Refinement of AADL models using early-stage analysis methods

Guillaume Brau[1], Jérôme Hugues[2] and Nicolas Navet[1]
guillaume.brau@uni.lu

[1]Laboratory for Advanced Software Systems (LASSY), University of Luxembourg, Luxembourg, Luxembourg
[2]Université de Toulouse – Institut Supérieur de l'Aéronautique et de l'Espace, Toulouse, France

The 4th Analytic Virtual Integration of Cyber-Physical Systems Workshop. Co-located with RTSS 2013. Vancouver, BC, Canada.

December 3rd, 2013

General scope

Context : Distributed Realtime Embedded (DRE) systems

- Safety-critical applications => how to meet the *functional* and *non-functional* requirements ?
  1. deploying specific technologies
  2. addressing the engineering process with relevant *methods* and *tools*

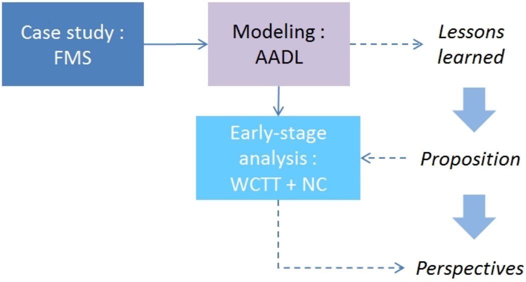
Virtual Integration (VI) for Cyber-Physical Systems (CPS)

VI approaches focus on **analytic techniques** that enable the **early discovery of faults** in CPS **before the system is integrated or its parts are built**.
→ The objective is to discover and resolve problems early during the design and implementation phases where cost impact is low.

Supporting VI

Objective : investigating the early-stage use of analysis methods in system modeling.

1. **lessons learned** from architectural modeling with AADL : benefits and limitations
2. **proposition** to overcome encountered problems
   → analysis as part of the design process
   → exemplification of our beliefs on a real case-study / practical results
3. research **perspectives** and future works



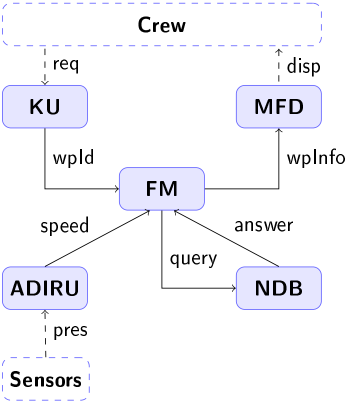
Part I

AADL modeling : lessons learned

Modeling an avionics system with AADL

Case study : Part of a Flight Management System (FMS) [Lauer12].

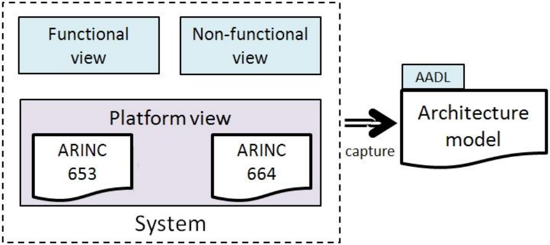
1. **Functional requirements**



2. **Non-functional requirements**.
E.g., temporal constraints :
   - functions response times
   - network traversal times
   - latencies alongs functional chains

3. **Platform** :
   - ARINC 653 for execution resources
   - ARINC 664 (AFDX) for communication resources

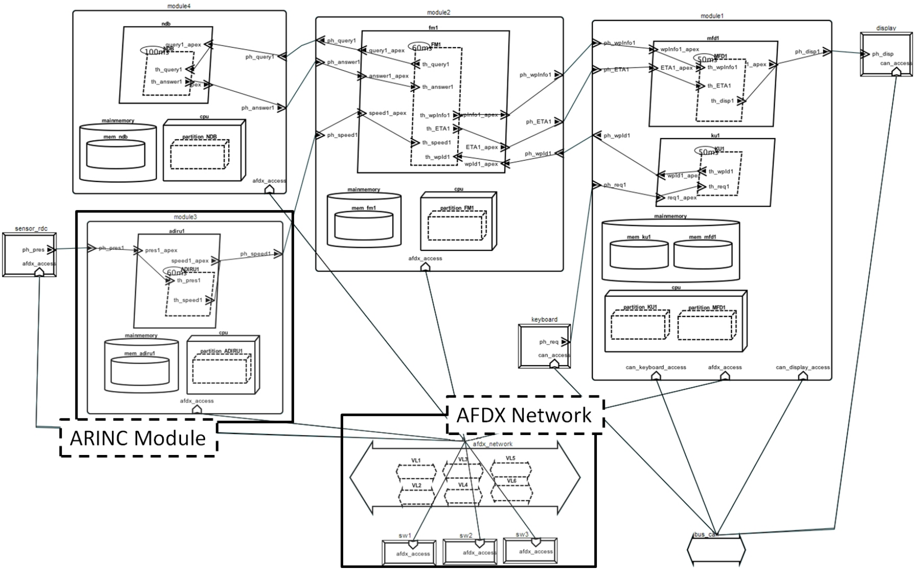
Modeling the FMS with AADL



How to address the FMS modeling?

- AADL core language [AADLv2]
  - standardized *components* classified under *software*, *execution platform* and *composite* categories,
  - basic artifacts : *features*, *implementations*, *properties*, etc.
- standardized annex languages [AADLannexes]
  - ARINC653 annex guidelines and property sets
- proposed extensions : ARINC664 property set and components

AADL model overview



Lessons learned from architectural modeling : benefits

Architectural model :

1. the full architecture model captures the system requirements
2. it is then possible to perform analysis on this architecture model

Benefits

Designed components are virtually integrated within the overall architecture model :
→ early discovering of integration and dimensioning problems
→ possible to (partially) verify the system before actually implementing it

Precondition

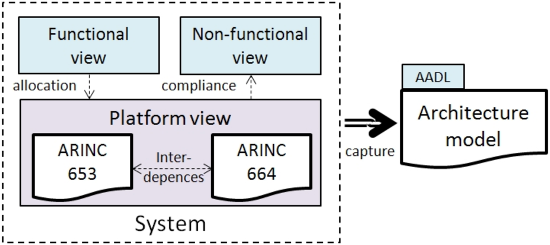
Getting the full architecture model so as to then derive performance analysis.

Lessons learned from architectural modeling : limitations

Modeling issues : allocations, non-functional constraints compliance, components inter-dependencies

→ **raised issues are same as the ones encountered during a classical engineering cycle,**

→ **the architecture model (Virtual Integration) replaces the real system (Integration problems).**

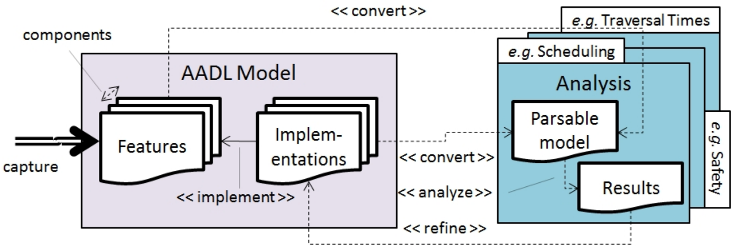


Conclusion

Defining the architecture model amounts to solve dependencies between modeling (and system) concerns

Proposed approach

Analysis as part of the design process :



1. modeling = design space exploration
   - use of analysis methods to discover the design space
   - gradual definition of the architecture model and refinement of its components
   - consistent definition of the parameters = refined parameters meet the constraints
2. (verification of the proposed architecture)
3. *derivation of code, manually or using code generation*

Part II

Exemplification – dimensioning and refining the communication parameters

Starting from an incomplete AADL model

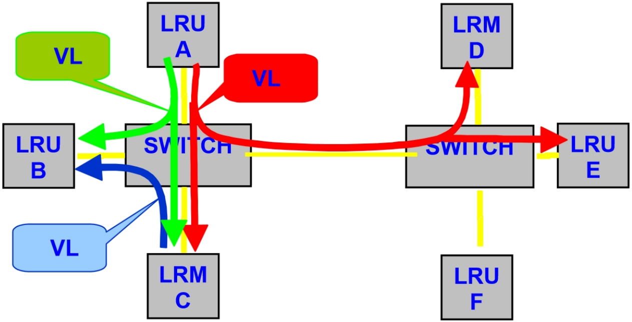
```
system fms end fms;

system implementation fms.impl
subcomponents -- modules and communication components
afdx_network : bus fms_hardware::physical_afdx_link.impl;
sw1 : device subsystem::afdx_switch;
--...
connections -- connections and busses accesses
nt_wpId : port module1.ph_wpId1 -> module2.ph_wpId1;
--...
flows -- wpId, wpInfo, query, answer, speed flows
wpId_fl : end to end flow module1.wpId_src ->
nt_wpId -> module2.wpId_sink ;
--...
properties -- here are specified the latency constraints
-- and bindings to VL that have to meet those constraints
Latency => 0ms .. 15 ms applies to wpId_fl;
--...
-- But how to define the communication components and parameters?
end fms.impl;
```

AFDX overview

ARINC 664 standard → AFDX

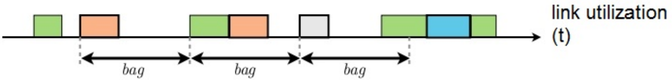
- Avionics Full-Duplex Switched Ethernet = deterministic communication network
- Virtual-Links : logical connections between one emitter and one or several receiver(s)
  - static route defined at system start-up



AFDX overview

ARINC 664 standard → AFDX

- Avionics Full-Duplex Switched Ethernet = deterministic communication network
- Virtual-Links : logical connections between one emitter and one or several receiver(s)
  - static route defined at system start-up
  - dedicated bandwidth according to parameters defined at system start-up :
    → Bandwidth Allocation Gap (ms) = minimum elapsed time between two frame sending
    → Maximal packet size (Bytes)



Defining the communication parameters

- It is possible to assume :
  - the necessary VLs → one VL for the data flows with the same emitter/receiver(s) couple
  - the VLs maximal packet size → set to the maximum standard value
  - the VLS routes → considering well-known routing algorithms (such as SPF)

What about the BAGs? How to be sure that their definitions will meet the constraints?

Defining the communication parameters

- It is possible to assume :
  - the necessary VLs → one VL for the data flows with the same emitter/receiver(s) couple
  - the VLs maximal packet size → set to the maximum standard value
  - the VLS routes → considering well-known routing algorithms (such as SPF)

What about the BAGs? How to be sure that their definitions will meet the constraints?

Defining the Bandwidth Allocation Gap

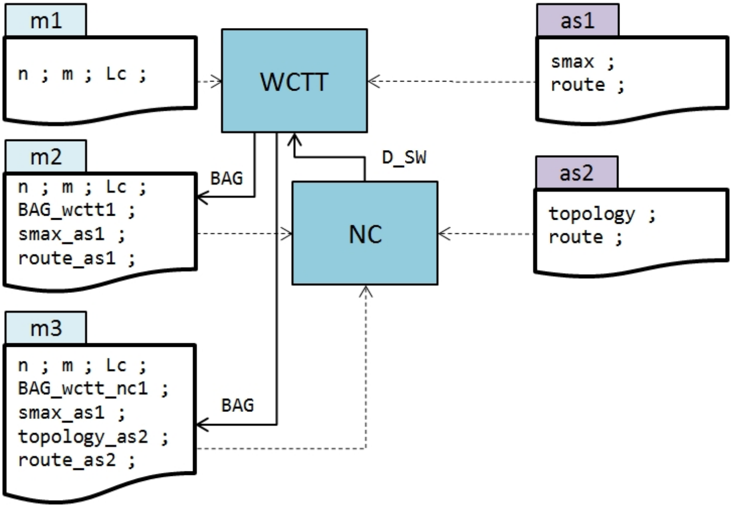
For each VL : $BAG = 2^k$ with k is an integer such as $k \in [0, ...7]$ [ARINC664]
→ 8^f solutions with f is the number of data flows
→ **not necessarily evident without appropriate guidelines and/or analysis supports**

Consistent VLs' definition

How to proceed? $\rightarrow$ looped process

1. isolating model input parameters that can be combined
2. finding out an applicable analysis method, given its :
   - mandatory items
   - assumptions
3. executing the analysis $\rightarrow$ refining the model
4. back to step 1

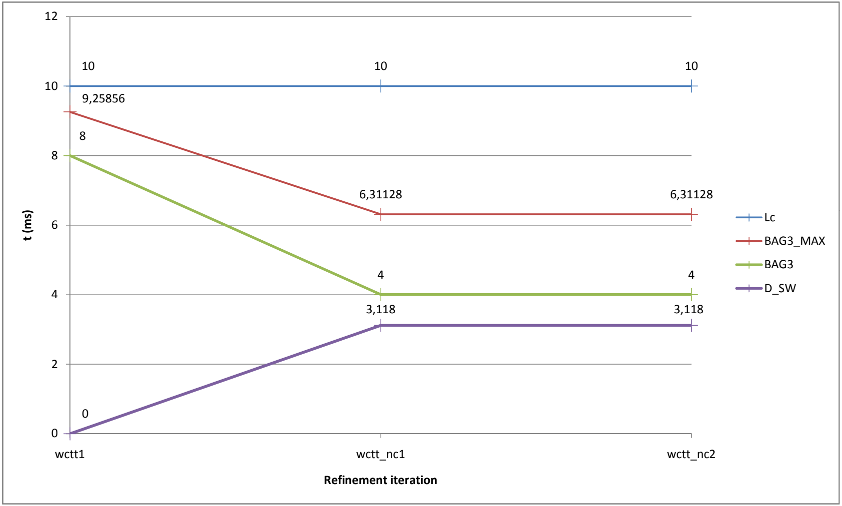
Proposed refinement process



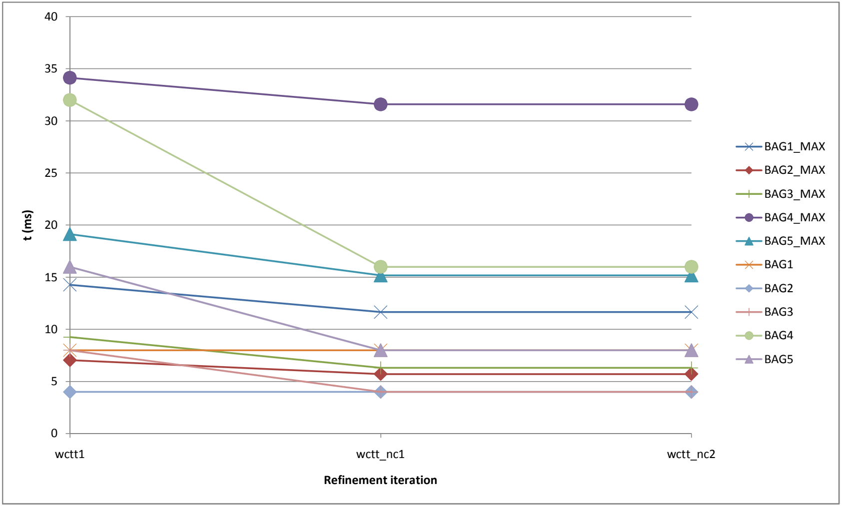
Analysis methods

- 2 complementary analysis methods → network traversal time evaluation
- WCTT = **main analysis method**
  - **outcome** : given the latency constraints expressed on the data flows, it is possible to figure out the BAGs
  - **precision** : coarse grained evaluation (depends on the precision of the model and on the complementary results)
  - **execution** : analytic formula computed "by hand"
- Network Calculus (NC)= **complementary analysis method**
  - **outcome** : given the data flows parameters, the delay suffered by each frame in the network can be computed
  - **precision** : exact evaluation
  - **execution** : NC algebra computed by dedicated tools → RTaW-Pegase in our case

Experimental results



Experimental results



Part III

Conclusion and perspectives

Pointed out problem

- modeling and analysis artifacts often addressed as **distinct and independent** steps (even if a "link" between modeling and analysis is always considered)

How to provide the full model that it is then possible to analyze and validate?

- dependencies between modeling concerns (functional, non functional, deployment)
- dependencies between components and their parameters
- missing information
- etc.

Defining the model to verify may not be so easy...

Exemplified proposition

Considering analysis methods as an “actor” of the design process

- applying early-stage analysis methods on the incomplete model to narrow gradually the design space
  $\rightarrow$ given the available information, the model is progressively refined and validated
  $\rightarrow$ the model is consistent = deduced parameters guarantee that non-functional constraints are met
  $\rightarrow$ the designed system is "correct-by-construction"

Future works

Formalizing the use of analysis methods along with modeling languages

→ analysis feasibility, associated assumptions, composition and/or complementarity between analysis, trust, etc.

Requirement Enforcement and Analysis Language

- REAL (under standardization) → manipulation of `theorems` to check a set of predicates defined on the system design
  1. checking global consistency of the model
- extension of REAL to manage *tools*
  2. detect when an analysis is feasible
  3. exploit relationships between analysis

Discussion

Thank you for your attention

Awaiting your questions !

References

M. Lauer.
Une méthode globale pour la vérification d'exigences temps réel - Application à l'Avionique Modulaire Intégrée.
Thèse de doctorat, Institut National Polytechnique de Toulouse, Toulouse, France, juin 2012.

SAE/AS2-C.
Architecture Analysis & Design Language V2 (AS5506A), January 2009.

SAE/AS2-C.
Data Modeling, Behavioral and ARINC653 Annex document for the Architecture Analysis & Design Language v2.0 (AS5506A), October 2009.

Aeronautical Radio Incorporated.
ARINC Report 664P7-1 Aircraft Data Network, Part 7, Avionics Full-Duplex Switched Ethernet Network.