Model-based
Software Health Management

Gabor Karsai
Vanderbilt University/ISIS

Supported by NASA Cooperative Agreement NKX08AY49A
The idea

Software Health-Management:
Application of Systems Health Management concepts and techniques in the area of Software-Intensive Systems

Why?

- Software is a crucial ingredient in aerospace systems
  - Software as a method for implementing functionality
  - Software as the ‘universal system integrator’
- Software could exhibit faults that lead to system failures
  - Software complexity has progressed to the point that zero-defect systems (containing both hardware and software) are very difficult to build
- Systems Health Management is an emerging field that addresses precisely this problem: how to manage systems’ health in the case of faults
Sources

- **Classical fault tolerance** (typically for hardware)
  - Random events of ‘nature’
  - Redundancy and independence to achieve desired reliability
  - Systems ‘run’ without interruption
  - *Does not work well for software: n-version programming based on the same requirements/design does not yield independence*

- **Software fault tolerance**
  - Check and catch low-level faults in the code, raise exceptions
  - Handle exceptions via hierarchical exception handling
  - Exceptions propagate to the ‘right’ level of the hierarchy and are handled there
  - *Complex development process, designers must anticipate failures, hierarchical exception handling is difficult to manage, does not handle faults triggered by errors in specification (Ariane 5)*

- **Fault protection**
  - For anticipated faults, design the ‘monitor → response’ reaction
  - Enumerate faults, design a mitigation strategy for each case
  - *Protection based on ‘reflex reactions’, very little knowledge of system ‘state’ or ‘goals’*
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’
- Software is modeled, and part of the code is 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, …
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 health management
‘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 (‘don’t know’-s)
- Software implementation is well-designed and well-tested
  - MBSHM is not a panacea that covers up weak 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 fully accessible
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 propagating them ‘upwards’
- Systems view
  - Software behavior is understood in the context of the systems and its mission objectives
- Closing the loop on a higher level
  - It is a feedback control loop over the software and the system

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 requirements model via a (verified) generator

- Implements a reflex-based, local fault management scheme
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

- Engines are model-driven and synthesized by (verified) 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
Model-based Software Health Management

- What has changed?
  - Use models at run-time – computing power is affordable
  - Consider faults in the system, not only in the software or only in the hardware but in both
  - Unanticipated combinations – let a search engine find them

- Will it work? Will it work safely? Will it work reliably?
  - Challenges:
    - Complexity of models
    - Handling undefined cases in the requirements model
    - Mitigation strategies
    - Verification

- MBSHM should be not only a technology but also an engineering process and mindset.