Safety-Critical Embedded Systems Development Issues & Cost Impact

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Peter H Feiler
April 2009
Outline

Need for Virtual System Integration
Analytical models with well-defined semantics
Cost impact & to-be business process
Airbus Gives Alert as Autopilot Caused Plane's Plunge (Update3)

By Ed Johnson

Oct. 15 (Bloomberg) -- Airbus SAS issued an alert to airlines worldwide after Australian investigators said a computer fault on a Qantas Airways Ltd. flight switched off the autopilot and generated false data, causing the jet to nosedive.

The Airbus A330-300 was cruising at 37,000 feet (11,277 meters) when the computer fed incorrect information to the flight control system, the Australian Transport Safety Bureau said yesterday. The aircraft dropped 550 feet within seconds, slamming passengers and crew into the cabin 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 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.''

Autopilot Off

A preliminary analysis of the Qantas flight showed the error occurred in one of the jet's three air-data inertial reference units, which caused the autopilot to disconnect, the ATSB said in a statement on its Web site.

The crew flew the aircraft manually to the end of the flight, except for a period of a few seconds, the bureau said.

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.

The flight control computer then commanded a nose-down aircraft movement, which resulted in the aircraft pitching down to a maximum of about 9.5 degrees, it said.

No Similar Event

``Airbus has advised that it is not aware of any similar event over the many years of operation of the Airbus,'' the bureau added, saying it will continue investigating.
Mismatched Assumptions

System Engineer

- Physical Plant Characteristics
  - System mass, heat, ...

- Plant & Environment Observations
  - System lag & response rate

- Electr. subsys
- A/C subsys
- Fuel subsys
- Electr controller
- A/C controller
- Fuel controller

System Under Control

Control Engineer

- Physical system modeling

Hybrid system control

Control System

Feiler, April 2009
© 2008 Carnegie Mellon University
Mismatched Assumptions

Why do system level failures still occur despite fault tolerance techniques being deployed in systems?
System Level Fault Root Causes

Violation of data stream assumptions
• Stream miss rates, Mismatched data representation, Latency jitter & age

Partitions as Isolation Regions
• Space, time, and bandwidth partitioning
• Isolation not guaranteed due to undocumented resource sharing
• Fault containment, security levels, safety levels, distribution

Virtualization of time & resources
• Logical vs. physical redundancy
• Time stamping of data & asynchronous systems

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

Performance impedance mismatches
• Processor, memory & network resources
• Compositional & replacement performance mismatches
• Unmanaged computer system resources

End-to-end latency analysis
Port connection consistency

Partitioned architecture models
Model compliance

Virtual processors & busses
Synchronization domains

Fault propagation
Security analysis
Architectural redundancy patterns

Resource budget analysis
& task roll-up analysis
Resource allocation & deployment configurations
Modeling an Embedded System Architecture

Elements of an embedded system architecture

- Software Architecture (task & communication) PLUS
- Hardware Architecture (relevant to embedded SW) PLUS
- Physical system/environment (relevant to embedded SW/HW) PLUS
- Logical interface between software and physical system PLUS
- Physical interface between hardware and physical system PLUS
- Deployment of software on hardware

SAE AADL supports modeling, analysis, and auto-generation of embedded system architectures.
Potential Model-based Engineering Pitfalls

Issues

- Late use of MBE
- Inconsistency between independently developed analytical models
- Lack of confidence that model reflects implementation

Solution

- Early & continuous use of MBE
- Architecture-centric model repository
- Generation from validated models

System models

System implementation

Software Engineering Institute  Carnegie Mellon

Safety-Critical Systems Validation
Feller, April 2009
© 2008 Carnegie Mellon University
Outline

Need for Virtual System Integration

**Analytical models with well-defined semantics**

Cost impact & to-be business process
Latency Contributors

- Processing latency
- Sampling latency
- Physical signal latency
Flow Use Scenario through Subsystem Architecture

Display -> IOProcessor ->
Command -> Comm -> Nav ->
IOProcessor -> Modem ->
IOProcessor -> Nav -> Comm ->
Command -> Display

Latency = Partition hops + processing + transfer
Independent clock per processor

Multiple rates and processors with independent clocks
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
Well-defined Execution & Communication Semantics in AADL

Analysis tools have to interpret clocks & timers to determine execution model.

AADL Execution & Communication Model

- Thread execution
- Communication timing
- Mode transition

Abstracted into AADL execution & communication semantics.
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
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?
Application-based Send and Receive (ASR)

$(\tau_P | \tau_C)^*$

3 buffers for ICO guarantee

$T_P \leq \alpha_P \leq S \leq \Omega_P \leq D_P$

$T_C \leq \alpha_C \leq R \leq \Omega_C \leq D_C$

$\alpha :$ actual execution start time

$\Omega :$ actual completion time

$\alpha_P - \Omega_P \cap \alpha_C - \Omega_C \neq \emptyset \Rightarrow$ non-deterministic sampling (S/R) order
Performance Improvement Gone Bad

A real customer experience

Ground station to accommodate sensor load growth
- Reduce load in network
- Two subsystems communicate state change instead of state

The impact
- Other subsystems increase network load sporadically
- Receiving subsystem goes down

The root cause
- Transmission protocol without guaranteed delivery
- Overload result in dropping of transmitted packets (state changes)
- Missing state changes result in inconsistent receiver state

Communication of state changes requires guaranteed & ordered delivery
Redundant Flight Guidance System

A Methodology for the Design and Verification of Globally Asynchronous/Locally Synchronous Architectures

Steven P. Miller and Mike W. Winslow
Rockwell Collins, Inc., Cedar Rapids, Indiana
Dan O’Brien, Matt P. Harwood, and Anjali Jhutis
University of Minnesota, Minneapolis, Minnesota

To validate:
1) At least one output
2) Exactly one output
3) Two outputs in critical mode

Increased complexity of property

Implementation samples state variables across processors

Potential button push misses discovered through AADL-based analysis
Dealing with Time in Model Checking

Active research for asynchronous systems
Verimag in ASSERT
Feasible with AADL
Outline

Need for Virtual System Integration
Analytical models with well-defined semantics
Cost impact & to-be business process
Benefits of Virtual Integration

- Predictive
  - Sensitivity analysis for uncertainty
    - Top-Level Verification Items
    - High-level AADL Model
    - Detailed AADL Model
    - Specify Model-Code Interfaces
    - Component Software Design
    - Software Architectural Design
    - System Design
    - Requirements Engineering

- Validated
  - Confidence in implementation
    - Code Development
    - Unit Test
    - Integration Test
    - System Test
    - Acceptance Test

→ generation of test cases
← updating models with actual data

Model-driven artifact generation and conformance of models and systems
Cost & Time Reduction due to Early Fault Discovery

Where faults are introduced

Where faults are found

The estimated nominal cost for fault removal

**Modified Business Model**

- System Integrator defines a new product using internal repository of virtual “parts”
- Specifications for virtual subcomponents sent to suppliers

---

**Standard Embedded Systems Architecture Interchange Format**
AADL XMI
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.