Deploying formal methods ...
Who we are?

Center of Competences for:
- Electronics and on board Software in real time applications
- Avionics and Simulation

Business Center
- Developing and selling products

EMPLOYEES
- Electronics: 140
- Software: 200
- Manufacturing: 115
- Other: 220
Our avionics products

Products / Equipment's sets

domains

- Flight control
- Warnings
- Maintenance
- Communication
Elements on avionics

Architecture overview for the A330/340

- SDU
- FMGEC
- ATSU
- SDAC
- Mobile surfaces
- Engines
- Ground systems
- Flight parameters
- Systems state parameters
- FWC
- DU
- MCDU
- ATSU
- SDU
- FADEC
- FCPC
- FCSC
Elements on avionics

Architecture overview for the A330/340

Means for dependability

- Quality
- Architecture redundancy
- Function dissymmetry
- Technology dissemblance

Systems state parameters

Flight parameters

Ground systems

Engines

Mobile surfaces

- SDAC
- FWC
- DU
- FMGEC
- MCDU
- ATSU
- SDU
- FADEC
- FCPC
- FCSC
From the most critical ...

- Electrical Flight Control
- Safety level: critical (A)
- Mono-application
- Sequential time-triggered application
- Hard realtime constraints

**Diagram:**

- **Sensor** to **Input device**
- **SAO or SCADE Application (Boolean, numeric computation)**
- **Hardware handling, Fault detection**
- **Computer hardware**
- **Output device** to **Actuator**

**System under control (mobile surfaces)**
... To less critical

- A/C – Ground data communications
- Safety level: essential (C)
- Several independant applications
- Multitask asynchronous application
- "Soft" realtime constraints (communication timeouts)
Based on DO-178B/ED-12B standards

• Guidance for satisfying airworthiness requirements
• Define processes and processes data
• Level of assurance and completion criteria depend on software level
• Industry-accepted techniques and methods
• Otherwise equivalence demonstration for alternative means

Regular revision of DO-178_B/ED-12_
• Next coming will be DO-178C
Life cycle: the “V” model

Development process
- Develop the software
- Avoid error
… Life cycle: the “V” model

Development process
- Develop the software
- Avoid error

Verification process
- Provide conformance evidence
- Detect error
... Life cycle: the “V” model

Quality of the processes

- Specification
- High Level Design
- Low Level Design
- Revues and Analyses
- Unit testing
- Validation testing
- Integration testing
- Coding

Development process
- Develop the software
- Avoid error

Verification process
- Provide conformance evidence
- Detect error
The driving trends

- More functions
- More complex
- Permanent evolution
- Increasing assurance requirements

<table>
<thead>
<tr>
<th>Year</th>
<th>Volume (k.bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td>4 k *</td>
</tr>
<tr>
<td>2001</td>
<td>23k - A300B</td>
</tr>
<tr>
<td>2002</td>
<td>2 M - A310</td>
</tr>
<tr>
<td>2003</td>
<td>4 M - A320</td>
</tr>
<tr>
<td>2004</td>
<td>12 M - A330/340</td>
</tr>
<tr>
<td>2005</td>
<td>200k - A300FF</td>
</tr>
</tbody>
</table>
Motivation for formal verification technique

• Recurrent problems with test-based verification
  ‣ Costs: test means and tools, test software, coverage completion
  ‣ Intrinsic difficulties on: robustness checks, determination of computer-resources upper-bounds, computation safety => suboptimal architecture, resources-consuming fault-tolerance mechanisms

• The problems are increasing
  ‣ Trend towards software-intensive systems: more functions implemented in software, more sophisticated functions, new functions
  ‣ Evolution of underlying hardware technology: integration level, modern processor architecture, floating-point operators
... Motivation for formal verification

• As a consequence
  ‣ Test alone will not cover all future needs in software verification
  ‣ But test will still remain

• Introduction of static analysis
  ‣ Main idea: all dynamic properties are « present » in the code of the program
  ‣ Analyse the source code - at compilation-time - to check execution-time properties
    ‣ exhaustive (notion of proof => maximum coverage)
    ‣ highly automatized
  ‣ Grounded on so-called « formal methods »
    ‣ Well-founded on scientific theory
    ‣ Hoare logic, theorem proving
    ‣ Abstract interpretation
Objectives

- Improve verification processes
- Priority
  - Safety-critical software
- Operational use
  - Early deployment on A3xx program
  - Generalisation of use on mid-term A/C program
Orientation

- Ease of learning and use.
  « Standard » avionics software developers must be able to use the tools

- Early payback.
  New process must have better characteristics (productivity, quality-effectiveness)

- Easy integration.
  The use of the tools should not break down the actual verification process and environment

- Ability to cope with real program.
  The tools must be able to analyze program without any modification
  C programming language
  Assembly language or binary code as required

- Scale-up to real size program.
  Analyze at least a 100000 LOC-program (whole-program tools)
  Analyze an unmodified elementary service (unit-service tools)
Properties of interest

- Real-time analyser
- Functional analyser
- Safety analyser
- Numerical analyser
- Resources properties
- Program under analysis
- Data properties
- End-user verification methodology based on formal static analysis

A set of independent static analysers

Floating-point precision properties

Runtime error properties
Main steps

• Start in 1996

• Research and Development step
  ‣ Industrial transfer driven by industry
  ‣ Separate short-term and mid-term problems
  ‣ Strong partnership with academics, spin-off
  ‣ Develop real-sized methods and tools

• Transfer to operational projects (2001)
  ‣ A380 A/C program
  ‣ Adaptation to real projects conditions
    – Partial implementation, knowledge transfer, ..

• Integration within the verification workbench (2003)
  ‣ Part of the “normal” verification techniques
  ‣ Maintenance, training, qualification
  ‣ End-user support
  ‣ ...
Functional properties

• CAVENAT tool based on Hoare logic
  ‣ Low level requirements checks
  ‣ Automatic theorem proving + interactive proof-assistant

Place in the development cycle

Precond: true

\[
\text{int find(int tab, int size, int elt)}\\
\{ \\
\text{........} \\
\}
\]

Postcond:

\[
(find = 1) \leftrightarrow (\exists i: i \geq 0 \land i < size\text{:tab}[i] = elt)\\
(find = 0) \leftrightarrow \neg (\exists i: i \geq 0 \land i < size\text{:tab}[i] = elt)
\]
Functional properties

- CAVEAT tool based on Hoare logic
  - Low level requirements checks
  - Automatic theorem proving + interactive proof-assistant

```c
int find(int tab, int size, int elt)
{
    ........
}
```

Postcond:

\[
(find = 1) \iff (\exists i: i \geq 0 \land i < \text{size}: \text{tab}[i] = \text{elt})
\]

\[
(find = 0) \iff \neg (\exists i: i \geq 0 \land i < \text{size}: \text{tab}[i] = \text{elt})
\]

Place in the development cycle
Resources properties

• AiT and Stack tools based on abstract interpretation
  ‣ Stack for execution stacks upperbounds
  ‣ AiT for Worst-Case Execution Time
  ‣ Both analyse the binary executable code

Place in the development cycle

Whole binary code of a > 120000 loc program
Stack usage ≤ Stack upperbound
WCET value ≤ WCET upperbound
Resources properties

- AiT and Stack tools based on abstract interpretation
  - Stack for execution stacks upperbounds
  - AiT for Worst-case Execution Time
  - Both analyse the binary executable code

Place in the development cycle

- Whole binary code of a > 120000 loc program
- Stack usage ≤ Stack upperbound
- WCET ≤ WCET upperbound
- WCET: Flight Control
- Stack: Flight Control
- Warning
- Communication
Computation safety properties

- ASTREE tool based on abstract interpretation
  - Prove the absence of runtime errors on synchronous program
  - Fully automatic, zero false alarm (under development)

Place in the development cycle
Computation safety properties

- ASTREE tool based on abstract interpretation
  - Prove the absence of runtime errors on synchronous program
  - Fully automatic, zero false alarm (under development)

Place in the development cycle

Whole program > 120,000 loc:
- Numeric overflow,
- Array bound check,
- ...

FCGU Flight Control
Conclusion

• Current status
  ‣ Introduction of static analysis well accepted
    – If clear and concrete benefits
    – If local impacts on activities and processes
  ‣ Positive first feedbacks from partial implementation on A380
  ‣ Formal verification re-conducted on A400M

• The future
  ‣ Generalisation (all classes of software) for next mid-term A/C
  ‣ Extension of tools (classes of properties)
  ‣ What could be the best-fitted certification framework?

• More details on tools
  ‣ CAVEAT [CEA Laboratory: www-drt.cea.fr]
  ‣ ASTREE [ENS Laboratory: www.astree.ens.fr]
  ‣ AiT, Stack [Absint Company: www.absint.com]
Ce document et son contenu sont la propriété d’AIRBUS FRANCE S.A.S. Aucun droit de propriété intellectuelle n’est accordé par la communication du présent document ou son contenu. Ce document ne doit pas être reproduit ou communiqué à un tiers sans l’autorisation expresse et écrite d’AIRBUS FRANCE S.A.S. Ce document et son contenu ne doivent pas être utilisés à d’autres fins que celles qui sont autorisées.

Les déclarations faites dans ce document ne constituent pas une offre commerciale. Elles sont basées sur les postulats indiqués et sont exprimées de bonne foi. Si les motifs de ces déclarations n’étaient pas démontrés, AIRBUS FRANCE S.A.S serait prêt à en expliquer les fondements.