Model Based Software Engineering for Real-Time Embedded Systems with AADLv2

Frank Singhoff+
+University of Brest, Lab-STICC/CNRS UMR 6285, France

Acknowledgments

- Some of these slides were written with or by Jérôme Hugues (SEI/Carnegie Mellon University) for:
  - AADLv2, An Architecture Description Language for the Analysis and Generation of Embedded Systems. J. Hugues, F. Singhoff. Half day tutorial presented in the ACM HILT conference, Portland, USA, October 2014.
  - AADLv2, a Domain Specific Language for the Modeling, the Analysis and the Generation of Real-Time Embedded Systems. F. Singhoff, J. Hugues. Half day tutorial presented in the International MODELS conferences, Valencia, Spain, September 2014.
  - AADLv2, an Architecture Description Language for the Analysis and Generation of Embedded Systems. J. Hugues F. Singhoff. Half day tutorial presented in the International EMSOFT/ESWEEK conferences, Montreal, Canada, September 2013.
  - Développement de systèmes à l'aide d'AADL - Ocarina/Cheddar. J. Hugues, F. Singhoff. Tutoriel présenté à l’école d'été temps réel (ETR'2009). Septembre 2009. Pages 25-34. Paris.
- Thank you Jérôme :-)

Safety critical systems

- **"A safety-critical system is a system whose failure or malfunction may result in death or serious injury to people, loss or severe damage to equipment/property, ...** "

- Examples: railway, aircraft, automotive, underground.
- Software contributes to the safety of the system.
- How to be sure that a software is safe? Bug free?
- Required by regulation (e.g. avionic systems).
- Today software embedded in critical systems is complex, large.

We focus on Real-Time, Critical, Embedded Systems

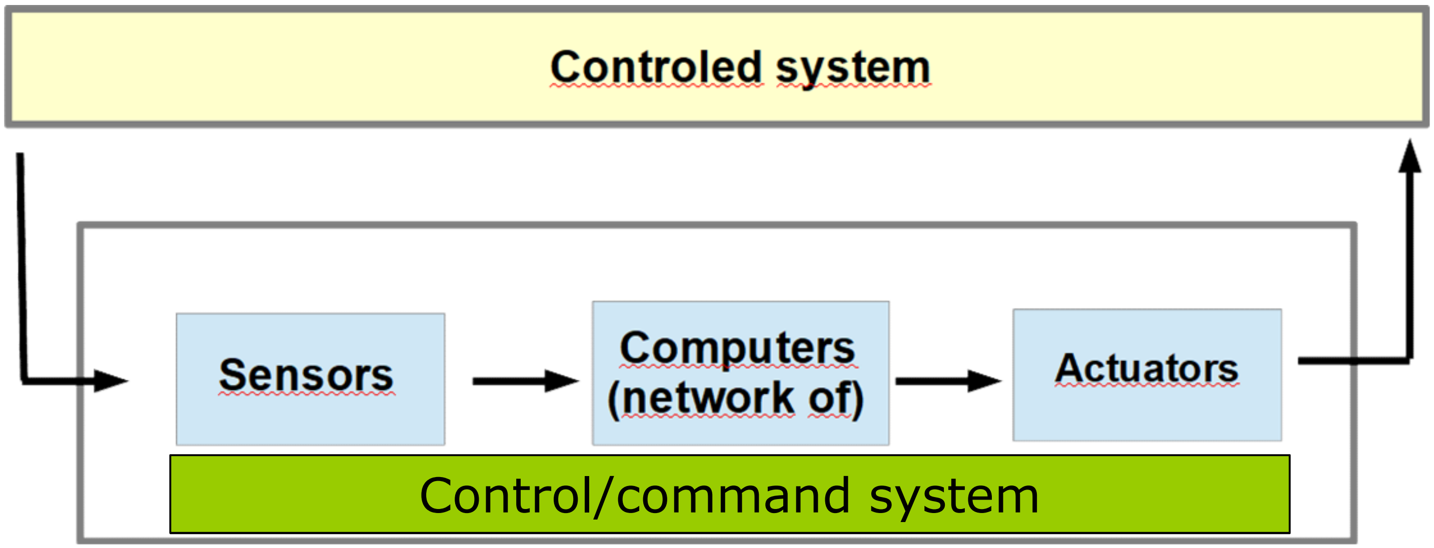
- *« The correctness of the system depends not only on the logical result of computation, but also on the time at which the results are produced »* Stankovic, 1988.
- Properties we look for:
  - Functions must be predictable: the same data input will produce the same data output.
  - Timing behavior must be predictable: must meet temporal constraints (e.g. deadline).
- Predictable means ... we can **compute** the program temporal behavior **before** execution time.

We focus on Real-Time, Critical, Embedded Systems

- **Critical real-time systems:** temporal constraints MUST be met, otherwise defects could have a dramatic impact on human life, on the environment, on the system,

- **Embedded systems:** computing system designed for specific control functions within a larger system.
  - Often with temporal constraints.
  - Part of a complete device, often including hardware and mechanical parts
  - Limited amount of resources.

We focus on Real-Time, Critical, Embedded Systems



- **Real-time control and command software:** computing system/programs which reacts in a given time 1) from sensor inputs 2) to send commands to actuators.

Why MBSE?

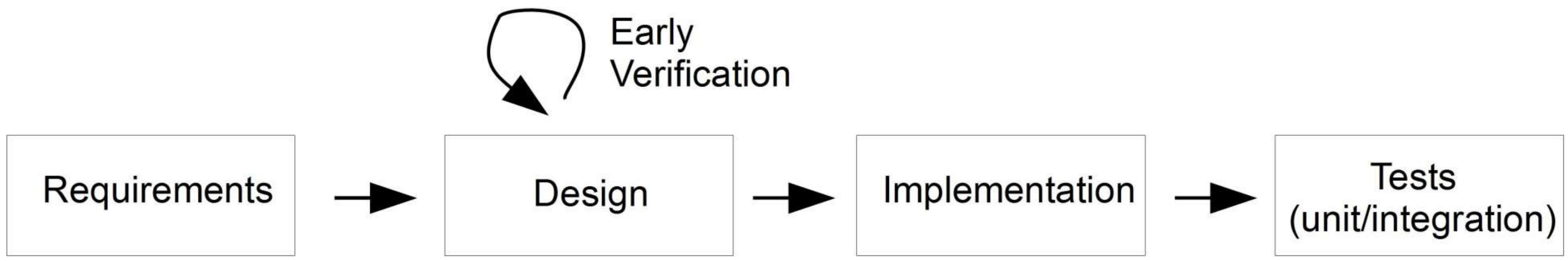
- Mellor et al.* **"... is simply the notion that we can construct a model of a system that we can transform into the real thing."** **S. Mellor, A. Clark, and T. Futagami, "Model driven development," IEEE Softw., vol. 20, no. 5, pp. 14–18, Sep./Oct. 2003.*

- Model Based Software Engineering: focus effort on models instead of software programs
- Working on a higher abstraction level to
  - Make verifications
  - Automatically produce a part of the software artifacts
  - Increase quality and reduce cost

Why MBSE?

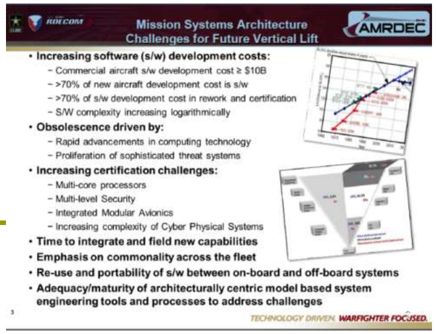
- Increasing complexity of systems to implement
- Concurrent applications: scheduling & communications & synchronization of threads/tasks
- Limited resources: operating system configuration
- Standards (e.g. DO-178)
- Design space exploration: uniprocessor or distributed?
- Verification of timing constraints
- Early verification

Why MBSE?



- **Software engineering** methods/models/tools to master quality and cost
- **Early verification:** multiple verifications, including expected performances, i.e deadlines can be met?

Why MBSE?



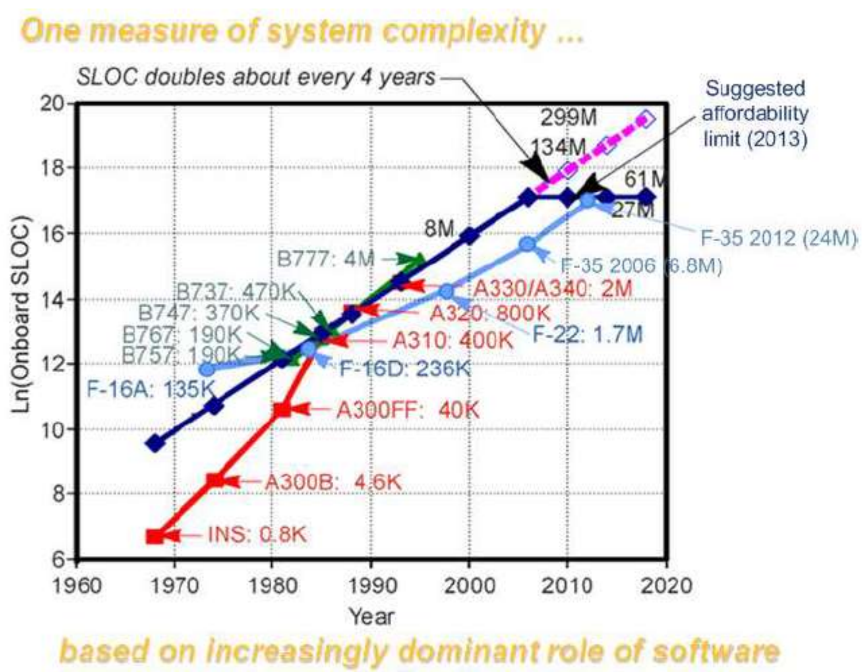
- **From NIST 2002:**
  - 70% of fault are introduced during the design step ; Only 3% are found/solved. Cost : x1
  - Unit test step: 20% of fault are introduced ; 16% are found/solved. Cost : x5
  - Integration test step: 10% of fault are introduced ; 50% are found/solved. Cost : x16

- **Objective:** increase the number of faults found at design step!

Avionic software

- From SAVI program (US research program) who investigated about software in avionic (Peter Feiler)

- SLOC, for Source Line of Code.
- F35 has approximately 175 times the number of SLOC as the F16.
- But, it is estimated to have required 300 times the development effort.
- Software size doubles every 4 years.



Airbus data

	A310	**A320**	**A340**	***A380***
Design	1982	1987	1991	*2000*
Software size (in Mo)	4	10	20	*Several hundreds*
Number of computers	77	102	115	*8*
Number of buses	136	253	368	*500 environ*
Size (in liter) of electronic devices	745	760	830	
Size (in liter) for the autopilot	134	63	31	
MIPS	60	160	250	*Several thousands*

DO-178 standard

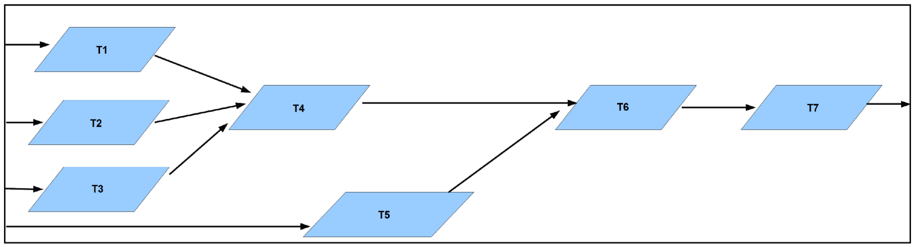
Criticality level	Specific rules	Volume of functions	Consequence	Max # of occurrences
E	0	5%	None	
D	28	10%	Minor	10^{-3}/h
C	57	20%	Major	10^{-5}/h
B	65	30%	Hazardous	10^{-7}/h
A	66	35%	Catastrophic	10^{-9}/h

- Criticality level, Design Assurance Level (DAL)
- DO-178 proposes rules to ensure the reliability of the software (functions, kernel, integration, etc.)
- A function is assigned a criticality level according to the severity of its failure
- Examples: code coverage from the high system requirements, use of formal methods, use model of based engineering (DO-178C)

Objectives of this tutorial

- One solution among others: use an architecture description language
  - to model the system,
  - to run various verification,
  - and to automatically produce the system
- Focus on the AADL 2.x SAE standard

Example: from a master student lab



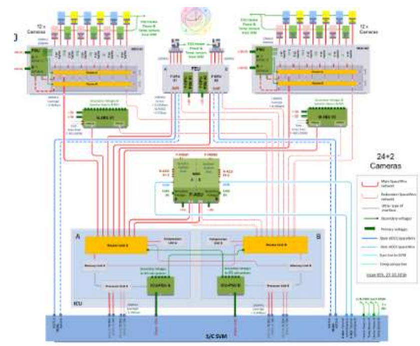
Task	Period and deadline	Execution time	Priority
T1	1000 ms	200 ms	10
T2	1000 ms	200 ms	20
T3	1000 ms	200 ms	30
T4	1000 ms	100 ms	40
T5	…	…	…

Architecture is feasible? Deadline constraints? Communication constraints?

How to design its scheduling? Its communication? How many processors?

- **As a usual business:** design, write programs, and test … and change the architecture (.e.g several processors => review scheduling/communication
  => There is no solution with our example with all constraints ☹
- **MBSE:** design, early verification. If OK move to prototyping (generate glue code and write applicative code), test, … Change the architecture? => change the model and regenerate ☺

Example: PLATO



- **PLATO:** mission of the ESA (launch for 2026) aiming to characterize exoplanetary systems. CNES & Observatoire de Paris.
- **Payload:** 26 cameras, applications on a multicore platform.
- **Space design process:** SRR (system requirement review), PDR, CDR (critical design review), TRR (test readiness review), ...
- **Produced model for the CDR:**
  - 2500 lines of AADL model
  - 2 processors (LEON), 34 threads, 28 data types
  - 562 property associations
  - 257 AADL component types of implementations (entities)

Objectives of this tutorial

- Goal: to model a simple radar system
- Let us suppose we have the following requirements

1. System implementation is composed by physical devices (Hardware entity): antenna + processor + memory + bus
2. and software entities : running processes and threads + operating system functionalities (scheduling) implemented in the processor that represent a part of execution platform and physical devices in the same time.
3. The main process is responsible for signals processing : general pattern: transmitter -> antenna -> receiver -> analyzer -> display
4. Analyzer is a periodic thread that compares transmitted and received signals to perform detection, localization and identification.
5. [..]

Outline

Goal: introduce model-based analysis of embedded real-time critical systems using the AADLv2 Architecture Description Language

- **Part 0: tutorial outline**
- **Part 1: introduction to AADLv2 core**
  - Syntax, semantics of the language
- **Part 2: introducing a case study**
  - A radar illustrative case study
- **Part 3: scheduling analysis with AADL**
  - Introducing real-time scheduling and its use with AADL
- **Part 4: code generation**
  - Embedding functions automatically
- **Part 5: conclusion**