Critical Avionics Software

Edited and presented by
Claire J. Tomlin
UC Berkeley

Chess Review
November 21, 2005
Berkeley, CA
Outline

• A viewpoint from production military systems [David Sharp, Boeing Phantom Works]
• System development and certification
  - DO 178 B and C
• High level design examples:
  - Collision avoidance systems
  - Operating envelope protection
• Tools for modeling, design, and code generation
• NITRD HCSS National Workshop on Software for Critical Aviation Systems
A Viewpoint from Production Military Aircraft

- Technology Trends in Avionics Systems Are Driving Exponential Growth in Software Complexity
  - Autonomous systems, adaptive systems...
- Traditional Approaches and Processes Are Already Stressed
  - Program-specific architectures, languages, tools
  - Unaligned with commercial practices

Current Technology, Practices and Culture of the Industry Cannot Meet Emerging System Needs

David Sharp, Boeing Phantom Works, HSCC Plenary Talk, Stanford, March 2002

"Critical Avionics Software", C. Tomlin
Example: Fighter Avionics Domains

David Sharp, Boeing Phantom Works, HSCC Plenary Talk, Stanford, March 2002

"Critical Avionics Software", C. Tomlin

Chess Review, Nov. 21, 2005
Mission Computing: Example Functionality

- Update Steering Cues
- Fuse Targets From Sensors
- Fuse Targets From Data Links
- Mission Computing
- Update Navigation State
- Perform Built-In-Test
- Activate Backup Mode
- Release Weapons
- Select Weapons
- Predict Selected Weapon Trajectories
- Update Displays
- Modify Display Suite Via Pilot Pushbutton
- Aperiodic
- Periodic

David Sharp, Boeing Phantom Works, HSCC Plenary Talk, Stanford, March 2002
"Critical Avionics Software", C. Tomlin

Chess Review, Nov. 21, 2005
Vehicle Management: Example Functionality

- Perform Initiated Built-In-Test
- Perform Periodic Built-In-Test
- Manage Redundancy
- Perform Input Signal Mgmt
- Perform Actuator Signal Mgmt
- Compute Inner Loop Controls
- Compute Outer Loop Controls
- Update Navigation State
- Manage Control Modes

David Sharp, Boeing Phantom Works, HSCC Plenary Talk, Stanford, March 2002
"Critical Avionics Software", C. Tomlin

Chess Review, Nov. 21, 2005
Typical Mission Computing Legacy Characteristics

- 10-100 Hz Update Rates
- Up To 10-100 Processors
- ~1M Lines of Code
  - $O(10^3)$ Components
- Proprietary Hardware
  - Slow CPU, small memory
  - Fast I/O
- Test-Based Verification
- Mil-Std Assembly Language
- Highly Optimized For Throughput and Memory

- Functional Architectures
  - Flowchart designs
- Frequently No Maintained Requirements or Design
  - Ad-hoc models used by algorithm developers
- Hardcoded Hardware Specific Single System Designs
- Isolated Use Of
  - Multi-processing
  - Schedulability analysis
    - Frequently overly pessimistic to be used

David Sharp, Boeing Phantom Works, HSCC Plenary Talk, Stanford, March 2002
"Critical Avionics Software", C. Tomlin
Typical Vehicle Management Legacy Characteristics

Additional Characteristics

- 80/160 Hz Update Rates
- Single CPU System/Quad Redundant
- Dual/Quad Redundant Sensors and Actuators
- <100K Lines of Code
- Extensive Built-In-Test
  - >50% of code
- Extensive Testing
  - Very conservative development culture
  - >50% of effort
- Control System Models Carefully Developed And Used
  - Home grown
  - Matlab/MatrixX with auto code generation

David Sharp, Boeing Phantom Works, HSCC Plenary Talk, Stanford, March 2002
System Development and Certification

Model V&V
- Control Power V&V
- Control Law V&V
- Functional V&V

Software V&V
- Unit/Component Test
- Hardware/Software Integration (HSI)

Hardware V&V
- Qualification Test (Safety of Flight)
- Aircraft Integration

System V&V
- Standalone (Static)
- Integrated (Dynamic)
- Failure Modes and Effects Test (FMET)

[Source: Jim Buffington, LM Aero]

"Critical Avionics Software", C. Tomlin
Project management, risk mitigation, design and testing activities for embedded software developed for the commercial avionics industry are based on the FAA standard:

**RTCA (Radio Technical Commission for Aeronautics) DO-178B:** “Software Considerations in Airborne Systems and Equipment Certification”

- “Process-based” certification
- Interesting points:
  - Certification applies to the end product (ie. airframe), encompassing all systems
  - It applies to a given application of a given product (other applications of the same product require further certification)
  - It requires that all code MUST be there as a direct result of a requirement
  - It requires full testing of the system and all component parts (including the software) on the target platform and in the target environment in which it is to be deployed
DO-178 History

• Timeline History
  - Nov. 1981- DO-178-SC145
  - Mar. 1985- DO-178A -SC152 (4 years)
    • Software Levels 1,2,3 - Crit, Essential, NonEss
    • Software Develop Steps D1-D5
    • Software Verification Steps V1-V7
  - Dec. 1992- DO-178B -SC167 (7 years)
    • Objectives Based Tables
      - What, not how
    • Criticality Categories (A,B,C,D) / Objectives Matrix
  - 12 years Since DO-178B ➔ (15 years)

[source: Jim Krodel, Pratt & Whitney]
Issues Under Consideration for SC205 Sub-groups

• Technology/Domains Under Consideration
  - Formal Methods
  - Model Based Design & Verification
    • Model Verification and Level of Pedigree
    • Certification of Proof by Models
  - Software Tools
    • And our reliance on them from a certification perspective
  - Object Oriented Technology
  - Comms-Nav-Sur/Air-Traffic-Management

[source: Jim Krodel, Pratt & Whitney]
Example 1: Collision Avoidance Systems

Differential game formulation:
Compute the set of states for which, for all possible maneuvers (d) of the red aircraft, there is a control action (u) of the blue aircraft which keeps the two aircraft separated.

http://www.cs.ubc.ca/~mitchell/ToolboxLS/

[Tomlin lab, 2002]
User Interaction with Aerospace Systems:

• Interaction between
  - System's dynamics
  - Mode logic
  - User's actions

• Interface is a reduced representation of a more complex system

• Too much information overwhelms the user

• Too little can cause confusion
  - Automation surprises
  - Nondeterminism

For complex, highly automated, safety-critical systems, in which provably safe operation is paramount,

What information does the user need to safely interact with the automated system?
Example 2: Operating Envelope Protection

- Controllable flight envelopes for landing and Take Off / Go Around (TOGA) maneuvers may not be the same.
- Pilot's cockpit display may not contain sufficient information to distinguish whether TOGA can be initiated.

[Tomlin lab, 2003]

http://www.cs.ubc.ca/~mitchell/ToolboxLS/
"Critical Avionics Software", C. Tomlin
Tools for modeling, design, and code generation

Designing safety critical control systems requires a seamless cooperation of tools:

- Modeling and design at the control level
- Development tools at the software level
- Implementation tools at the platform level

- Corresponding research needed:
  - Development of algorithms and tools to verify and validate the high level design - currently tools such as reachability analysis tools for hybrid systems are limited to work in up to 4-5 continuous state dimensions
  - Development of code generation tools (ideally, verified to produce correct code)
  - Tools to check the correctness of the resulting code
  - Algorithms and tools to automatically generate test suites
Static Program Analysis Tools

**Static program analysis** is used at compile time to automatically determine run-time information and properties which are extractable from the source code. These include:

- Ensuring that the allowable range of array indexes is not violated
- Ensuring simple correctness properties: functional (such as dependencies between aspects of variables or invariants on the shape of data structures) or nonfunctional (such as confidentiality or integrity for security-critical applications)
- Identifying potential errors in memory access
- Type checking
- Interval analysis
- Checking for illegal operations, like division by zero

Currently, properties such as absence of run time errors and worst case execution time have been tackled; more research is needed to address problems arising from a distributed, embedded setting, such as checking for safety conditions, and for the absence of deadlocks

"Critical Avionics Software", C. Tomlin
NITRD HCSS National Workshop on Software for Critical Aviation Systems

- Workshop co-chairs: Tomlin and Hansman
- NITRD HCSS Co-Chair: Helen Gill
- Planning meeting: University of Washington, Nov 9-10 (~35 participants from Industry, DoD, Govt, and Academia)
- Workshop, June 2006, Washington DC
- Application domains:
  - Air traffic management, C&C
  - Flight control, UAVs
  - CNS, aircraft and infrastructure integration
  - Satellite and space system control

NITRD = Federal Networking and Information Technology Research and Development
HCSS = High Confidence Software and Systems

"Critical Avionics Software", C. Tomlin
Issues:

• Reduce software development time and costs for next generation avionics platforms
  - Distributed systems
  - Adaptive systems
  - Mixed criticality systems
  - Human in the loop
  - Security in the loop

• Design for certification
• Design for re-use
• Minimize re-test
• Open experimental platforms: high pedigree models for application of technologies