF {Engineer wants to overwrite the ADR Cycle Command Procedure} THEN
  - 4.1. Engineer performs a "Save" on the modified ADR Cycle Command Procedure. ELSE
  - 4.2. Engineer performs a "Save As" on the modified ADR Cycle Command Procedure.
5. Remainder of the ADR Cycle Command Procedure executes to completion.



# Class Responsibility Collaborator (CRC) cards



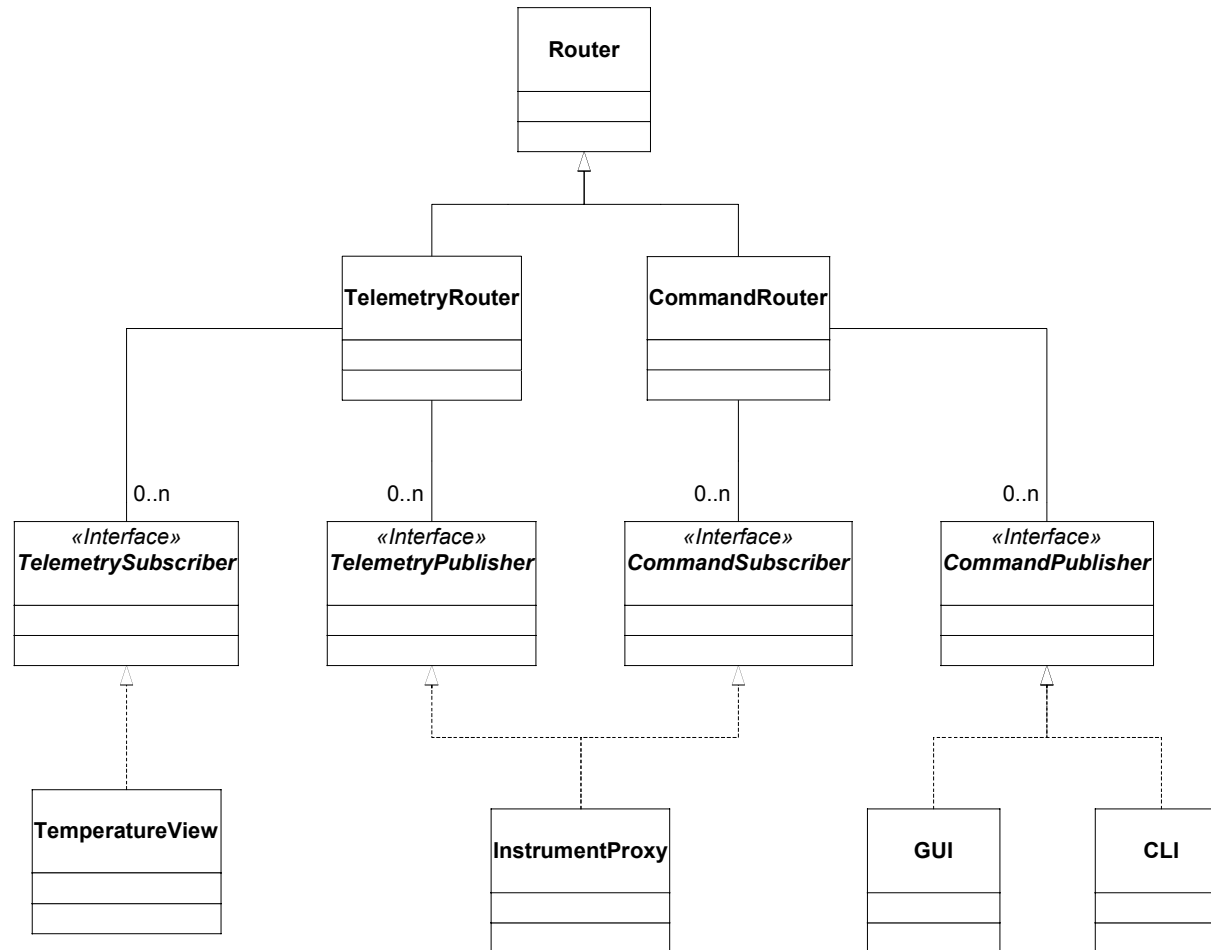
ClassName: Instrument Proxy	
Implements: <i>Command Subscriber, Telemetry Publisher</i>	
Receives raw telemetry stream from the instrument	Telemetry Parser
Creates Telemetry Objects based on raw data	Telemetry Objects
Publishes telemetry objects	Telemetry Router

ClassName: Telemetry Router	
Maintain list of telemetry subscribers	Telemetry Subscribers
Receives published telemetry	Telemetry Publishers
Notifies interested subscribers when new telemetry values are available	Telemetry Objects

ClassName: Temperature View	
Implements: <i>Telemetry Subscriber</i>	
Establish connection to a particular thermometer data stream	Namespace
Registers for temperature telemetry	Telemetry Router
Updates real-time display based on new telemetry data	



# Class Diagram





---

# Management and Build Plan

Management Plan

Design Schedule and Milestones

Test/Verification Plan

Risk Analysis and Mitigation



# Management Plan



- 
- All HAWC software activities consolidated into single IPT
  - IPT Chair Responsible for:
    - Coordination of all deliverables to support instrument integration and delivery to Observatory
    - Configuration Management
    - Defect Tracking
    - Requirements / Performance Traceability



# Multi-Pass Design and Implementation

---



- **Pass 1-3: High level design and prototype**
  - Pass 1 – Top Level Requirements, PDR materials (1998 Oct 1)
  - Pass 2 – Full Design (1998 Dec 22)
  - Pass 3 – Prototype Delivery (1998 Dec 22)
  
- **Pass 4-10: Lower level design and implementation cycles**
  - Pass 4 – Subsystem Integration (1999 March 15)
  - Pass 5 – Instrument Integration (1999 July 1)
  - Pass 6 – Functional Instrument Operations (1999 Oct 1)
  - Pass 7 – MCCS Simulator Operations (2000 May 1)
  - Pass 8 – Not-Expert User Interfaces (2001 Jan 2)
  - Pass 9 – Instrument checkout and flight operations (2001 July 1)
  - Pass 10 – Outside Team Operations (2001 Oct 1)
  
- **Pass 11: Post Delivery Clean up.**



# Test/Verification Plan



- 
- Full Life-Cycle Object Oriented Testing
  - Explicitly included throughout individual passes' development cycle
  - Testing in Analysis and Design phases
  - Use Case Scenario Testing (full bi-directional traceability)
  - Requirements Reviews
  - Usability Testing for all user interfaces
  - Code Tests
  - Automated regression testing (grows with each iteration)



# Test/Verification Plan (cont.)



- 
- Integrated test cases in delivered code, where possible
  - Method and class/type testing
  - System Testing
  - Comprehensive functional testing against new requirements for pass - explicit bi-directional traceability between tests and use cases
  - Stress testing, particularly for performance critical areas
  - User acceptance testing (includes user and designer documentation, revisits usability testing of interfaces)
  - Test results are an input into the next development pass



# Risk Analysis and Mitigation Plan

---



- *Note:* Risk assessments are based on the state of technology today. Based on current rates of progress, we expect lower risk in each case below within 1 year.
- Baseline design has moved data acquisition from hardware (DSP) to software on a CPU.
  - Risk assessment: moderate
  - Alternatives/Impact: specialized or multi-processor architecture. Potentially higher costs.



# Risk Analysis and Mitigation Plan (cont.)

---



- Software Frameworks: immaturity on some platforms
  - Risk assessment: low
  - Alternatives/Impact: Assign performance critical processes to low risk platforms. Less flexibility in distributing processes.
- Software Frameworks: immaturity of new API's
  - Risk assessment: low
  - Alternatives/Impact: Implement functionality based on existing APIs. Potentially longer development time or possibly eliminating some desirable but not critical features.



# Risk Analysis and Mitigation Plan (cont.)

---



- Logistics: distributed development team
  - Geographic distribution of team poses some difficulties for collaborative development.
  - Risk assessment: low to moderate
  - Alternative/Impact: Provide opportunity for more direct, face-to-face development (pending budgetary constraints). Improve quality of collaborative tools and tools for teleconferencing. Cost of travel increases.