Virtual Integration and Incremental Assurance of Critical Systems

Peter H. Feiler

Layered Assurance Workshop 2015
Dec 8, 2015
Copyright 2015 Carnegie Mellon University

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

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.

[Distribution Statement A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

This material 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.


DM-0003120
Virtual Integration and Incremental Assurance of Critical Systems

Dec 8, 2015

© 2015 Carnegie Mellon University

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

Agenda

Challenges and Four Pillar Strategy for Critical Software Systems
Virtual System Integration
Software Hazards and Vulnerabilities
Incremental Lifecycle Assurance
We Rely on Software for Safe Aircraft Operation

Quantas Airbus A330-300 Forced to make Emergency Landing - 36 Injured
Written by hbw on Oct-15 08 1:48pm
From: soyawannaknow.blogspot.com

Thirty-six passengers and crew were injured, some seriously, in a mid-air drama that forced a Qantas jetliner to make an emergency landing, the Australian carrier and police said Tuesday.

The terrifying incident saw the Airbus A330-300 in a mayday call when it suddenly changed altitude during a flight from Singapore to Perth, Qantas said.

Oct. 15 (Bloomberg) -- Airbus SAS issued an alert to airlines after Australian investigators said a computer fault on a Qantas Ltd. flight switched off the autopilot and generated false data that led to a nose dive.

The Airbus A330-300 was cruising at 37,000 feet (11,277 meters) when an external computer fed incorrect information to the flight control systems, the Australian Transport Safety Bureau said yesterday. The plane spiraled down 650 feet within seconds, slamming passengers and crew onto the floor and ceiling, before the pilots regained control.

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

FAA says software problem with Boeing 787s could be catastrophic
By Dan Catchpole
@dcatchpole

The Federal Aviation Administration says a software problem with Boeing 787 Dreamliners could lead to one of the advanced jetliners losing electrical power in flight, which could lead to loss of control.

The FAA notified operators of the airplane Friday that if a 787 is powered continuously for 248 days, the plane will automatically shut down its alternating current (AC) electrical power.
Software Problems not just in Aircraft

Lexus GX 460 passes retest; Consumer Reports lifts "Don't Buy" label

Consumer Reports is lifting the Don't Buy: Safety Risk designation from the 2010 Lexus GX 460 SUV after recall work corrected the problem it displayed in one of our emergency handling tests. (See the original report and video, "Don't Buy: Safety Risk—2010 Lexus GX 460.")

We originally experienced the problem in a test that we use to evaluate what's called lift-off oversteer. In this test, as the vehicle is driven through a turn, the driver quickly lifts his foot off the accelerator pedal to see how the vehicle reacts. When we did this with our GX 460, its rear end slid out until the vehicle was almost sideways. Although the GX 460 has electronic stability control, which is designed to prevent a vehicle from sliding, the system wasn't intervening quickly enough to stop the slide. We consider this a safety risk because in a real-world situation this could cause a rear tire to strike a curb or slide off of the pavement, possibly causing the vehicle to roll over. Tall vehicles with a high center of gravity, such as the GX 460, heighten our concern. We are not aware, however, of any reports of injury related to this problem.

Lexus recently duplicated the problem on its own test track and developed a software upgrade for the vehicle's ESC system that would prevent the problem from happening. Dealers received the software fix last week and began notifying GX 460 owners to bring their vehicles in for repair.

We contacted the Lexus dealership from which we had anonymously bought the vehicle and made an appointment to have the recall work performed. The work took about an hour and a half.

Following that, we again put the SUV through our full series of emergency handling tests. This time, the ESC system intervened earlier and its rear did not slide out in the lift-off oversteer test. Instead, the vehicle understeered—or plowed—when it exceeded its limits of traction, which is a more common result and makes the vehicle more predictable and less likely to roll over. Overall, we did not experience any safety concerns with the corrected GX 460 in our handling tests.

How do you upgrade washing machine software?

How do you prevent your engine from cheating?
High Fault Leakage Drives Major Increase in System Cost

Aircraft industry has reached limits of affordability due to exponential growth in SW size and complexity.

Where faults are introduced
Where faults are found
The estimated nominal cost for fault removal

Requirements Engineering
System Design
Software Architectural Design
Component Software Design
Code Development
Unit Test
Integration Test
System Test
Acceptance Test

70% Requirements & system interaction errors
80% late error discovery at high rework cost
20.5% 300-1000x
0%, 9% 80x
20%, 16%
10%, 50.5% 20x
70%, 3.5% 1x
10%, 50.5% 20x
70%, 3.5% 1x

Major cost savings through rework avoidance by early discovery and correction
A $10k architecture phase correction saves $3M

Software as % of total system cost
1997: 45% → 2010: 66% → 2024: 88%

Post-unit test software rework cost 50% of total system cost and growing

Sources:
Mismatched Assumptions in System Interactions

**System Engineer**
- Hazards: Impact of system failures
- Operator Error: Automation & human actions

**Control Engineer**
- Measurement Units, value range
- Boolean/Integer abstraction
- Air Canada, Ariane, 7500 Boolean variable architecture

**System Under Control**
- Physical Plant Characteristics: Lag, proximity
- Data Stream Characteristics: Latency jitter affects control behavior, Potential event loss

**Application Developer**
- Concurrency Communication: iTunes crashes on dual-cores

**Hardware Engineer**
- Distribution & Redundancy: Virtualization, load balancing, mode confusion

**Compute Platform**
- System User/Environment

**Runtime Architecture**
- Embedded SW System Engineer

**Application Software**
- Embedded software system as major source of hazards

Why do system level failures still occur despite fault tolerance techniques being deployed in systems?
Model-based Engineering Pitfalls

The system

System models

System implementation

Inconsistency between independently developed analytical models

Confidence that model reflects implementation

This aircraft industry experience has led to the System Architecture Virtual Integration (SAVI) initiative
Awareness of Requirement Quality

Textual requirement quality statistics

- Current requirement engineering practice relies on stakeholders traceability and document reviews resulting in high rate of requirement change

### System to SW requirements gap [Boehm 2006]

**How do we trace low level SW requirements against system requirements?**

When StartUpComplete is TRUE in both FADECs and SlowStartupComplete is FALSE, the FADECStartupSW shall set SlowStartupIncomplete to TRUE
Assurance & Qualification Improvement Strategy

Assurance: Sufficient evidence that a system implementation meets system requirements

|--------------------------------------------|-------------------------------------------------|---------------------------------------------|----------------------------------------------------------|

- Mission Requirements
  - Function
  - Behavior
  - Performance
- Survivability Requirements
  - Reliability
  - Safety
  - Security

- Model Repository
  - Architecture Model
  - Component Models
  - System Implementation
  - System configuration

- Operational & failure modes
- Resource, Timing & Performance Analysis
- Reliability, Safety, Security Analysis

Early Problem Discovery through Virtual System Integration & Analysis
Improved Assurance through Better Requirements & Automated Verification
Improved Cost, Time and Quality

Reduced Cost and Time through Early Discovery

Improved Quality through Better Requirements & Evidence

Build the System

Assure the System
Agenda

Challenges and Four Pillar Strategy for Critical Software Systems
Virtual System Integration
Software Hazards and Vulnerabilities
Incremental Lifecycle Assurance
SAE Architecture Analysis & Design Language (AADL) to the Rescue

Physical system

Command & Control

Task & Communication Architecture

Deployed on

Physical interface

Distributed Computer Platform

AADL Addresses Increasing Interaction Complexity and Mismatched Assumptions
SAE AADL Standard Suite (AS-5506 series)

Core AADL language standard (V2.1-Sep 2012, V1-Nov 2004)

- Strongly typed language with well-defined semantics
- Textual and graphical notation
- Standardized XMI interchange format

**Standardized AADL Extensions**

- Error Model language for safety, reliability, security analysis
- ARINC653 extension for partitioned architectures
- Behavior Specification Language for modes and interaction behavior
- Data Modeling extension for interfacing with data models (UML, ASN.1, …)
- Guidance for runtime executive generation

**AADL Extensions in Progress**

- Requirements Definition and Assurance Language
- Synchronous System Specification Language
- Hybrid System Specification Language
- System Constraint Specification Language
AADL: The Language

Precise execution semantics for components
  • Thread, process, data, subprogram, system, processor, memory, bus, device, virtual processor, virtual bus

Continuous control & event response processing
  • Data and event flow, call/return, shared access
  • End-to-End flow specifications

Operational modes & fault tolerant configurations
  • Modes & mode transition

Modeling of large-scale systems
  • Component variants, layered system modeling, packaging, abstract, prototype, parameterized templates, arrays of components, connection patterns

Accommodation of diverse analysis needs
  • Extension mechanism, standardized extensions
System Level Problem Areas

Time-sensitive Data Stream Assumptions
- Stream miss rates, Mismatched data representation, Latency jitter & age

Partitions as Isolation Regions
- Competing demands by security and safety
- Space and time partitioning for processors and networks
- Isolation not guaranteed due to undocumented resource sharing

Virtualization of Resources
- Logical vs. physical redundancy
- Virtualization of time

Inconsistent System States & Interactions
- Modal systems with modal components
- Concurrency & redundancy management
- Application level interaction protocols

Resource guarantees
- Performance impedance mismatchess
- Unmanaged system resources

End-to-end latency analysis
Port connection consistency

Process and virtual processor to model partitioned architectures

Virtual processors & buses
Multiple time domains

Operational and failure modes
Interaction behavior specification
Dynamic reconfiguration
Fault detection, isolation, recovery

Resource allocation & deployment configurations
Resource budget analysis & scheduling analysis

Codified in Virtual Upgrade Validation method
Well-defined Execution Semantics

OMG MARTE

Focus on implementation
- Timers to trigger task execution
- Send/receive operations
- Behavioral states and transitions

SAE AADL

Focus on Architecture Abstraction
- Thread execution
- Communication timing
- Operational modes & architecture reconfiguration
Partitioned Run-Time Architecture

Application Software Component
• Timing Protection
• OS Call Restrictions
• Memory Protection

Application Software Component

Application Software Component

Application Software Component

Virtual Integration and Incremental Assurance of Critical Systems
Dec 8, 2015
© 2015 Carnegie Mellon University
Distribution Statement A: Approved for Public Release; Distribution is Unlimited

AADL Runtime System

Real-Time Operating System

Embedded Hardware Target

Strong Partitioning
• Timing Protection
• OS Call Restrictions
• Memory Protection

Interoperability/Portability
• Tailored Runtime Executive
• Standard RTOS API
• Application Components

AADL ARINC653 Annex aligned with latest ARINC653 Standard
Analysis of Virtually Integrated Software Systems

Single Annotated Architecture Model Addresses Impact Across Operational Quality Attributes

Safety & Reliability
- MTBF
- FMEA
- Hazard analysis

Security
- Intrusion
- Integrity
- Confidentiality

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

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

Resource Consumption
- Bandwidth
- CPU time
- Power consumption

Architecture Model
- Change of Encryption from 128 bit to 256 bit
- Higher CPU demand
- Increased latency

Potential new hazard
- Affects temporal correctness

Auto-generated analytical models

Affects temporal correctness

Virtual Integration and Incremental Assurance of Critical Systems
Dec 8, 2015
© 2015 Carnegie Mellon University
Incremental Multi-Tier Assurance in SAVI

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

System & SW Engineering:
- Mechatronics: Actuator & Wings
- Safety Analysis (FHA, FMEA)
- Reliability Analysis (MTTF)

OEM & Subcontractor:
- Subsystem proposal validation
- Functional integration consistency
- Data bus protocol mappings

Repeated Virtual Integration Analyses:
- Power/weight
- MIPS/RAM, Scheduling
- End-to-end latency
- Network bandwidth

Proof of Concept Demonstration and Transition by Aerospace industry initiative
- Architecture-centric model-based software and system engineering
- Architecture-centric model-based acquisition and development process
- Multi notation, multi team model repository & standardized model interchange

- Multi-tier system & software architecture (in AADL)
- Incremental end-to-end verification of system properties
Latency Sensitivity in Control Systems

Common latency data from system engineering
- 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
Incremental Latency Analysis

Latency analysis throughout life cycle
- Functional & system architecture: latency budgets
- Task & communication architecture: processing, sampling, transfer
- Platform architecture: partitions, protocols, computer hardware

Latency contributors
- Systems: processing, sampling, queuing latency
- Connections: protocol overhead, physical transfer, sampling
- Partitions: sampling, window schedule

Trade studies
- Best-case & worst-case, latency jitter
- Mid-frame and frame-delayed communication
- Synchronous and asynchronous systems
- Partition end and major frame output policy
- Empty & full queue
- Data set processing

Top-down & bottom-up

Utilizes end-to-end flows
Incremental refinement
Interprets deployment bindings
Operational mode specific analysis
## Detailed Latency Analysis Reports

<table>
<thead>
<tr>
<th>Contributor</th>
<th>Min Specific Value</th>
<th>Min Method</th>
<th>Max Specific Value</th>
<th>Max Method</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partition cpu.part1</td>
<td>0.0ms</td>
<td>partition offset</td>
<td>0.0ms</td>
<td>partition offset</td>
<td>Initial 200.0ms partition latency not added</td>
</tr>
<tr>
<td>thread s1.ts</td>
<td>0.0ms</td>
<td>first sampling</td>
<td>0.0ms</td>
<td>first sampling</td>
<td>Initial 20.0ms sampling latency not added</td>
</tr>
<tr>
<td>thread s1.ts5</td>
<td>1.0ms</td>
<td>processing time</td>
<td>2.0ms</td>
<td>processing time</td>
<td></td>
</tr>
<tr>
<td>Partition cpu.part1</td>
<td>199.0ms</td>
<td>partition output (MF)</td>
<td>198.0ms</td>
<td>partition output (MF)</td>
<td>Output at 200.0ms major frame</td>
</tr>
<tr>
<td>Connection</td>
<td>0.0ms</td>
<td>no latency</td>
<td>0.0ms</td>
<td>no latency</td>
<td></td>
</tr>
<tr>
<td>Partition cpu.part3</td>
<td>100.0ms</td>
<td>partition offset</td>
<td>100.0ms</td>
<td>partition offset</td>
<td></td>
</tr>
<tr>
<td>thread p.tf</td>
<td>0.0ms</td>
<td>sampling</td>
<td>0.0ms</td>
<td>sampling</td>
<td></td>
</tr>
<tr>
<td>thread p.tf</td>
<td>2.0ms</td>
<td>processing time</td>
<td>3.0ms</td>
<td>processing time</td>
<td></td>
</tr>
<tr>
<td>Partition cpu.part3</td>
<td>98.0ms</td>
<td>partition output (MF)</td>
<td>97.0ms</td>
<td>partition output (MF)</td>
<td>Output at 200.0ms major frame</td>
</tr>
<tr>
<td>Connection</td>
<td>0.0ms</td>
<td>no latency</td>
<td>0.0ms</td>
<td>no latency</td>
<td></td>
</tr>
<tr>
<td>Partition cpu.part4</td>
<td>150.0ms</td>
<td>partition offset</td>
<td>150.0ms</td>
<td>partition offset</td>
<td></td>
</tr>
<tr>
<td>thread a.tc</td>
<td>0.0ms</td>
<td>sampling</td>
<td>0.0ms</td>
<td>sampling</td>
<td></td>
</tr>
<tr>
<td>thread a.tc</td>
<td>1.0ms</td>
<td>processing time</td>
<td>3.0ms</td>
<td>processing time</td>
<td></td>
</tr>
<tr>
<td>Immediate Connection</td>
<td>0.0ms</td>
<td>no latency</td>
<td>0.0ms</td>
<td>no latency</td>
<td></td>
</tr>
<tr>
<td>thread a.td</td>
<td>0.0ms</td>
<td>no latency</td>
<td>0.0ms</td>
<td>no latency</td>
<td></td>
</tr>
<tr>
<td>thread a.td</td>
<td>1.0ms</td>
<td>processing time</td>
<td>2.0ms</td>
<td>processing time</td>
<td></td>
</tr>
<tr>
<td>Latency Total</td>
<td>0.0ms</td>
<td>552.0ms</td>
<td>0.0ms</td>
<td>555.0ms</td>
<td></td>
</tr>
<tr>
<td>End to End Latency</td>
<td>20.0ms</td>
<td>30.0ms</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### End to end Latency Summary

**ERROR** Minimum actual latency total 552.0 ms exceeds expected maximum end to end latency 30.0ms

**ERROR** Maximum actual latency 555.0ms exceeds expected end to end latency 30.0ms
Finding Problems Early

**Issue:** Contractor could not assess integration risk early enough.

**Action:** 6 Week Virtual Integration identified 20 major issues.

**Result:** Adjusted CDR Schedule to remediate.

- Prevented 12 month delay in a 2 year project.

The current method would not have identified the issues until 3 months before delivery.

**System Architecture Virtual Integration (SAVI) 2008-**
Proof of concept with AADL led to ten year commitment

**SAVI ROI Study (2009/10)**
$2B savings on $10B aircraft through 33% early detection

**Architecture-centric Virtual Integration Practice (ACVIP)**

2014/15 Virtual Integration Shadow led to early discovery of 85+ potential integration issues

Led to acceleration of adoption by JMR contractors and inclusion in RFP for FY16/17 projects
Towards an Architecture-Centric Virtual Integration Practice (ACVIP)
Agenda

Challenges and Four Pillar Strategy for Critical Software Systems
Virtual System Integration
Software Hazards and Vulnerabilities
Incremental Lifecycle Assurance
Safety Practice in Development Process Context

Leveson (MIT) Socio-technical Control Framework based on Rasmussen (NASA) model of risk management

Multiple hazard contributors in development and operational context

Safety analysis automation through virtual system integration allows for use at all layers and throughout life cycle.

Labor-intensive
Early in system engineering
Rarely repeated due to cost

Focus on System Engineering Largely Ignores Software as Hazard Source
AADL Error Model Scope and Purpose

System safety process uses many individual methods and analyses, e.g.
- hazard analysis
- failure modes and effects analysis
- fault trees
- Markov processes

Related analyses are also useful for other purposes, e.g.
- maintainability
- availability
- Integrity

Goal: a general facility for modeling fault/error/failure behaviors that can be used for several modeling abstractions and analyses.

Annotated architecture model permits checking for consistency and completeness between these abstractions.

SAE ARP 4761 Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment
Error Model V2: Abstraction and Refinement

Four levels of abstraction:

- Focus on fault interaction with other components
  - Probabilistic error sources, sinks, paths and transformations
  - Fault propagation and Transformation Calculus (FPTC) from York U.
- Focus on fault behavior of components
  - Probabilistic typed error events, error states, propagations
  - Voting logic, error detection, recovery, repair
- Focus on fault behavior in terms of subcomponent fault behaviors
  - Composite error behavior state logic maps states of parts into (abstracted) states of composite
- Types of malfunctions and propagations
  - Common fault ontology
  - User definable types
Fault Propagation Taxonomy and Guide Words

Key Taxonomy Terms
- Service: Omission/Commission
- Timing: Early/late arrival
- Value: Out of range/incorrect value
- Stream: High/low/variable rate
- Replication: Asymmetric value error
- Concurrency: race condition
- Authentication/authorization

Guide Words
- Missing sensor reading/command
- Loss of power
- Early/late command, feedback delay
- Inaccurate measures
- Command value out of range
- Commanded volume too high

error types
- SensorDataOmission: type extends ErrorLibrary::ItemOmission;
- SensorDataOutOfRange: type extends ErrorLibrary::OutOfRange;
- SensorDataClippedHigh: type extends ErrorLibrary::OutOfBounds;
- SensorDataClippedLow: type extends ErrorLibrary::OutOfBounds;
- SensorDataBiasedHigh: type extends ErrorLibrary::SubtleValueError;
- SensorDataBiasedLow: type extends ErrorLibrary::SubtleValueError;
- SensorDataDrifting: type extends ErrorLibrary::SubtleValueError;
Consistency in Error Propagation

Present and absent outgoing and incoming error propagations
Error sources, paths, and sinks (FPTC)
Connections as error sources

Mismatched fault propagation and containment assumptions
Discovery of unhandled error propagations.
Software Induced Flight Safety Issue

Original Preliminary System Safety Analysis (PSSA)
System engineering activity with focus on failing components.
Unhandled Hazard Discovery through Virtual Integration

Virtual integration of architecture fault models recording SIL test observations detects unhandled fault.

- Flight Mgmt System
  - Auto Pilot
    - Operational
    - Failed
  - FMS Processor
    - Operational
    - Failed

- EGI Logic
  - Oper'l
  - Failed
  - Corrupted

- EGI HW
  - Oper'l
  - Failed

Corrupted data shows airspeed of 2000 knots.

Vibration causes data corruption through touching boards.

Response to corrupted airspeed causes stall.
Automation of SAE ARP4761 System Safety Assessment Practice

**FHA**
- Spreadsheet
- Uses error sources

**AADL & EMV2**

**FMEA**
- Spreadsheet
- Uses error flows & propagations

**FTA**
- CAFTA, OpenFTA
- Uses composite error behavior

**Markov Chain**
- PRISM
- Uses error flows & behavior

**RBD/DD**
- OSATE plugin
- Uses composite error behavior

<table>
<thead>
<tr>
<th>Component</th>
<th>Error</th>
<th>Hazard Description</th>
<th>Class/refer</th>
<th>Functional Failure</th>
<th>Operational</th>
</tr>
</thead>
<tbody>
<tr>
<td>StabilatorPositionSet</td>
<td>Service/omission</td>
<td>Failure to move stabilator into desired position</td>
<td>1.1.1</td>
<td>Loss of actuator functionaity</td>
<td>All</td>
</tr>
<tr>
<td>StabilatorActuator1</td>
<td>Service/omission</td>
<td>Failure to move stabilator into desired position</td>
<td>1.1.2</td>
<td>Loss of actuator functionaity</td>
<td>All</td>
</tr>
<tr>
<td>StabilatorActuator2</td>
<td>Service/omission</td>
<td>Failure to move stabilator into desired position</td>
<td>1.1.3</td>
<td>Loss of actuator functionaity</td>
<td>All</td>
</tr>
<tr>
<td>StabilatorController1</td>
<td>Null on Act/End</td>
<td>Absence of computed data should signs</td>
<td>1.1.1</td>
<td>Loss of guidance values</td>
<td>Approach</td>
</tr>
<tr>
<td>StabilatorController2</td>
<td>Null on Act/End</td>
<td>Absence of computed data should signs</td>
<td>1.1.2</td>
<td>Loss of guidance values</td>
<td>Approach</td>
</tr>
<tr>
<td>StabilatorController3</td>
<td>Null on Act/End</td>
<td>Absence of computed data should signs</td>
<td>1.1.3</td>
<td>Loss of guidance values</td>
<td>Approach</td>
</tr>
</tbody>
</table>

Virtual Integration and Incremental Assurance of Critical Systems
Dec 8, 2015
© 2015 Carnegie Mellon University
Automated FMEA Experience

Failure Modes and Effects Analyses are rigorous and comprehensive reliability and safety design evaluations

- Required by industry standards and Government policies
- When performed manually are usually done once due to cost and schedule
- If automated allows for
  - multiple iterations from conceptual to detailed design
  - Tradeoff studies and evaluation of alternatives
  - Early identification of potential problems

Largest analysis of satellite to date consists of 26,000 failure modes

- Includes detailed model of satellite bus
- 20 states perform failure mode
- Longest failure mode sequences have 25 transitions (i.e., 25 effects)
Contract-based Compositional Verification

Secure Mathematically-Assured Composition of Control Models

**Key Problem**

*Many vulnerabilities occur at component interfaces. How can we use formal methods to detect these vulnerabilities and build provably secure systems?*

**Technical Approach**

- Develop a complete, formal architecture model for UAVs that provides robustness against cyber attack
- Develop compositional verification tools driven from the architecture model for combining formal evidence from multiple sources, components, and subsystems
- Develop synthesis tools to generate flight software for UAVs directly from the architecture model, verified components, and verified operation system

**Accomplishments**

- Created AADL model of vehicle hardware & software architecture
- Identified system-level requirements to be verified based on input from Red Team evaluations
- Developed Resolute analysis tool for capturing and evaluating assurance case arguments linked to AADL model
- Developed example assurance cases for two security requirements
- Developed synthesis tool for auto-generation of configuration data and glue code for OS and platform hardware

**Increasing Proactive Architecture-focused Security Research**

16 months into the project

Draper Labs could not hack into the system in 6 weeks

Had access to source code

Open source tools available at github.com/smaccm
Agenda

Challenges and Four Pillar Strategy for Critical Software Systems

Virtual System Integration

Software Hazards and Vulnerabilities

Incremental Lifecycle Assurance
Incremental Lifecycle Assurance Objectives

Measurably improve critical system assurance through

• Better requirement coverage and managed uncertainty
• Incremental analytical verification throughout lifecycle
• Focus on high payoff areas
Requirements & Architecture Design Constraints

Textual Requirements for a Patient Therapy System

1. The patient shall never be infused with a single air bubble more than 5ml volume.
2. When a single air bubble more than 5ml volume is detected, the system shall stop infusion within 0.2 seconds.
3. When piston stop is received, the system shall stop piston movement within 0.01 seconds.
4. The system shall always stop the piston at the bottom or top of the chamber.

Same Requirements Mapped to an Architecture Model

Importance of understanding system boundary

We have effectively specified a system partial architecture

U Minnesota Study
Awareness of Requirement Change Uncertainty

Managed awareness of requirement uncertainty reduces requirement changes by 50%

- 80% of requirement changes from development team
- Expert assessment of change uncertainty
- Focus on high uncertainty and high importance areas
- Engineer for inherent variability

Rolls Royce Study
Three Dimensions of Incremental Assurance

Incremental assurance through virtual system integration for early discovery

Early Discovery leads to Rework Reduction

Priority focused architecture design exploration for high payoff

Contract-based Compositional Verification

Auto-generated Assurance Cases
Three Dimensions of Requirement Coverage

System interactions, state, behavior

- Constraints/Controls
- Environment
- Behavior
- Output
- State
- Input
- Resources

Guarantees
Assumptions

Invariants

Implementation constraints
Exceptional conditions

Fault impact & contributors

Fault Propagation Ontology

- Omission errors
- Commission errors
- Value errors
- Sequence errors
- Timing errors
- Replication errors
- Rate errors
- Concurrency errors
- Authentication errors
- Authorization errors

Design & operational quality attributes

- Performance
- Data Latency: Reduce storage latency on customer DB to < 200 ms.
- Transaction Throughput: Deliver video in real time.

- Modifiability
- New products: Add CORBA middleware in < 20 person-months.
- Change COTS: Change Web user interface in < 4 person-weeks.

- Availability
- H/W failure: Power outage at site1 requires traffic redirected to site2 in < 3 seconds.
- COTS S/W failures: Network failure detected and recovered in < 1.5 minutes.

- Security
- Data confidentiality: Credit card transactions are secure 99.999% of the time.

Leveson STAMP pattern
Automated Incremental Assurance Workbench

Identify Assurance Hotspots throughout Lifecycle

Tier 0

Model

Ver Plan

Req

Tier 1

Model+1

Ver Plan

Req+1

Code

Tier 2

Model+2

Model+2'

Ver Plan

Req+2

Assurance Case

Stakeholder Goals

High Abstraction

Abstraction Level

Low Level

Close to Implementation

Virtual Integration and Incremental Assurance of Critical Systems

Dec 8, 2015

© 2015 Carnegie Mellon University

Software Engineering Institute | Carnegie Mellon University
Integration of Concepts/Meta models

RDAL
- Hazards Obstacles
- Goals
- Verification Methods
- Verification Activities
- Resolute
- Agree

GORE/KAOS
- Requirements
- Architecture specification
- Assurance
- Claims & Arguments
- Assurance Results

AADL
- Architecture instance

EMV2

SACM

SVM
Assurance Automation and Metrics

Assurance Automation

• Compositional verification plans, priority-focused assurance case configurations
• Multi-valued verification results (success, fail, timeout, error)

Measurement based assurance hotspot identification throughout lifecycle

• Requirement coverage measures
• Weighted claims, verification methods & results
• Priority measures: Change uncertainty, safety/security risk and esign assurance levels

- System AircraftTier2: (S93 F1 T0 E1 tbd0 EL0 T50)
  - ✓ ClaimR1: The weight of the Aircraft system shall not exceed 70000.0kg (S2 F0 T0 E0 tbd0 EL0 T50)
  - ✓ System ELE: (S1 F1 T0 E0 tbd0 EL0 T50)
  - ✓ ClaimR1: The weight of the Electrical system shall not exceed 75.0kg (S1 F0 T0 E0 tbd0 EL0 T50)
  - ✓ ClaimR2: The Electrical System shall be capable of handling at least 24000.0W (S0 F1 T0 E0 tbd0 EL0 T50)
  - ✓ Evidence power GAS: Analyze Electrical power demands against supply and capacity. This method is performed.
  - ✓ System ELE: ** ELE power budget total 24500.0 W exceeds capacity 24000.0 W
  - ✓ System ELE: budget total 24500.0 W within supply 25000.0 W
- System FGS: (S90 F0 T0 E1 tbd0 EL0 T50)
  - ✓ ClaimR1: The FGS shall weigh no more than 300.0kg (S0 F0 T0 E1 tbd0 EL0 T50)
  - ✓ Evidence weight limit: Perform full weight (mass) analysis. This includes net/gross weight consistency, weight bud
  - ✓ System FGS: [G] Sum of weights 76.300 kg less than gross weight 280.000 kg (using gross weight)
  - ✓ System FGS: [A] Sum of weights 280.000 kg below weight limit 300.000 kg (6.7 % Weight slack)
  - ✓ ClaimR2: The FGS shall not draw in excess of 2000.0W (S1 F0 T0 E0 tbd0 EL0 T50)
  - ✓ ClaimR3: The FGS shall be capable of processing at least 1100.0MIPS (S1 F0 T0 E0 tbd0 EL0 T50)
  - ✓ ClaimR4_1: The RAM memory needs of the FGS shall be no more than 80 percent of 2048.0MByte (S1 F0 T0 E0 tbd}
Time-sensitive Auto-brake Mode Confusion

SAVI case study
- Wheel braking system from SAE AIR6110 & real accidents
- Can software related hazards be identified analytically?

Auto-brake mode selection by push button
- Three buttons for three modes

Event sampling in asynchronous system setting
- Dual channel COM/MON architecture
- Each COM, MON unit samples separately
  - Button push close to sampling rate results in asymmetric value error
  - Repeated button push does not correct problem
  - Operational work around (1 second push) is not fool proof

Avoidable complexity design issue
- Concept mismatch: desired state as flag that is sampled
- Better solution: State communication by multi-position switch
Diagnostics and Redesign Assurance

Stepper motor (SM) controls a valve

- Commanded to achieve a specified valve position
  - Fixed position range mapped into units of SM steps
- New target positions can arrive at any time
  - SM immediately responds to the new desired position

Safety hazard due to software design

- Execution time variation results in missed steps
- Leads to misaligned stepper motor position and control system states
- Sensor feedback not granular enough to detect individual step misses

Software modeled and verified in SCADE
Full reliance on SCADE of SM & all functionality
Problems with missing steps not detected

Software tests did not discover the issue
Time sensitive systems are hard to test for.

Two Customer Proposed Solutions
Sending of data at 12ms offset from dispatch
Buffering of command by SM interface
No analytical confidence that the problem will be addressed

We utilized execution and communication timing semantics of AADL model, Error Model annex as diagnostic tool, and auto-generated assurance case from verification evidence
Analysis Results and Solution

Architecture Fault Model Analysis

- Fault impact analysis identifies **multiple sources** of missed steps
  - Early arrival of step increment commands
  - Step increment command rate mismatch
  - Transient message corruption or loss
- Understanding of error cause
  - Guaranteed delivery and synchronous system assumptions
  - Quantifying too early condition
- Complexity in architecture design
  - Impact of avoidable complexity
  - Communication of state change vs. state
Agenda

Challenges in Software Reliant Systems

Four Pillar Improvement Strategy

Virtual System Integration

Incremental Lifecycle Assurance
Conclusion

Virtual system integration and analysis leads to early detection, error leakage reduction, and cost reduction

Incremental lifecycle assurance leads to reduced risk and increased confidence

Opportunity to make (re)certification more affordable through automation and priority-focused design and verification
References

AADL Website www.aadl.info and AADL Wiki www.aadl.info/wiki
Blog entries and podcasts on AADL at www.sei.cmu.edu
AADL Book in SEI Series of Addison-Wesley
On AADL and Model-based Engineering
http://www.sei.cmu.edu/library/assets/ResearchandTechnology_AADLandMBE.pdf
On an architecture-centric virtual integration practice and SAVI
http://www.sei.cmu.edu/architecture/research/model-based-engineering/virtual_system_integration.cfm
On an a four pillar improvement strategy for software system verification and qualification
Contact Information

Peter H. Feiler
Principal Researcher
RTSS
Telephone: +1 412-268-7790
Email: phf@sei.cmu.edu

Web
Wiki.sei.cmu.edu/aadl
www.aadl.info

U.S. Mail
Software Engineering Institute
Customer Relations
4500 Fifth Avenue
Pittsburgh, PA 15213-2612
USA

Customer Relations
Email: info@sei.cmu.edu
SEI Phone: +1 412-268-5800
SEI Fax: +1 412-268-6257