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

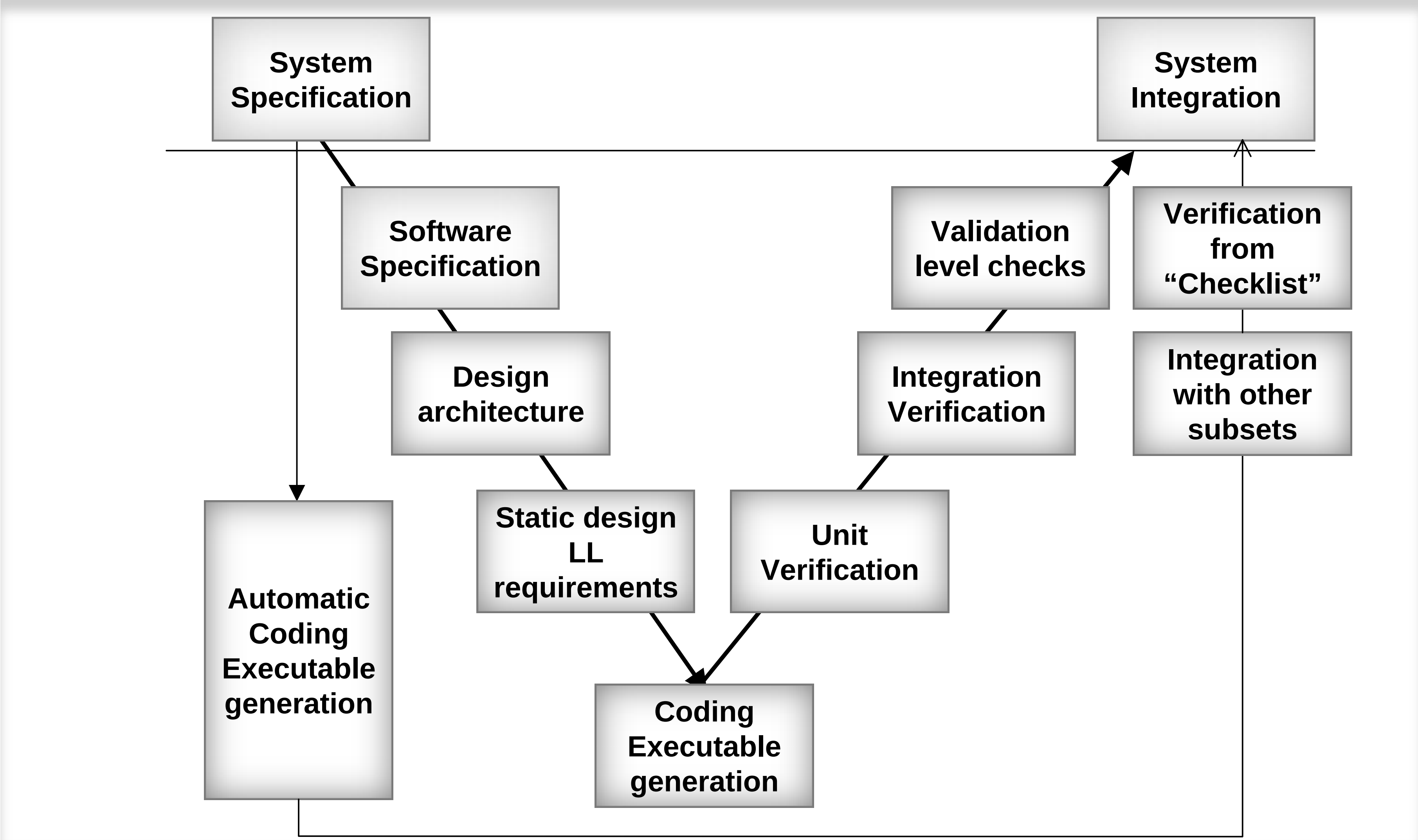
Context: avionics software products

- **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



AIRBUS

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

AIRBUS

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)

  Rule checking

  - Astrée (AbsInt, http://www.absint.com/astree/index.htm)
  - Fluctuat (CEA)
  - a3 / Stack(AbsInt, http://www.absint.com/stackanalyzer/index.htm)
  - a3 / WCET (AbsInt, http://www.absint.com/ait/index.htm)

  Executability

- **Program Proof** (deductive methods)
  - Caveat (CEA)

  Functional verification

AIRBUS

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

- **a^3 / Stack (**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
- **a^3 / WCET (**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

AIRBUS

Current scope of the tools (+method)

	Flight Controls (DAL A)	(Platform) DAL B & C functions	(Platform) DAL D functions	Platform software (drivers)	I/O boards	DAL E or Software tools
Rule checking	✓	✓	✓	✓	✓	✓
Executability	✓	✓[1,2]	✓[1,2]	✓[1,2]	✓	✓[1,2]
Program proof	✓	✓	✓	✓	✓	✓

[1] : RTE (Astrée, CodeSonar), Floating-point (Fluctuat)
[2] : Stack usage (a3 / Stack)
[3] : a3 / WCET

AIRBUS

Tools (+method of use) vs objectives

	Perfo[1]	Computation safety	No other (sound) mean	Activity cost savings	Early maturity	Product durability
Rule checking	-	-	-	✓	✓	-
Executability	✓[1]	✓	✓	-	✓	✓[1]
Program proof	-	✓	-	✓	✓	-

[1]: contribution to the optimal use of hardware resources: a³ / Stack and WCET

Tools (+ methods) vs constraints

	Soundness	Automaticity & scalability	Unaltered programs	Standard engineers	Standard machines	DO-178
Rule checking	✓[1]	✓	✓	✓	✓	✓
Executability	✓[1]	✓	✓[2]	✓[3]	✓	✓[4]
Program proof	✓	✓	✓	✓	✓	✓

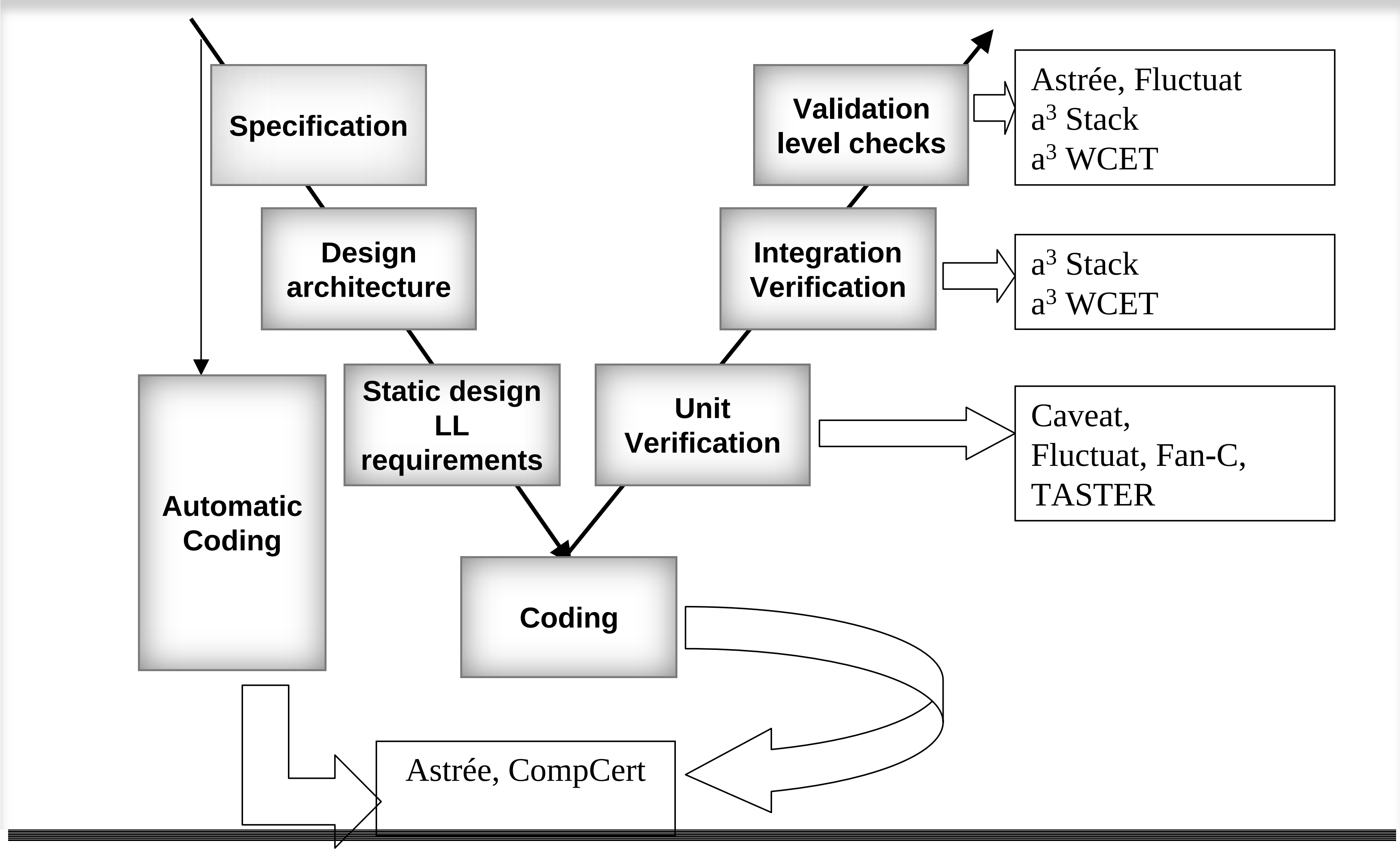
[1] : With the exception of syntactic and pattern matching tools
[2] : Some pieces of code like asm blocks must be removed (rare); insertion of directives
[3] : Astrée, Fluctuat: service currently performed by static analysis specialists ;
[4] : So far, the decision to claim a certification credit from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

Current Deployment

	Flight Controls (DAL A)	(Platform) DAL B&C functions	(Platform) DAL D functions	Platform software (drivers)	I/O boards	DAL E or Software tools
Rule checking	✓[1]	✓[1,2]	✓[1,2]	✓[1]	✓[1]	-
Executability	✓[3,4,5]	✓[3,4]	✓[3,4]	✓[4]	✓[4,5]	-
Program proof	✓	-	-	-	-	✓

[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



Outline

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

PowerPC, MacOS X
PowerPC, Linux
PowerPC, EABI, with GNU or Unix tools
PowerPC, EABI, with Diab tools
ARM, Linux
IA32 (x86 32 bits), Linux
IA32 (x86 32 bits), BSD
IA32 (x86 32 bits), MacOS X
IA32 (x86 32 bits), Cygwin environment under Windows

CompCert (INRIA, htt

- **Functionality**
  - Optimising C compiler fo
    - 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