Model-based Software Health Management

Gabor Karsai
Vanderbilt University/ISIS

Supported by NASA Cooperative Agreement NKX08AY49A
What is ‘Systems Health Management’?

The ‘on-line’ view:
1. Detection of anomalies in system or component behavior
2. Identification and isolation of the fault source/s
3. Prognostication of impending faults that could lead to system failures
4. Mitigation of current or impending fault effects while preserving mission objective/s
What is ‘Model-driven Software Development’ for Control Systems?

- Models are (partial) higher-order, domain-specific programs
  - Models should capture the ‘what’ as well as the ‘how’
  - Models are specs, designs, implementations

- Software is modeled, and part of the implementation could be generated from models

- Different modeling languages for different stages of development:
  - Component modeling, system modeling, …

- The modeling language is often tailored for a “domain”
  - Distributed, real-time embedded systems running on a specific runtime platform

- System integration is helped via (1) the code generators and (2) the execution environment/platform

- Models can be analyzed (at design-time)
  - Consistency, dependencies, schedulability, timing, …

- Models could also be used at run-time!
What is ‘Model-based Software Health Management’?

- The implementation of software health-management function enabled by the tools and techniques of
  - Model-driven software development
  - Model-based systems health management

![Diagram showing the relationship between Models for Software and Systems, Model-based System Health Management Techniques, and Model-based Software Health Management.](diagram.png)
‘Ground Rules’

- Software and system specifications are known and available in the form of models with precise semantics
  - Specify the ‘shall’-s and the ‘shall not’-s
  - Have an executable interpretation that can be used at run-time
  - May be incomplete (but should indicate ‘don’t know’-s)
  - 

- Software implementation is well-designed and well-tested
  - MBSHM is not a replacement for proper software practices

- Software (and the system) has latent, residual faults that slipped through verification and testing

- Software is built from components with well-defined interfaces
  - Components run on a ‘virtual machine’: a computational platform that provides fault containment capabilities
  - The platform mitigates all component interactions
  - Components are ‘observable’: input/s, output/s, and state are accessible
  - Pre- and post-conditions and invariants for component operations are known
  - Components are transactional and restartable
What’s new

- Models are available at run-time
  - Computationally feasible to use them for checks
- Robust computational platform
  - Provides fault mitigation for lower-level faults and does not allow their ‘upwards’ propagation
- Systems view
  - Software behavior is understood in the context of the systems and its mission objectives
- ‘Closing the loop’ on a higher level
  - Health management closes a feedback/ control loop over the software

The price:

- Requirements analysis and modeling is essential and it is a significant factor in effort
- Layered robust platform must implement the ‘assume-guarantee’ principle
- Systems engineers and software engineers must work together
Steps towards Model-based Software Health Management: Components

- ‘Component Health Manager’ for every relevant component:
  - To monitor component behavior
  - To mitigate component faults in the local context
- CHM is generated (instantiated) from the component model via a generator
  - Derive monitor from requirements model
  - Isolation: interpret anomalies
  - Mitigator is generated from design models of mitigation logic
- Implements a reflex-based, local fault management scheme
- See: Safety monitor
  - DO-178B Sec 2.3.3
Steps towards Model-based Software Health Management: Systems

- Cascading faults: non-local fault effects that propagate to other components
  - Model: causal/temporal fault propagation graph
  - Fault source isolation: TFPG engine
  - Mitigation: System-level mitigation strategies implemented by a model-based engine

- System-level models and mitigation that could take into account physical component faults
  - Mitigation needs state estimates for software health as well as system health
  - Mitigation strategies are pre-defined, their selection is dynamic based on actual failure modes

- Detection, diagnosis and mitigation engines are model-driven and/or synthesized by generators
Steps towards Model-based Software Health Management: Systems

- Systemic faults: functional faults that appear on the system level, even when no low-level anomalies are detected
  - Detect and interpret faults in the system context
  - Isolate failing function in the current ‘feature model’ of the system
  - Derive updated (and feasible) feature model – planning based on state estimates for software and system health
  - Reconfigure / reset components
- Use active feature models: a modeling technique to capture all possible and legal feature combinations / component configurations
  - Search: via constraint solving
Issues in Software Health Management

- What technological barriers must be overcome to make SW Health Management feasible?
  - Robust and trusted architecture and reusable architectural elements
  - Trusted generators/engines for monitors, diagnosers, mitigators

- What constitutes a valid model for SW Health Management?
  - There is a simple, robust, high-reliability infrastructure for health management that is feasible to instantiate from well-defined models

- How do we prevent SW Health Management capabilities from becoming a source of problems? How do we guard against the guardian?
  - Subject the guardian to a higher degree of verification level
    - Verify the engines and the models
    - Verify the generators (or use instance-based verification)

- Is there a meaningful notion of prognosis for software defects?
  - Software faults are always latent, one running instance of the software probably does not yield enough data to make sensible prognostics… but data collected fleet-wide possibly could! Needs: software anomaly logging.

- What constitutes sufficient evidence that a Software Health management system improves safety?
  - Formal verification of requirements. monitors, diagnosers, mitigators for fault → safety cases
  - Effects of seeded, antagonistic, safety-critical faults are successfully mitigated (testing)