**Objective**

Software health management solutions using model-based software development techniques

The specific goals of the research:

1. To develop a comprehensive approach to modeling and managing faults in software in the context of the system the software is applied
2. To construct (a) a prototype modeling tool, and (b) a prototype multi-layered software health management engine that implements the principles involved
3. To demonstrate the feasibility of the approach through experiments using realistic problems, systems, and scenarios.

**Perspective on the results:**

Model-based software tools that assist software developers in building complex airborne software that is robust against systemic faults – can have a positive impact on safety and supportability.

**Technical Approach**

- Platform: Component framework for ARINC-653
- Fault detection: Monitoring component interfaces
- Fault isolation:
  - Component-level: detection → isolation
  - System-level: system-wide diagnostics based on Timed Failure Propagation Graph (TFPG) model.
- Component-level fault mitigation:
  - Reactive state machine
- System-level fault mitigation:
  - Reactive state-machine with pre-determined mitigation actions
  - Deliberative reasoner computes reactions to restore functionality (in progress)

**Analysis**

- Component interactions are restricted to those supported by the software framework.
  - Choices: (Synchronous (Provides/Requires), Asynchronous (Publish/Consume)) X (Periodic, Aperiodic) X (Intra, Inter Partition) May decrease performance, but makes the system analyzable
- Component architecture is explicit, includes data flow and control flow inside the components.
- Observations of state transitions may be used to detect licensing issues.
- Latent defects are detected through observations on the component interfaces.
- Concurrency violations: Lock time-outs.
- Root causes for fault effect cascades are isolated on the system level
- Reactive mitigation actions are specified by the designer
- Deliberative mitigation actions are determined based on the state of the system and the active functions

**Technical Challenges**

Software is a complex engineering artifact that can have latent faults, uncaused by testing and verification. Such faults become apparent during operation when unforeseen modes and/or (system) faults appear.

- Fault detection: Monitors and detects anomalies in the running software
- Fault isolation: Identifies the source(s) of the fault(s)
- Fault mitigation: Takes corrective action to counter the fault.

**Challenge:** Tools that assist developers in constructing the fault detection, isolation and mitigation services with minimal impact on performance and developer’s productivity

**Results**

- ARINC-653 Component Framework (ACM)
  - ACM Tools
    - Modeling language
    - Software generators
    - Failure Cascade Patterns (templates)
  - Component-level Health Manager
    - Monitoring engine
    - Reactive mitigation engine
  - System-level Health Manager
    - Fault diagnostics engine
    - Reactive mitigation engine
  - Deliberative mitigation engine (in progress)
- Demonstrative examples
  - Simple GPS example with 3 components – reactive, local mitigation
  - Medium ADRU example – common mode cascading failure, system-level mitigation
  - Larger avionics example – redundant subsystems, system-level faults

**Conclusion**

- Software health management is feasible and affordable using a model-based, component-oriented approach
- 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.

---

**Model-Based Techniques for Software Health Management**

Abhishek Dubey, Gabor Karsai, Nagabhushan Mahadevan

(Institute for Software Integrated Systems, Vanderbilt University)

Sponsor: NASA Aviation Safety program