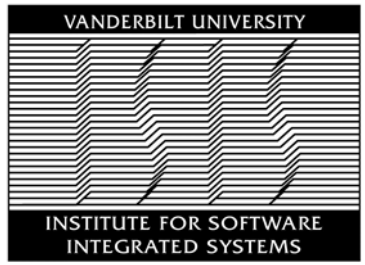
APPLICATION OF SOFTWARE HEALTH MANAGEMENT TECHNIQUES

NASA Cooperative Agreement NNX08AY49A

Nagabhushan Mahadevan
Abhishek Dubey *
Gabor Karsai
Institute for Software Integrated Systems
Vanderbilt University
Nashville TN

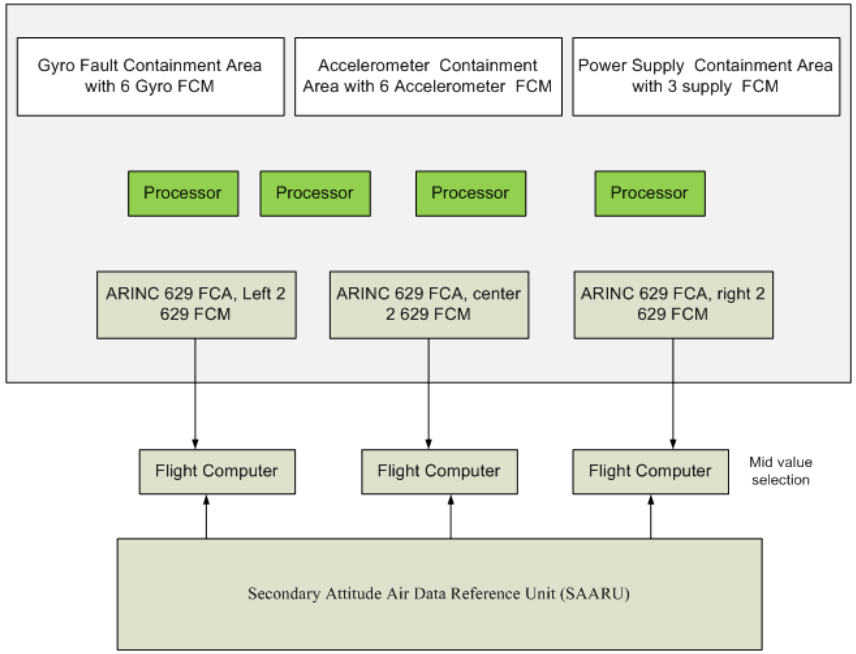


Software Failure: Malaysian Air (Boeing 777) in-flight upset (2005)

- Airplane's autopilot experienced excessive acceleration values.
- Autopilot pitched nose-up to 17.6 degree and climbed at a vertical speed of 10,650 fpm.
- Airspeed reduced to 241 knots and aircraft descended 4,000 ft.
- Re-engagement of autopilot followed by another climb of 2,000 ft.

Source: Google Images

Contributing Factors: "An anomaly existed in the component software hierarchy that allowed inputs from a known faulty accelerometer to be processed by the air data inertial reference unit (ADIRU) and used by the primary flight computer, autopilot and other aircraft systems."
-- from
http://aviation-safety.net/database/record.php?id=20050801-1

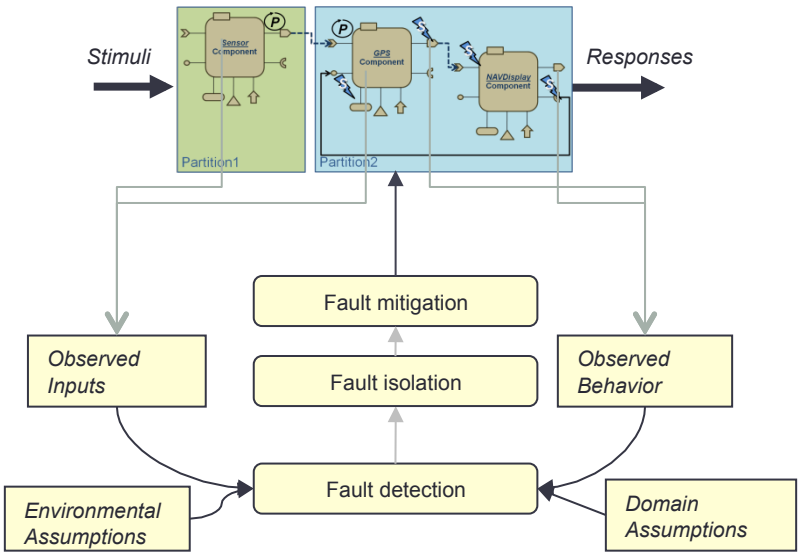


Based on Air Data Inertial Reference Unit (ADIRU) Architecture (ATSB, 2007, p.5)

Software Health Management

- Software is a complex engineering artifact.
- Software can have latent faults.
- Faults appear during operation when unforeseen modes or interactions happen.
- Techniques like Voting and Self-Checking pairs have shortcomings
  - Common mode faults
  - Fault cascades

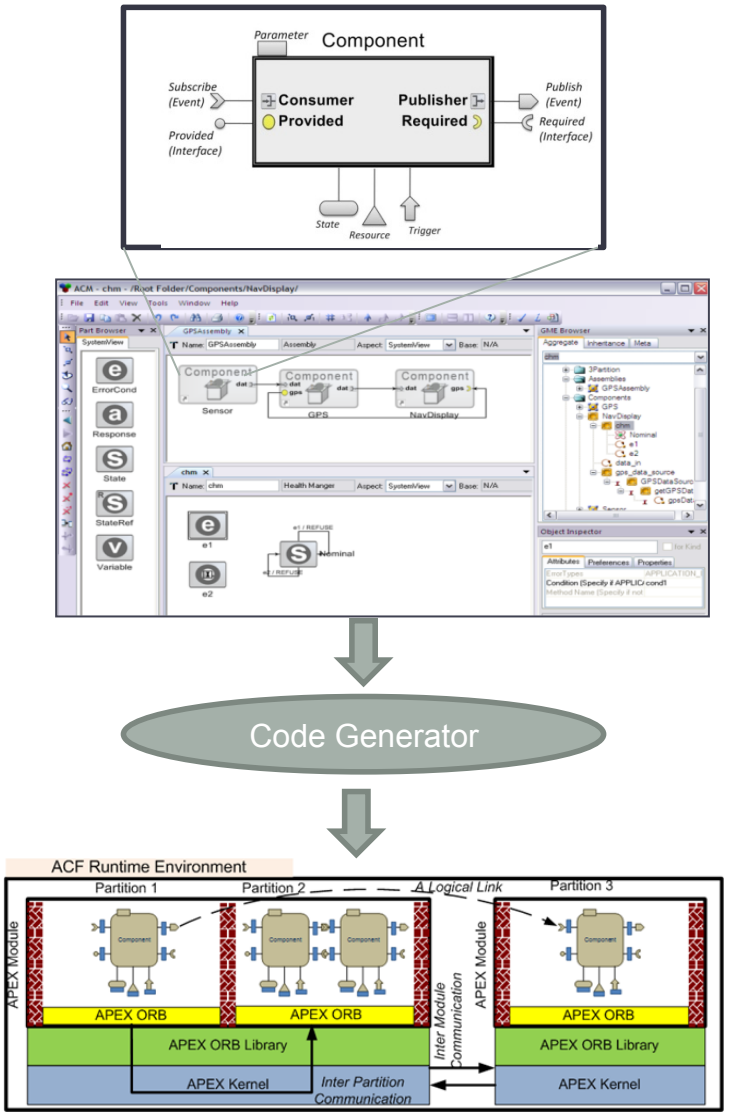
- SHM is the extension of FDIR techniques in Physical systems to Software.



OUR APPROACH

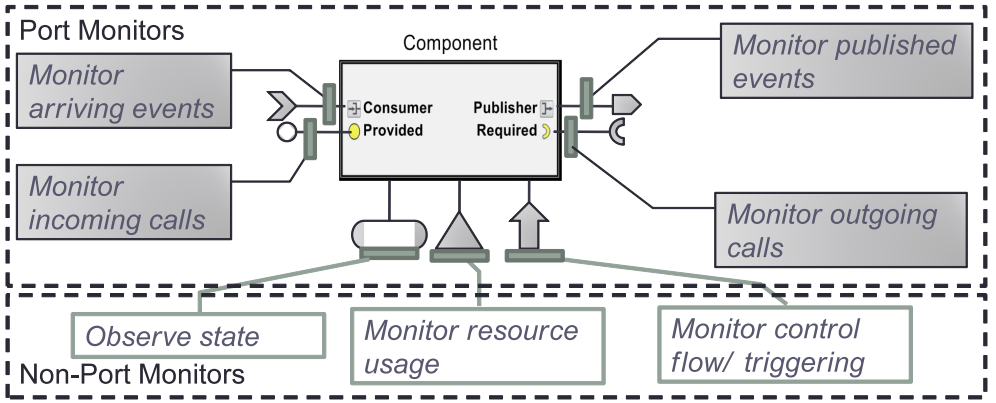
System Structure

- Component-based system.
  - Strictly Specified Interactions
    - Synchronous Interfaces: call/return
      - Periodic: time-triggered
      - Aperiodic: event-triggered
    - Asynchronous Interfaces: publish-subscribe
      - Periodic: time-triggered
      - Aperiodic: event-triggered
- **Model-Based Software Development**
  - Specification of Monitoring expressions
  - Specification of Reactive Mitigation
- Hard-Real time ARINC-653 based runtime
  - Spatial separation
  - Temporal separation
  - Deadline Monitoring



Anomaly Detection

- Model-Based Specification of Monitoring expressions
  - Post/Pre condition violations: threshold, rate, custom filter (moving average)
  - Resource Violations: Deadline
  - Validity Violation: Stale data on a consumer
  - Concurrency Violations: Lock Time Outs.
  - User code violations: reported error conditions from application code.
- Code Generators
  - Synthesize code for implementing this monitors in the system.

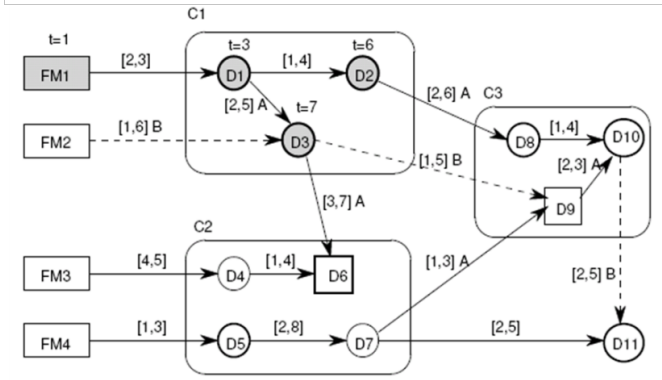


- Based on these local detection, each component developer can implement a local health manager
- It is a reactive timed state machine with pre specified actions.
- All alarms, actions are reported to the system health manager

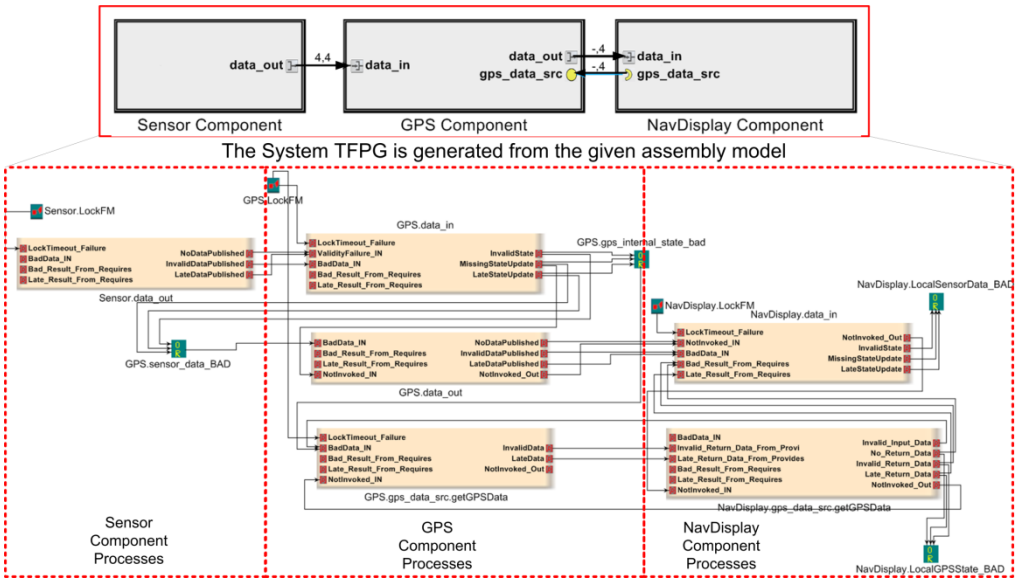
Failure Diagnosis (Isolation)

- Model-based diagnoser is automatically generated
- Specification of data and control flow inside each component.
- Code Generators
  - Generate Failure Propagation graphs based on the system assembly and deployment information.
  - We use fault propagation templates developed beforehand for each category of component interaction.
- Overall, this gives a full timed failure propagation graph of the system.
- This graph is used to generate (multiple root) failure hypothesis that can explain the alarms seen in the system.
- Hypothesis are ranked by
  - Plausibility
  - Robustness

A TFPG model ($t = 10$, Mode=A $\forall t \in [0, 10]$)



Example: GPS Assembly

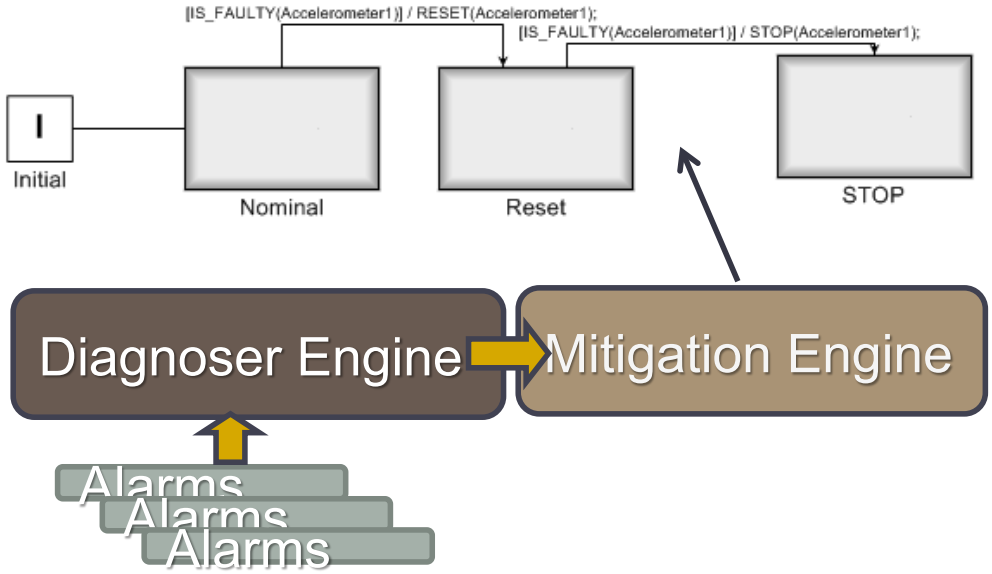


System-level Fault Mitigation

- Model-based mitigation specification at two levels
  - Component level: quick action
  - System level: Reactive action taking the system state into consideration
  - System designer specifies them as a parallel timed state machine.
- Fixed set of mitigation actions are available
- Runtime code is generated from models
- **Advantages:**
  - Models are higher-level programs to specify (potentially complex) behavior – more readable and comprehensible
  - Models lend themselves to formal analysis – e.g. model checking

List of predefined Mitigation Actions

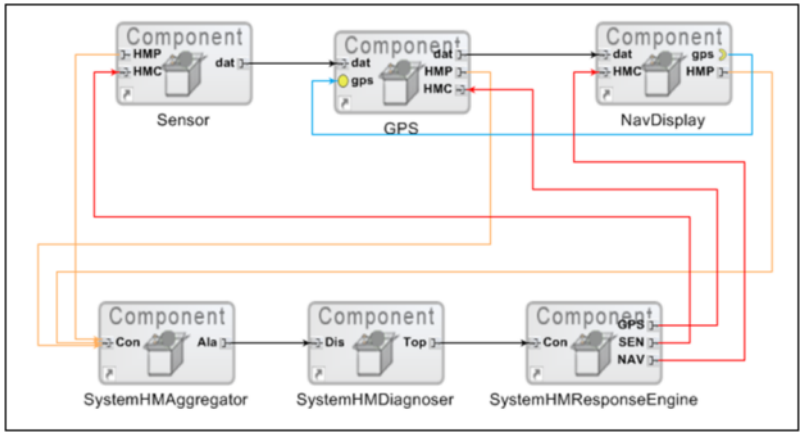
CLHM:ABORT	Discontinue current operation, but opera-
	tion can run again
CLHM: USE PAST DATA	Use most recent data (only for operations that expect fresh data)
CLHM: STOP	Discontinue current operation
	Aperiodic processes (ports): operation can run again
	Periodic processes (ports): operation must be enabled by a future START HM action
CLHM: START	Re-enable a STOP-ped periodic operation
CLHM RESTART	A Macro for STOP followed by a START for the current operation
SLHM: RESET	Stop all operations, initialize state of compo start all periodic operations
SLHM: STOP	Stop all operations



System-level Health Management: Data Flow

- 1. Aggregator:
  - Integrates (collates) health information coming from components (typically in one hyperperiod)
- 2. Diagnoser:
  - Performs fault diagnosis, based on the fault propagation graph model
  - Ranks hypotheses
  - Component that appears in all hypotheses is chosen for mitigation
- 3. Response Engine:
  - Issues mitigation actions to components based on diagnosis results
  - Based on a state machine model that maps diagnostic results to mitigation actions

These are automatically generated



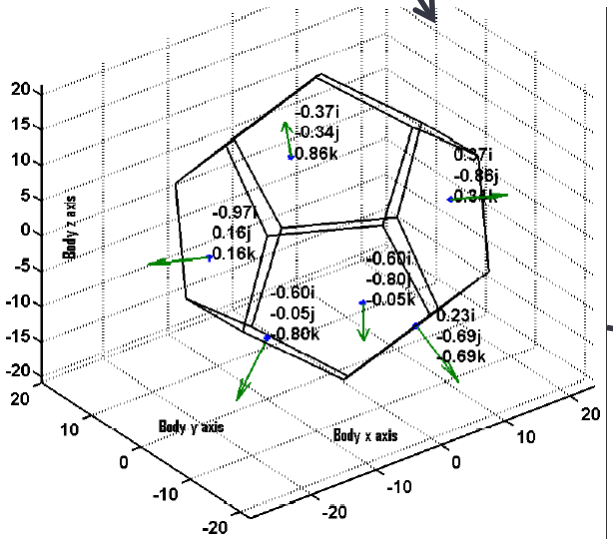
The Health Management flow::

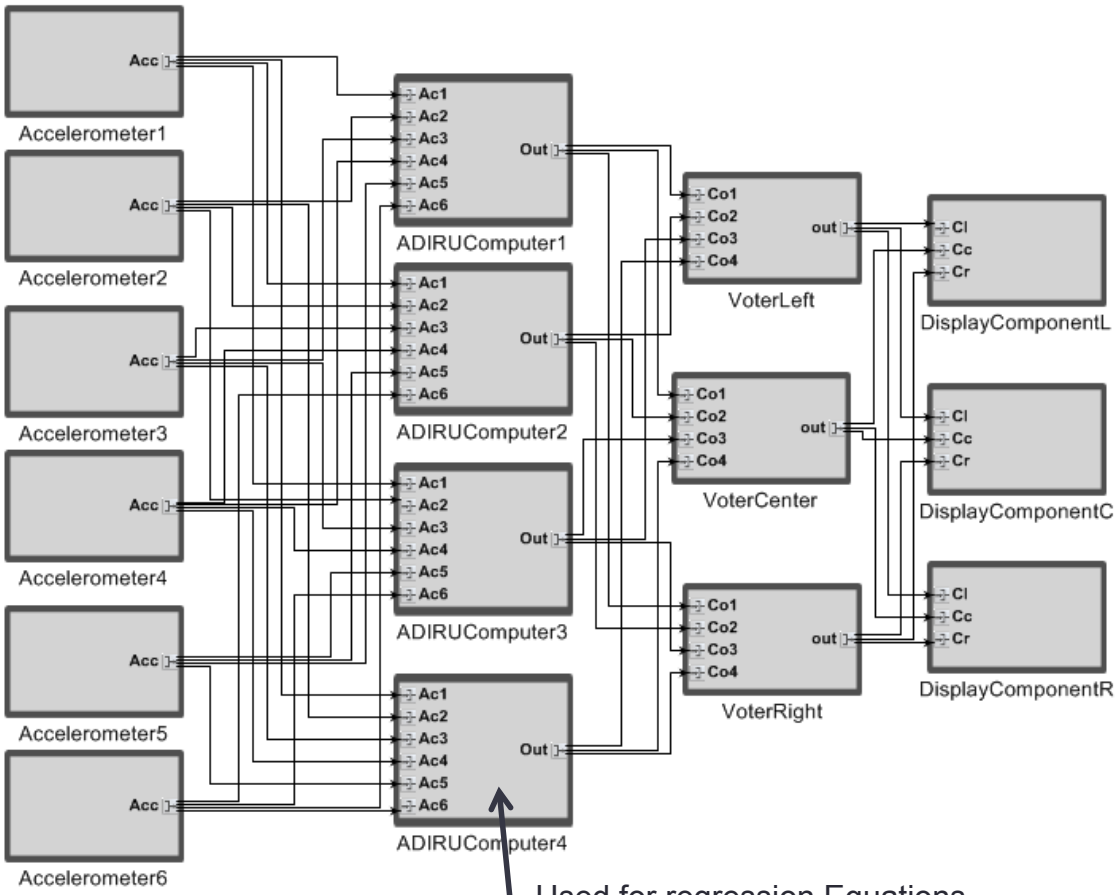
1. *Locally detected anomalies are mitigated locally first. – Quick reactive response.*
2. *Anomalies and local mitigation actions are reported to the system level.*
3. *Aggregated reports are subjected to diagnosis, potentially followed by a system-level mitigation action.*
4. *System-level response commands are propagated to components.*

CASE STUDY

Case Study

- Modeled the architecture as a software component assembly
- Created the fault scenario
- Only modeled part of the system to illustrate the point of SHM
- Accelerometers are arranged on six faces of a dodecahedron.

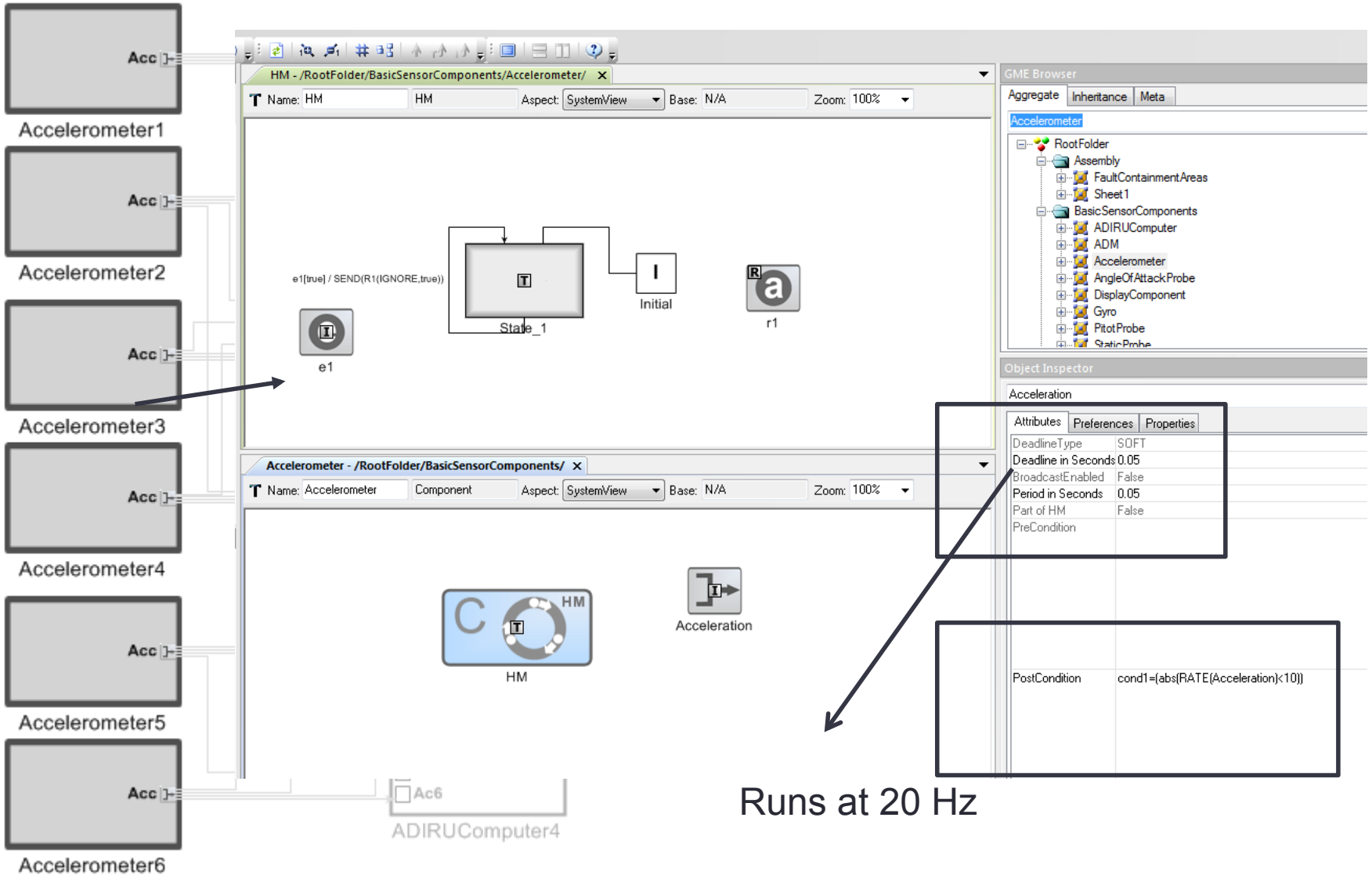




Used for regression Equations

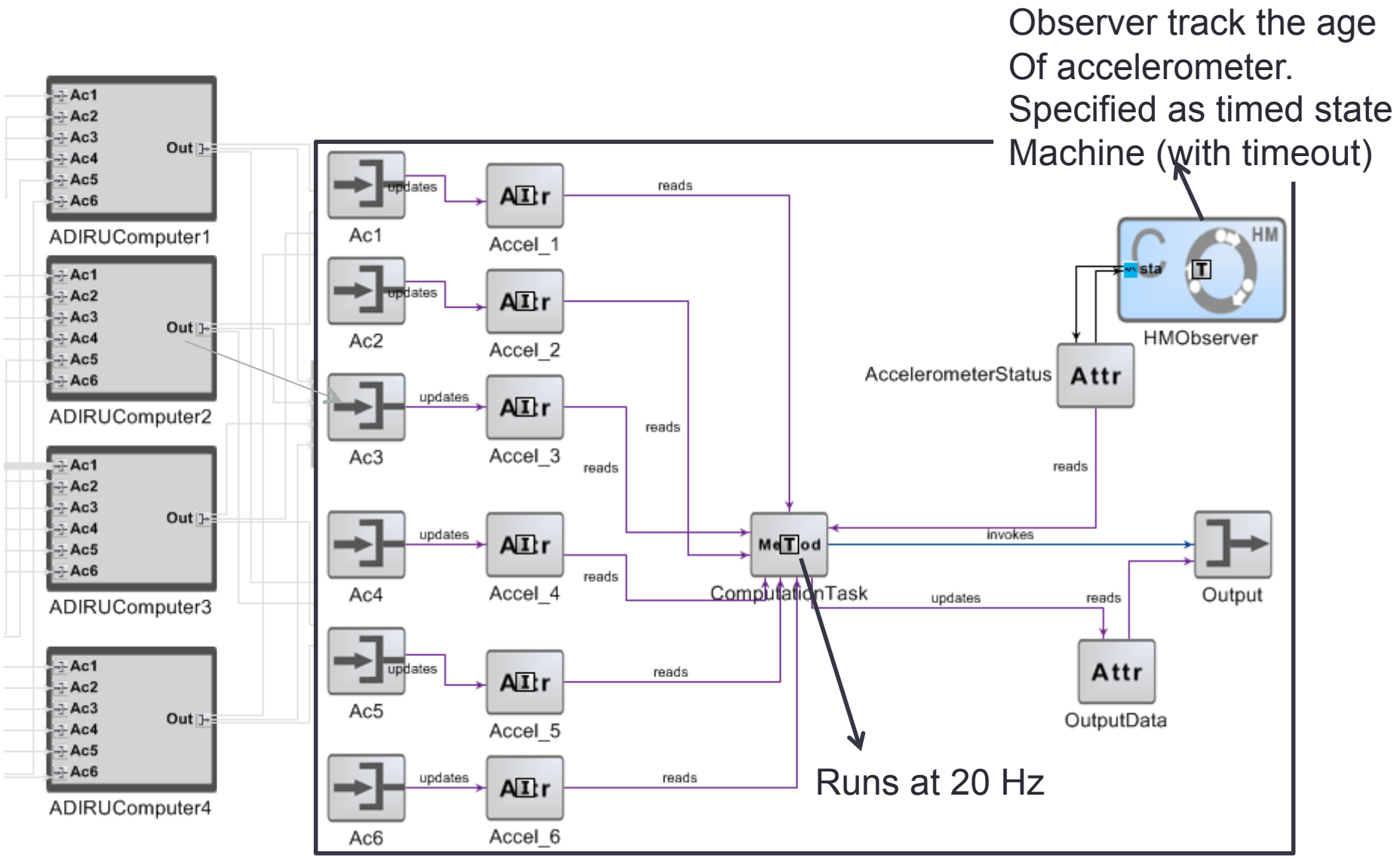
$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \end{bmatrix} = \begin{bmatrix} -0.3717 & -0.3386 & 0.8644 \\ 0.3717 & -0.8644 & 0.3386 \\ -0.6015 & -0.7971 & -0.0536 \\ -0.9732 & 0.1625 & 0.1625 \\ -0.6015 & -0.0536 & -0.7971 \\ 0.2298 & -0.6882 & -0.6882 \end{bmatrix} \times \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} + \mathcal{N}$$

ADIRU Assembly (Accelerometers)

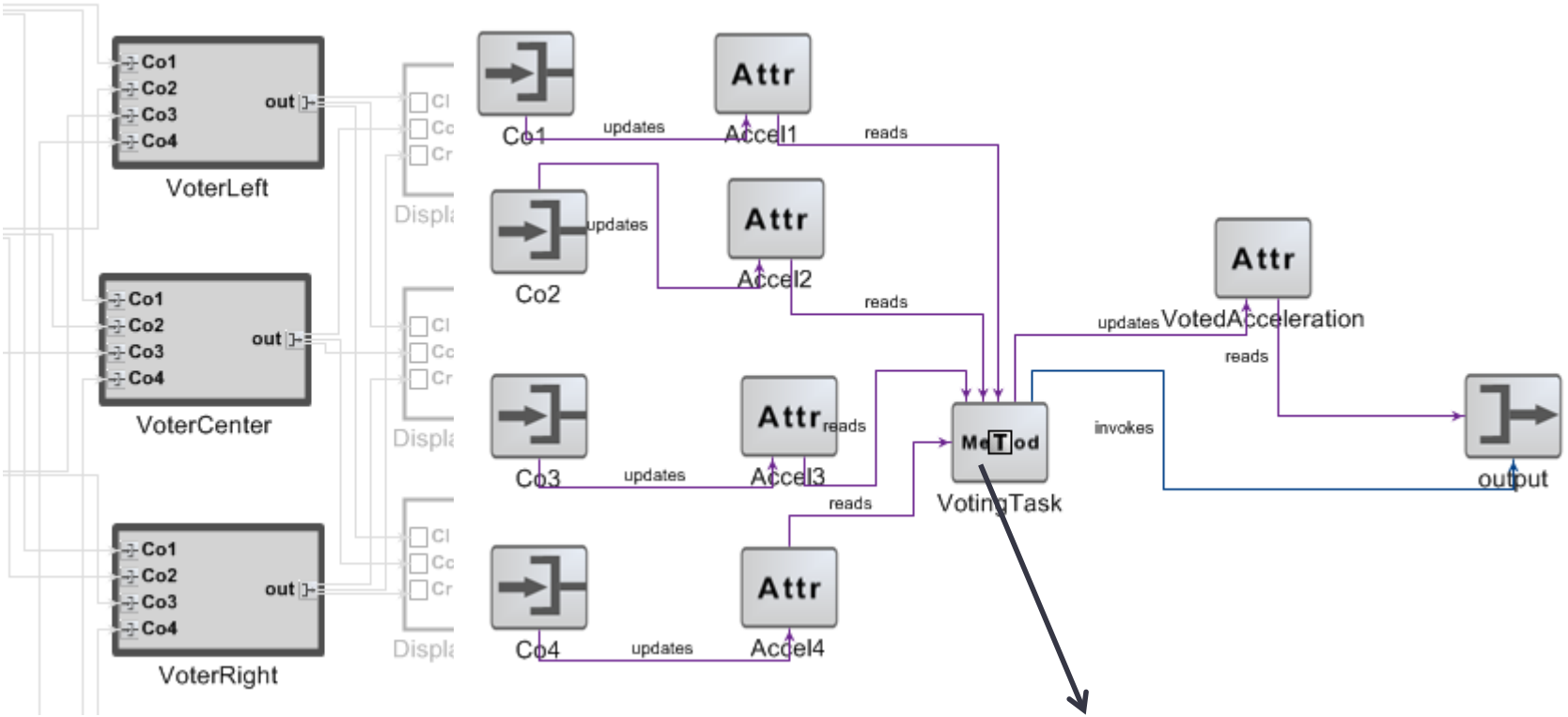


Runs at 20 Hz

ADIRU Assembly (Processors)

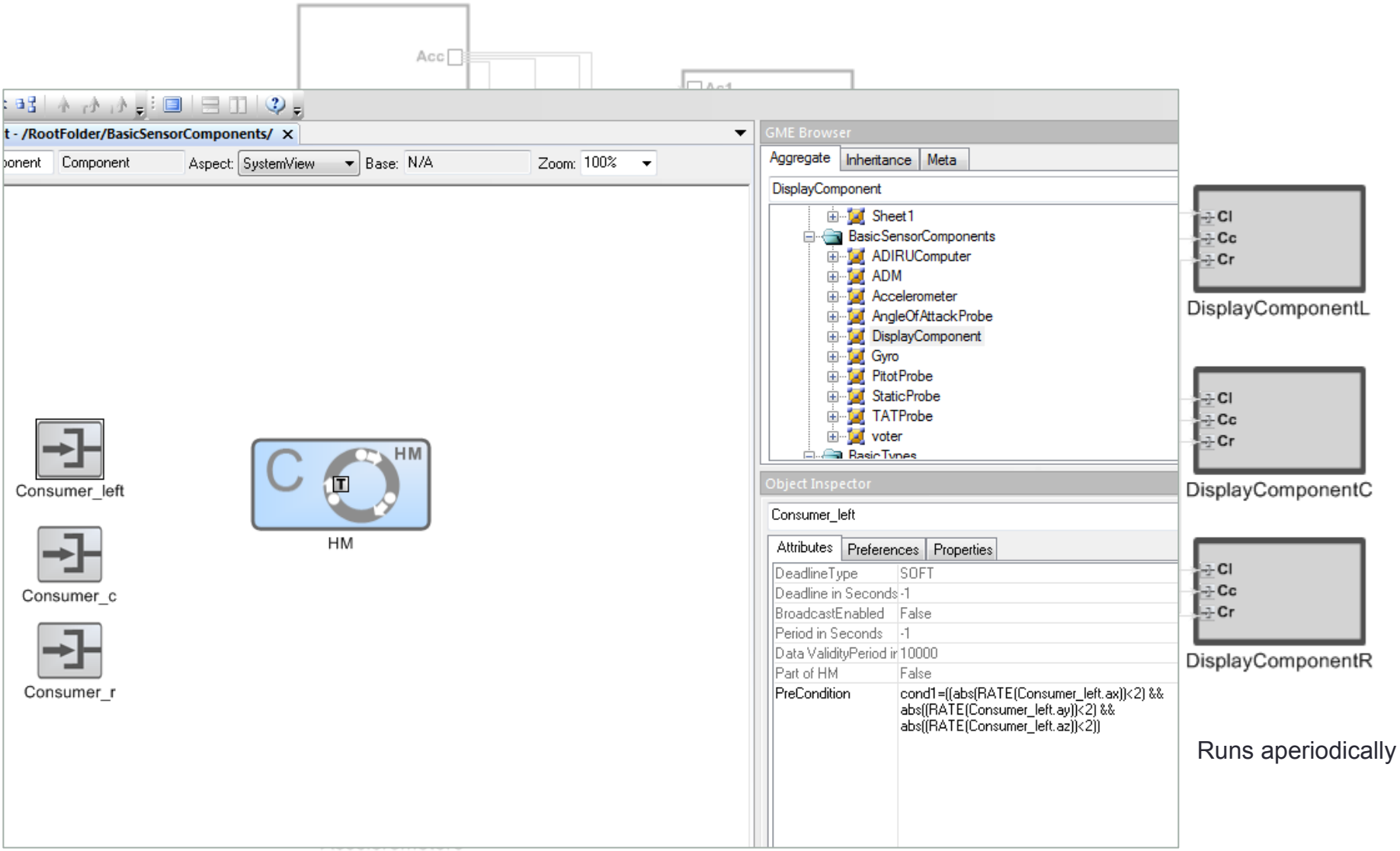


ADIRU Assembly (Voters)



Runs at 20 Hz

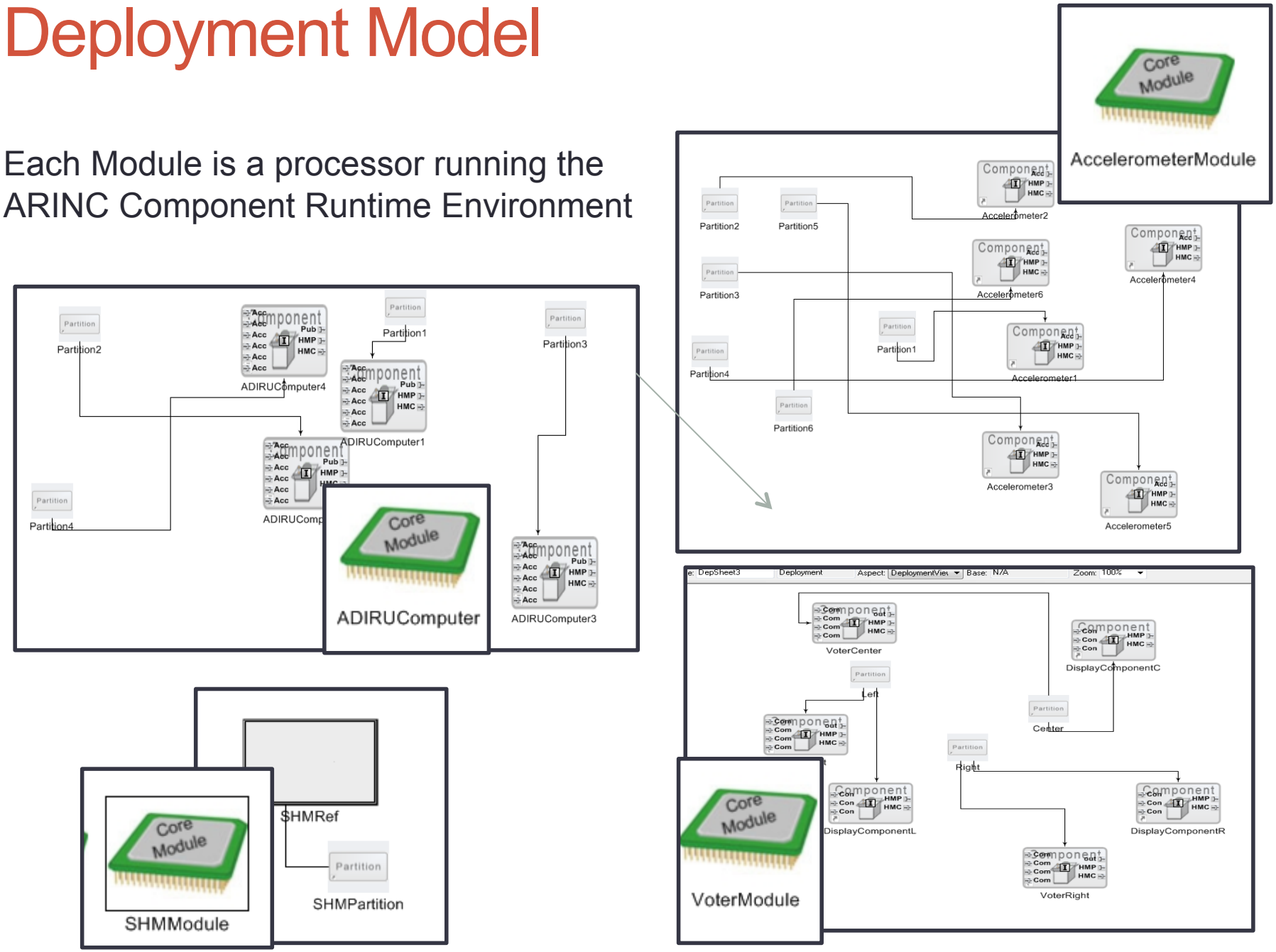
ADIRU Assembly (Display- Mimics PFC)



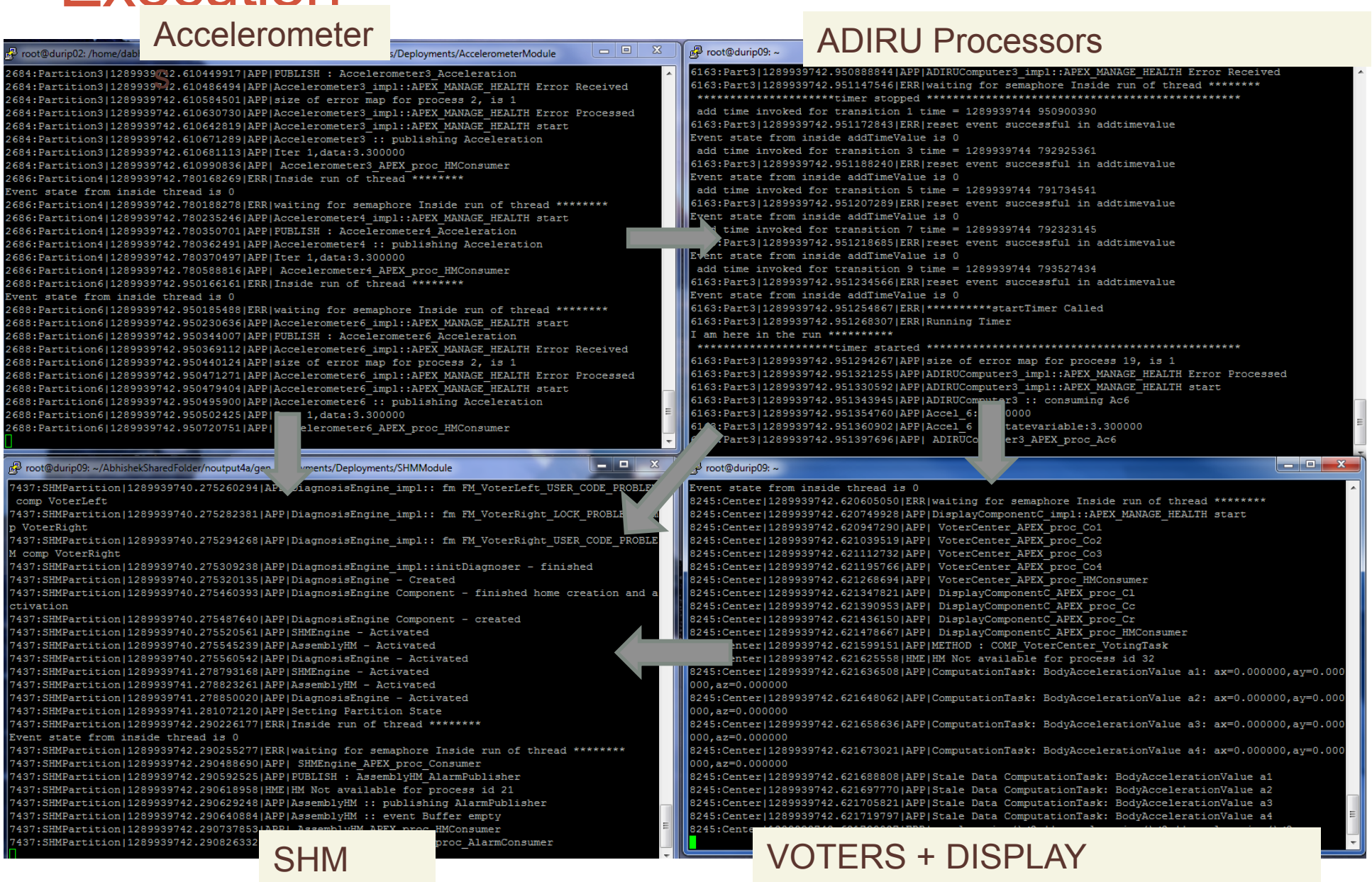
Runs aperiodically

Deployment Model

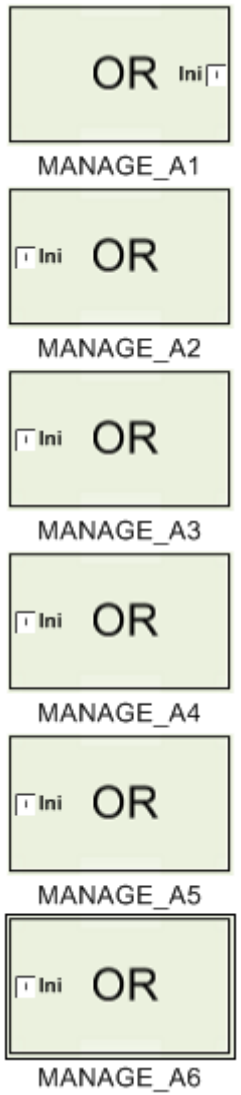
Each Module is a processor running the ARINC Component Runtime Environment

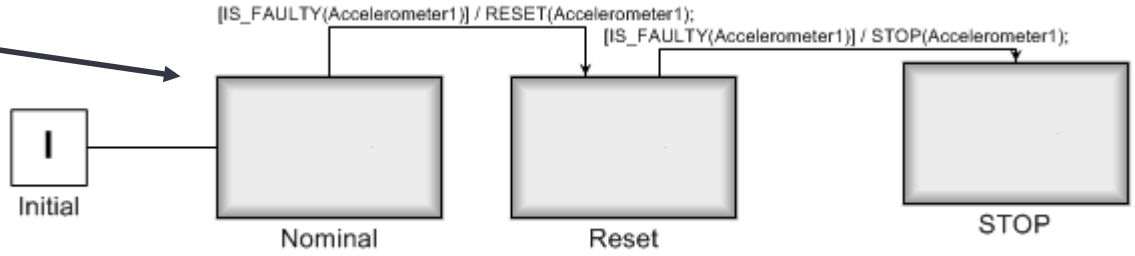


Execution

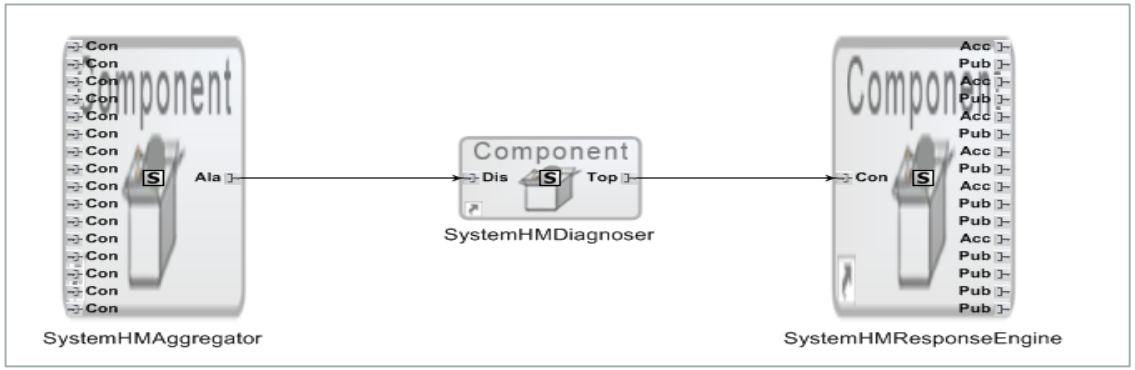


System Health Manager





other machines have similar specification



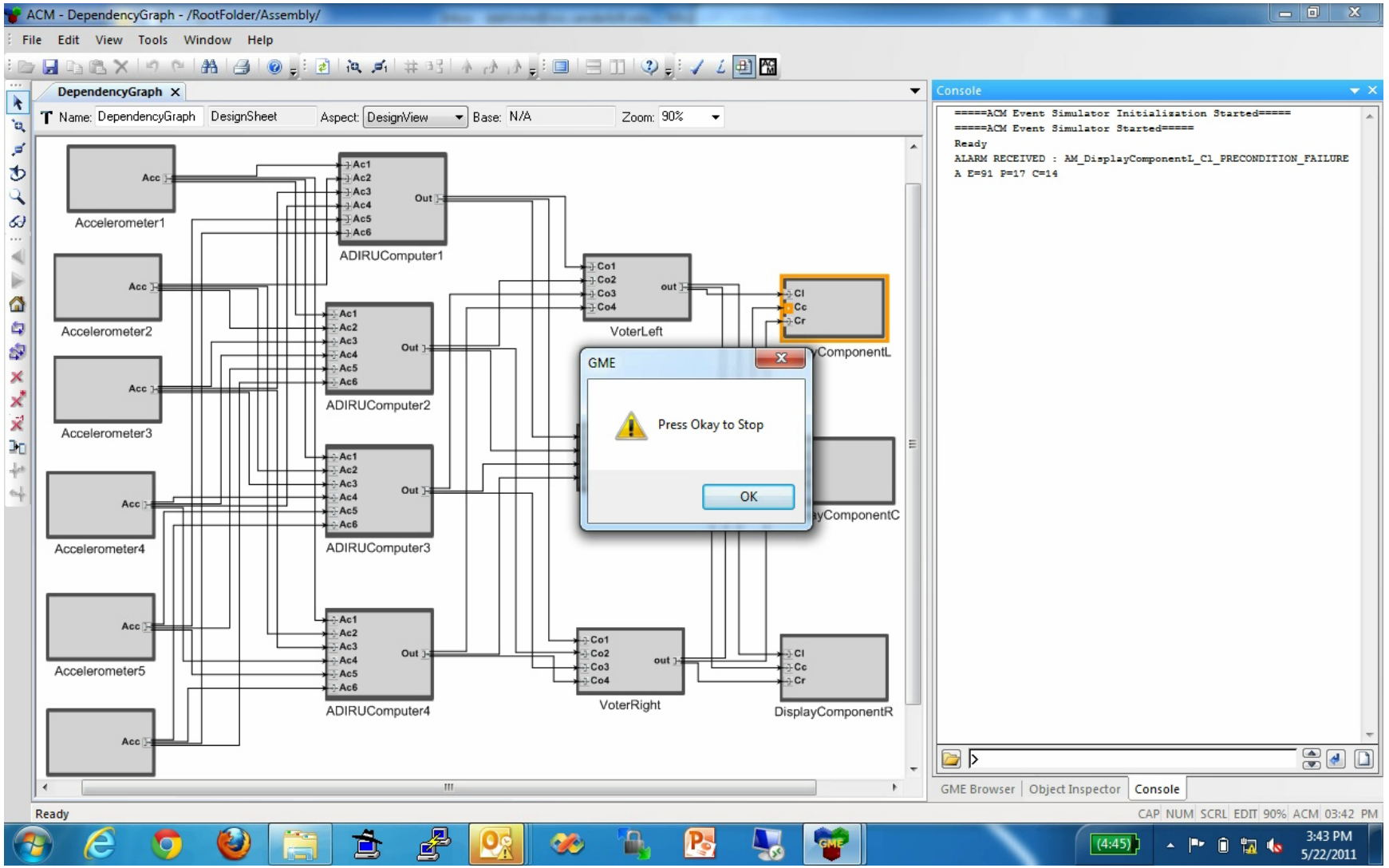
These components are auto generated

The hypothesis generated by the diagnoser is translated to Component(s) that is most likely faulty. This list is fed to Response Engine, which triggers the mitigation state machine

Demonstration

- Fault Scenario
- Accelerometer 5 has initial fault
- It is started which causes an alarm
- Then Accelerometer 6 develops fault
- Successful mitigation
  - Identifying the faulty components
  - Stopping the fault components
  - Processors can still function with four accelerometers.

Demonstration: Faulty Scenario (Movie)



Conclusion

- Software health management is feasible and affordable using a model-based, component-oriented approach.
- We showed the results of an experiment that simulated partial abilities of a Boeing 777 ADIRU.
- Component framework is essential
  - Managed component interactions
  - Monitored interfaces
- Architecture and component models are needed for deriving fault models and to determine correct mitigation actions
- Fault diagnostics across components is relevant to isolate root causes
- Mitigation can be reactive (with pre-determined reactions) or deliberative (with dynamically computed reactions)
- Deliberative mitigation is the subject of ongoing research.