Challenges in Validating Safety-Critical Embedded Systems

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Peter H Feiler
Nov, 2009
Outline

Need for Predictable System Integration
Multi-fidelity Model-based Analysis
Validation of Implementations
Towards Architecture-Centric Engineering
Embedded software systems introduce a new class of problems not addressed by traditional system modeling & analysis.

"This appears to be a unique event," the bureau said, adding that Toulouse, France-based Airbus, the world's largest maker of commercial aircraft, issued a telex late yesterday to airlines that fly A330s and A340s fitted with the same air-data computer. The advisory is "aimed at minimizing the risk in the unlikely event of a similar occurrence."

Even with the autopilot off, flight control computers still "command control surfaces to protect the aircraft from unsafe conditions such as a stall," the investigators said.

The unit continued to send false stall and speed warnings to the aircraft's primary computer and about 2 minutes after the initial fault "generated very high, random and incorrect values for the aircraft's angle of attack."
Mismatched Assumptions Cause Many Issues

Why do system level failures still occur despite best design methods & fault tolerance techniques being deployed in systems?
Problem Root Cause Areas in Software Runtime Architecture

Violation of data stream assumptions
- Data representation: units, base types
- Time sensitive data: Latency, jitter & age contributors
- Stream characteristics: completeness & miss rates, range limits in data series

Resource management
- Resource impedance mismatch
- Undocumented resource sharing
- Unmanaged shared resources

Virtualization of resources
- Resource redundancy: Logical vs. physical redundancy
- Resource isolation guarantees: Space, time, and bandwidth partitioning
- Virtualized time: Time stamping of data & asynchronous system communication
- Virtualization & mixed criticality

Distributed Discrete state behavior
- Replicated & mirrored state machines
- Coordination through state vs. state transition communication
- Consistent implementation: e.g., sampled event observation & non-deterministic sampling
Modeling an Embedded System Architecture

SAE AADL industry standard (AS5506A) supports modeling, analysis, and auto-generation of embedded system architectures.
Dimensions of System Validation

- Validation of models against system
- Model-based validation of system
- Validation of implementation against system models

The system

Multiple Analysis Truths

System models

Disconnect Between Model & Implementation

System implementation
Architecture-Centric Modeling Approach

Single Source Annotated Architecture Model

Availability & Reliability
- MTBF
- FMEA
- Hazard analysis

Security
- Intrusion
- Integrity
- Confidentiality

Data Quality
- Data precision/accuracy
- Temporal correctness
- Confidence

Resource Consumption
- Bandwidth
- CPU time
- Power consumption

Auto-generated analytical models

Real-time Performance
- Execution time/Deadline
- Deadlock/starvation
- Latency

SAE AADL is extensible & supports model annotations

MTBF FMEA Hazard analysis

Safety-Critical Systems Validation Challenges
Feiler, Nov 2009
© 2009 Carnegie Mellon University
Outline

Need for Predictable System Integration

Multi-fidelity Model-based Analysis

Validation of Implementations

Towards Architecture-Centric Engineering
AADL: The Language (www.aadl.info)

Component-based architecture model
- Component type (external spec), component implementation (blueprint)
- Application: Thread, process, data, subprogram, system, abstract
- Platform: Device, processor, memory, bus, virtual processor, virtual bus
- Typed properties, properties values include units, references into model

Component interaction: task interaction architecture, platform architecture
- Data, event & message ports & connections, synchronous call/return, shared access
- End-to-End flow specifications, blackbox flow specification

Operational modes & fault tolerant configurations
- Modes & mode transition, mode-specific properties & configurations

Modeling of large-scale systems
- Component variants & specializations, parameterized templates, packaging of AADL models

Accommodation of diverse analysis needs
- Extension mechanism (property set, sublanguage), standardized
Impact Across Quality Dimensions

**Security**
- Increased confidentiality requirement
  - Change of encryption policy

**Resource Consumption**
- Increased computational complexity
  - Increases WCET
  - Increases CPU utilization
  - Increases power consumption
  - May increase latency

**Architectural Model**
- Key exchange frequency changes
  - Message size increases
  - Increases bandwidth utilization
  - Increases power consumption

**Real-Time Performance**
- Deadlock/Starvation
- Latency
- Execution Time/Deadline
Latency Contributors

- Processing latency
- Sampling latency
- Physical signal latency

Impact of Scheduler Choice on Controller Stability
A. Cervin, Lund U., CCACSD 2006
Software-Based Latency Contributors

Execution time variation: algorithm, use of cache
Processor speed
Resource contention
Preemption
Legacy & shared variable communication
Rate group optimization
Protocol specific communication delay
Partitioned architecture
Migration of functionality
Fault tolerance strategy
Outline

Need for Predictable System Integration
Multi-fidelity Model-based Analysis
Validation of Implementations
Towards Architecture-Centric Engineering
Double Buffering

From Customer Design Document

“The 200 Hz update rate was used because the MUX data needed to be processed at twice the rate of the fastest channel to avoid a race condition. Because channel 3 operates at 100 Hz, the IO processor had to operate at 200 Hz. The race condition has been fixed by double-buffering data, but the IO processor execution rate was left at 200 Hz to reduce latency of MUX data.”

Did double buffering solved the problem or do we need to do more buffering?

\[ \alpha_\text{P} - \Omega_\text{P} \cap \alpha_\text{C} - \Omega_\text{C} \neq \emptyset \Rightarrow \text{non-deterministic sampling with race condition} \]
What is the Scheduling & Execution Behavior?

Legacy Ada tasks as “partitions”
- Are scheduled by cyclic executive
- Periodic application tasks scheduled within Ada task as cyclic executive
- Harmonic subrates: finish in frame, manual load distribution

Preemptive partition scheduling on commercial RTOS
- Oxymoron?: ARINC653 specifies static line scheduling

Dispatch by virtual timer
- Virtual timer per legacy Ada task/partition
- All partitions per processor at same rate
- Timer alignment in priority order to reduce context switches

Asynchronous set of processors
- Each processor on its own clock

The abstract partition scheduling & execution behavior
We have periodic partitions with some partitions reducing their rate to ½ to 1/3 etc. under overload conditions
Well-defined Execution Semantics

- Thread execution
- Communication timing
- Mode transition
Outline

Need for Predictable System Integration
Multi-fidelity Model-based Analysis
Validation of Implementations
Towards Architecture-Centric Engineering
Aerospace Vehicle Systems Institute

System Architecture Virtual Integration (SAVI)

This line fit is pegged at 27.5 M SLOC because the SLOC sizes for 2010-2020 are not affordable. The COCOMO II estimated costs to develop that much software is in excess of $10B

Acronyms:
- SLOC: source lines of code
- COCOMO II: Constructive Cost Model II

Airbus data source: J. P. Potocki De Montaill, “Computer Software in Civil Aircraft,” Sixth Annual Conference on Software Assurance (Compass’91), Gaithersburg, MD, June 24-27, 1991
Boeing data source: J. J. Chilenski, 2009
AADL-based Proof of Concept Demonstration

**Aircraft: (Tier 0)**

- Aircraft system: (Tier 1)
  - Engine, Landing Gear, Cockpit, ...
  - Weight, Electrical, Fuel, Hydraulics, ...

**LRU/IMA System: (Tier 2)**
- Hardware platform, software partitions
- Power, MIPS, RAM capacity & budgets
- End-to-end flow latency

**Subcontracted software subsystem: (Tier 3)**
- Tasks, periods, execution time
- Software allocation, schedulability
- Generated executables

**OEM & Subcontractor:**
- Subsystem proposal validation
- Functional integration consistency
- ARINC 429 protocol mappings

**Analysis and Demonstration**
- Propagate requirements and constraints
- Higher level model down to suppliers' lower level models
- Verification of lower level models satisfies higher level requirements and constraints

**Additional Opportunities:**
- Safety & security analysis
- Fault modeling & impact analysis
- What-if trade studies

**Multi-tier system & software architecture**

**Integrator & subcontractor virtual integration**
Cooperative System Engineering

- Application Software Runtime Architecture (task & communication)
- Physical System Architecture (interface with embedded SW/HW)
- Operational Environment (People, Use scenarios)
- Computer Platform Architecture (processors & networks)
- Hardware Components (circuits & logic) VHDL
- Application Software Components (source code) Java, UML, Simulink
- Physical Components (mechanical, electrical, heat) Modelica
- Environments
  - Control Engineering
  - Application Software Engineering
  - Electrical Engineering
  - Mechanical Engineering

- Embedded System Engineering
- System Engineering

Software Engineering Institute | Carnegie Mellon
Cost & Time Reduction due to Early Fault Discovery

Incremental virtual integration & analysis early & throughout life cycle


Where faults are introduced
Where faults are found
The estimated nominal cost for fault removal
NO WARRANTY

THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN “AS-IS” BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

Use of any trademarks in this presentation is not intended in any way to infringe on the rights of the trademark holder.

This Presentation may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

This work was created in the performance of Federal Government Contract Number FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 252.227-7013.