Formal Methods at Airbus: Experience Feedback

Presented by
Jean Souyris / EYYWDV – Verification and dependability support
Outline

• Introduction
• Context, Objectives and Constraints
• Basics
• Industrial state-of-the-use
• Next transfers
• Main issues: solved & remaining
• Conclusion
Outline

• Introduction
• **Context, Objectives and Constraints**
• Basics
• Industrial state-of-the-use
• Next transfers
• Main issues: solved & remaining
• Conclusion
Context: avionics software products

• **Avionics domains**
  - **Flight controls**: safety-critical (DAL A), time-critical, SCADE specification (synchronous paradigm), no operating system, floating-point calculus, and also non SCADE “driver-like” functions
  - **Flight Warning**: medium criticality (DAL C), asynchronous (multi-tasks) functions running on IMA platform, complex data structures (non dynamic allocation)
  - **Board/ground communication**: medium criticality (DAL C), asynchronous (multi-tasks) functions running on IMA or POSIX platforms, complex data structures (no dynamic allocation)
  - **Maintenance functions**: low criticality (DAL D & E), asynchronous (multi-tasks) functions running on POSIX OS

• **Verification environment**
  - **SIMUGENE**: hardware virtualization for verification by execution
Context: elements of development organisation

**• Avionics software development teams**
  • Specify, design, code and verify software products from system specifications
  • In conformance with Airbus’s reference development processes and methods, thus with DO-178B

**• Support teams** (specification, design, verification, configuration management, modification management)
  • Strategies
  • Operational support Methods and tools (including training)
  • (new) Service activities on behalf of development teams

**• Process and assurance teams (“Quality”)**
  • Process definition
  • Check the conformance with reference process and DO-178B
Avionics software development Process

- System Specification
  - Software Specification
    - Design architecture
      - Automatic Coding Executable generation
      - Static design LL requirements
        - Coding Executable generation
  - Unit Verification
    - Integration Verification
      - Validation level checks
        - System Integration
          - Verification from “Checklist”
          - Integration with other subsets
Context / Objectives for formal tools

- **Steady increase of System complexity**
  - Master verification costs
  - Performance: contribute to the safe and optimal use of modern hardware and software features
  - Keep computation safety (executability) verification at high level

- **Need for early maturity**
  - Exhaustive verification techniques
  - Available as soon system design / code is available

- **Long term product durability and maintainability**
  - Localized modifications and automatic replay
  - Postpone hardware re-engineering by optimal resource usage analysis

Towards Calculus Based Engineering and Product Based Assurance
Constraints

• **Soundness**

• **High Automaticity and scalability**
  • In intended usage domain at airbus’s

• **Analysis of unaltered programs**
  • “What is analysed is what will fly”

• **Usability by standard software engineers on standard machines**
  • No initial high level skills in theoretical computer science required
  • Standard workstations Airbus uses to buy

• **Ability to be integrated into the DO178B (and C !) conforming process**
Outline

• Introduction
• Context, Objectives and Constraints
• **Basics**
  • Industrial state-of-the-use
  • Next transfers
  • Main issues: solved & remaining
• Conclusion
Basic principles

• Abstract Interpretation based Static Analysis
  • http://www.di.ens.fr/~cousot/AI/

• Program Proof
  • Hoare’s triple: http://en.wikipedia.org/wiki/Hoare_logic
  • Dijkstra’s Weakest Precondition:
    http://en.wikipedia.org/wiki/Predicate_transformer_semantics
  • Automatic theorem proving:
    http://ergo.lri.fr/
Outline

- Introduction
- Context, Objectives and Constraints
- Basics
- **Industrial state-of-the-use**
- Next transfers
- Main issues: solved & remaining
- Conclusion
Verification Tools

• Static Analysis
  • Frama-c (CEA, http://frama-c.com/) / TASTER (Airbus)
  • Frama-c / FAN-C (Airbus)
  • Fluctuat (CEA)

• Program Proof (deductive methods)
  • Caveat (CEA)
Rule checking

• **CheckC / TASTER**
  • Functionality: **Verification of C coding rules**
  • On top of Frama-c kernel (exploits the AST built by the kernel)

• **Fan-C**
  • Functionality: **Verification of control and data flows (conformity LLR <-> C code)**
  • Abstract Interpretation based static analysis of the C source
  • Takes profit from Frama-C Kernel (AST CIL) and plug-ins : Value, Users, Inout and From
Executability

• **Astrée (AbsInt, ENS [http://www.astree.ens.fr/])**
  - Functionality: *proof of absence of Run Time Errors of C programs*
  - Abstract Interpretation based static analysis of the C source code
  - “Double specialisation” paradigm for precision (“zero false alarm”)
  - Best suited for embedded synchronous C programs produced from “SCADE like” specifications

• **Fluctuat (CEA)**
  - Functionality: computes floating-point inaccuracies, proves stability computation schemes, performs some functional proofs
  - Abstract Interpretation based static analysis of the C source code
  - Best suited for the analysis library components
Executability

• **a³ / Stack** ([http://www.absint.com/ait/index.htm](http://www.absint.com/ait/index.htm))
  - Functionality: computes an upper-bound of the memory consumed by the program stack (usually from a task’s entry point)
  - Maxim memory allocated to the stack is set accordingly
  - Static analysis by Abstract Interpretation of programs in binary form

• **a³ / WCET** ([http://www.absint.com/stackanalyzer/index.htm](http://www.absint.com/stackanalyzer/index.htm))
  - Functionality: computes an upper-bound of the Worst Case Execution Time (usually from a task’s entry point)
  - This upper-bound can then be compared to an allowed time-budget
  - Static analysis by Abstract Interpretation of programs in binary form
  - Includes a model of the processor and peripherals
  - Best suited for embedded synchronous C programs produced from “SCADE like” specifications
Program proof

- Caveat (CEA)
  - **Functionality**: Proof of specifications expressed in first order logic
  - Analysis of C source code
  - Weakest Precondition (Dijkstra) computation
  - Theorem proving (Caveat’s theorem prover + Alt-Ergo (INRIA))
  - Best suited for source code vs Low Level requirements verification
## Current scope of the tools (+method)

<table>
<thead>
<tr>
<th></th>
<th>Flight Controls (DAL A)</th>
<th>(Platform) DAL B &amp; C functions</th>
<th>(Platform) DAL D functions</th>
<th>Platform software (drivers)</th>
<th>I/O boards</th>
<th>DAL E or Software tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rule checking</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Executability</td>
<td>✓</td>
<td>✓12</td>
<td>✓12</td>
<td>✓12</td>
<td>✓</td>
<td>✓12</td>
</tr>
<tr>
<td>Program proof</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

1: RTE (Astrée, CodeSonar), Floating-point (Fluctuat)
2: Stack usage (a3 / Stack)
3: a3 / WCET
### Tools (+method of use) vs objectives

<table>
<thead>
<tr>
<th></th>
<th>Perfo(^1)</th>
<th>Computation safety</th>
<th>No other (sound) mean</th>
<th>Activity cost savings</th>
<th>Early maturity</th>
<th>Product durability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rule checking</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
</tr>
<tr>
<td>Executability</td>
<td>✓ (^1)</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>✓</td>
<td>✓ (^1)</td>
</tr>
<tr>
<td>Program proof</td>
<td>-</td>
<td>✓</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
</tr>
</tbody>
</table>

\(^1\): contribution to the optimal use of hardware resources: a\(^3\) / Stack and WCET
## Tools (+ methods) vs constraints

<table>
<thead>
<tr>
<th></th>
<th>Soundness</th>
<th>Automaticity &amp; scalability</th>
<th>Unaltered programs</th>
<th>Standard engineers</th>
<th>Standard machines</th>
<th>DO-178</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rule checking</td>
<td>✓1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Executability</td>
<td>✓1</td>
<td>✓</td>
<td>✓2</td>
<td>✓3</td>
<td>✓</td>
<td>✓4</td>
</tr>
<tr>
<td>Program proof</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

1: With the exception of syntactic and pattern matching tools
2: Some pieces of code like asm blocks must be removed (rare); insertion of directives
3: Astrée, Fluctuat: service currently performed by static analysis specialists
4: So far, the decision to claim a certification credit from the use of Astrée and Fluctuat has not been made;
## Current Deployment

<table>
<thead>
<tr>
<th>Rule checking</th>
<th>Flight Controls (DAL A)</th>
<th>(Platform) DAL B&amp;C functions</th>
<th>(Platform) DAL D functions</th>
<th>Platform software (drivers)</th>
<th>I/O boards</th>
<th>DAL E or Software tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rule checking</td>
<td>✓ 1</td>
<td>✓ 12</td>
<td>✓ 12</td>
<td>✓ 1</td>
<td>✓ 1</td>
<td>-</td>
</tr>
<tr>
<td>Executability</td>
<td>✓ 345</td>
<td>✓ 34</td>
<td>✓ 34</td>
<td>✓ 4</td>
<td>✓ 45</td>
<td>-</td>
</tr>
<tr>
<td>Program proof</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>✓</td>
</tr>
</tbody>
</table>

1: Coding rule checker (CheckC/TASTER)
2: Data & Control flow checker (Fan-C)
3: Astrée, CodeSonar, Fluctuat
4: a3 / Stack
5: a3 / WCET
Development Process

- Specification
  - Design architecture
  - Static design
    - LL requirements
  - Automatic Coding
    - Static design
      - LL requirements
  - Coding
    - Unit Verification
      - Integration Verification
        - Validation level checks
          - Astrée, Fluctuat
          - a³ Stack
          - a³ WCET
        - a³ Stack
          - a³ WCET
      - Integration Verification
        - Coding
          - Integration Verification
            - Validation level checks
              - Astrée, Fluctuat
              - a³ Stack
              - a³ WCET
            - a³ Stack
              - a³ WCET
  - Coding
    - Integration Verification
      - Validation level checks
        - Astrée, Fluctuat
        - a³ Stack
        - a³ WCET
  - Astrée, CompCert
Outline

• Introduction
• Context, Objectives and Constraints
• Basics
• Industrial state-of-the-use
• **Next transfers**
• Main issues: solved & remaining
• Conclusion
CompCert (INRIA, http://compcert.inria.fr/)

• Functionality
  • Optimising C compiler for (processor, execution platform):
    • Targets

• Underlying principles & Technology
  • C compiler developed and proved in Coq

• First application domain (EYYW)
  • EYYW’s interest in CompCert
    • Under control optimisations => WCET reduction
    • Proofs made on source still hold after compilation
  • Ricardo Bedin França’s CIFRE Thesis (Airbus / IRIT)
  • Ongoing feasibility study for application to a flight control function
AstréeA (Ecole normale supérieure, http://www.astreea.ens.fr/)

- **Functionality**
  - Proof of absence of Run Time Errors of asynchronous programs

- **Underlying principles & Technology**
  - Abstract Interpretation based static analysis of the C source code
  - Included: a model of the ARINC 653 parallel model

- **Targeted application domain (EYYW)**
  - IMA functions (e.g: Flight Warning)
  - POSIX functions
Dynamic Analysis

• Functional Verification
  • Properties expressed formally, i.e., in ACSL (Frama-C specification language)

• Execution on SIMUGENE

• Evaluation of properties rather than proof

• First tool (internal research prototype)
  • Low Level Requirement functional coverage for DAL C function
  • Automation of an heavy intellectual analysis
  • Run time data are captured during execution on SIMUGENE
  • Evaluation is then performed
Outline

• Introduction
• Context, Objectives and Constraints
• Basics
• Industrial state-of-the-use
• Next transfers
• **Main issues: solved & remaining**
• Conclusion
Main issues: solved & remaining

• **Context**
  - All formal tools Airbus uses come from research
  - Airbus has been working with the researchers and tool developers from the beginning

• **Solved**
  - Peculiarities of embedded code (very often low level code)
  - Conformance to DO-178B
  - Acceptance by developers and managers

• **Remain to do for benefiting more from Formal Methods**
  - Proof confirmation after compilation (semantic preservation)
  - Deeper process transformation: towards much more computation based engineering
Conclusion

- **Ongoing research about a new development strategy**
  - Rule checking and executability as soon as code is available
  - Functional verification and coverage by a **combination** of
    - Proof
      - Requires formalised requirements
    - Dynamic analysis
      - The oracles are the formalised requirements
    - Classical test
  - **Process definition**
    - will still comply with DO-178[BC]...
    - Perhaps without being fully structured by the standard (as it is now)
End