Refinement of AADL models using early-stage analysis methods – An avionics example

Defining communication parameters during architecture-level design using jointly AADL and network traversal time analysis methods

Guillaume Brau, Nicolas Navet
Laboratory for Advanced Software Systems
University of Luxembourg
6, rue R. Coudenhove-Kalergi, Luxembourg (Luxembourg)

Jérôme Hugues
Institut Supérieur de l'Aéronautique et de l'Espace
Université de Toulouse
10, avenue Edouard-Belin, Toulouse (France)

TR-LASSY-13-06

Thursday 6th February, 2014

Contents

Date	Version	Description	Author(s)
October 11, 2013	1	Initial document	G. Brau, J. Hugues, and N. Navet
February 6, 2014	2	General revision (released)	G. Brau, J. Hugues, and N. Navet

Table 1: Revision History

Refinement of AADL models using early-stage analysis methods

Guillaume Brau[1] Jérôme Hugues[2] Nicolas Navet[1]
[1]University of Luxembourg, Laboratory of Advanced Software Systems,
6, rue Richard Coudenhove-Kalergi, L-1359 Luxembourg, Luxembourg
{guillaume.brau, nicolas.navet}@uni.lu
[2]Université de Toulouse – ISAE, 10 avenue E. Belin, 31055 Toulouse, France
jerome.hugues@isae.fr

Abstract

Model-Driven Engineering (MDE) is a relevant approach to support the engineering of distributed embedded systems with performance and dependability constraints. MDE involves models definitions and transformations to cover most of the system life-cycle: design, implementation and Verification & Validation activities towards system qualification. Still, few works evaluate the early integration of performance evaluation based on architectural models. In this report, we investigate the early-stage use of analysis in AADL modeling. Precisely, we exemplify on an avionics case study how to dimension the data flows for an application distributed over an AFDX network. Based on the insight from this study, we suggest a simple framework and associated techniques to efficiently support analysis activities in the early-stage design phases.

1 Introduction

Context of the work. Distributed Real-time Embedded (DRE) systems are present in safety-critical domains such as transportation, telecommunications, health services, military or space. These systems have to meet both the *functional* and *non-functional* requirements. Hence, DRE systems encompass specific technologies to realize the required service with the expected performance metrics (*e.g.* time, security or safety) through dedicated networks, processors or real-time operating systems. In addition, the engineering process has to address efficiently system modeling and evaluation of all metrics. In such context, Model-Driven Engineering (MDE) is a relevant approach to support the engineering of DRE systems. MDE involves models definitions and transformations to cover the system life-cycle towards system qualification.

Definition of the problem. Many experiments indicate that the distance between the activities steps in a classical V-cycle is detrimental and usually slows down the development process [1]. In practice, a significant part of errors is injected at early-stage of the engineering process, while being detected during later integration phases. As a consequence, regressions and rework activities have an important weight on the overall project costs. Designers have to face the following paradox : mastering all the facets of the system and the underlying implementation and integration problems before the system is actually implemented and the verification activities, that aim to detect the problems, carried out.

We believe that this paradox can be lifted considering models with sufficient power of expression to guide the conception phases (e.g. using interface or behavioral models on which one can both reason and iterate) and supporting early-stage analysis. This “integrate, then build” approach, also known as *virtual integration* is promising to support the design of complex systems.

Contributions and objectives. In this paper, we use the Architecture Analysis & Design Language (AADL) [2] as the pivot to capture the system architecture and derive its performance. AADL is an Architecture Description Language (ADL) suitable to describe systems, capturing the *functional* and *non-functional* concerns together with the *operational platform.* As AADL provides modeling elements with a precise *syntax* and well-defined *semantics*, it is possible to : 1) perform analysis and 2) derive implementations thanks to code generation [3].

The focus of this paper is twofold. First, taking a specific example coming from the avionics, we show how to accurately seize important parameters in the AADL model. As the system that is modeled includes technologies not supported within the core language, we extend the work in [4] so as to include in the AADL model the networking elements and capture the overall DRE system.

Secondly, based on lessons learned during the modeling experience, we propose to jointly use AADL models and analysis methods in order to gradually refine and verify the model. We investigate and examplify this strategy one the avionics case study. Taking as example the network communications and the expressed timing constraints, we show that using complementary analysis methods allows to deduce missing parameters while maintaining the consistency of the model (i.e. chosen parameters guarantee that non-functional constraints are met).

Related works. There are several related works which deal with early-stage analysis. Considering scheduling issues, the MoSaRT approach [5] proposes a domain specific language in order to assist designers to validate their architecture during the design phase, reducing the gap with analysis tools. MAST [6] is an open environment that allows to describe a real-time system thanks to a MAST model and to compute a worst-case response time schedulability analysis using a set of tools developed within the suite. Based on AADL, Delange et al. [4] explain how to support the scheduling analysis of avionics systems with a transformation tool chain towards their Cheddar analysis tool. The MASIW project provides an AADL-based tool set [7] in order to design and integrate avionics systems. It enriches the AADL basics and allow to perform several analysis in order to check the compliance with constraints. Other dimensions like security [8] or safety [9] have been addressed using AADL as a root.

The paper is organized as follows: we first introduce the avionics case study and present how to model it in AADL (Section 2). In Section 3, we illustrate relationships between modeling concerns, and, using analysis, we explain how to solve those dependencies for some important communication parameters. Finally, in Section 4, we draw conclusions from this case study and discuss future work.

2 Modeling avionics systems with AADL

In this section we describe how to jointly capture in the Architecture Analysis & Design Language the functional description of a Flight Management System (FMS), its temporal constraints and the target execution platform, called Integrated Modular Avionics (IMA).

2.1 The Flight Management System

The avionics system to model, coming from Lauer et al. [10], is part of an aircraft's Flight Management System (FMS). It interacts with the crew and provides static and dynamic information about the flight plan (*i.e.* the predefined path between departure and arrival points) : current location, remaining distance and estimated arrival time.

Functional description. The system is made up of five main functions as depicted in Figure 1. The *Keyboard and cursor control Unit (KU)* reads data inputted by the pilot (or copilot) through the keyboards while the *Multi Functional Display (MFD)* refreshes the displays consecutively.

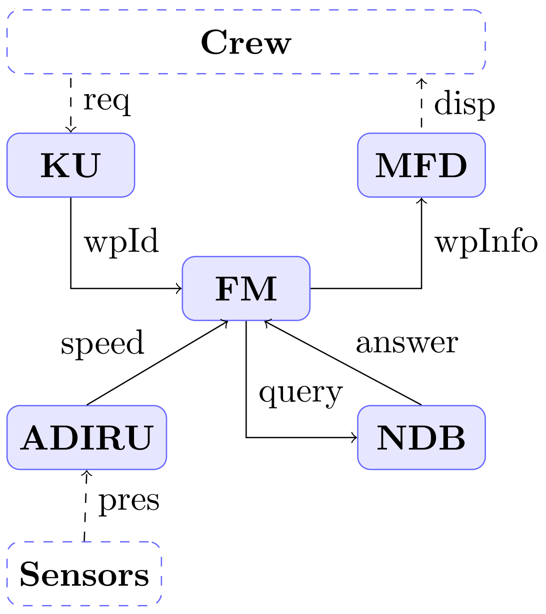


Figure 1: The Flight Management System functional architecture depicts the functions and the data exchanged as well as the interactions with the actors.

From the KU function, crew requests are forwarded to the *Flight Manager (FM)* function which computes the response about the flight plan and returns it to the MFD. For this, it requests static data to the *Navigation Data Base (NDB)* function and also relies on dynamic data from the *Air Data Inertial Reference Unit (ADIRU)*, computed based on sensors measurements.

Temporal constraints. In the avionics context, the system has to conserve a predictable behavior. Several temporal constraints may be expressed upon different ”locations”. Typically, temporal constraints concern :

- response times which are the delays needed to carry out the functions,
- traversal times refering to communication delays between functions,
- latencies along functional chains that encompass a succession of response times and traversal times.

2.2 The Integrated Modular Avionics platform

The *functions* are executed in an *Integrated Modular Avionics (IMA)* environment. In particular, this execution platform supports two standards, used in the case study, that defines the use of the shared hardware and software resources in a deterministic way :

- ARINC653 [11] for computational resources,
- ARINC664 [12] for communication resources.

In the following, we give an overview of the main concepts of these standards. The description emphasizes the key notions that the AADL architecture model has to capture.

ARINC653. The ARINC653 standard defines the management of the functions hosted by a same hardware/software platform (referred as an execution *module* in the following). In this environment, each function is located in a different *partition* with a strict access to processing and memory resources. It means :

- a *temporal* partitioning : partitions are executed during specific time slots defined at system start-up,
- a *spatial* partitioning : each partition has its reserved memory space defined at system start-up.

The temporal and spatial partitioning, as well as the communication channels are hidden to the functions hosted by a partition. It is the aim of the underlying ARINC653 operating system to handle partitions scheduling, access to memory resources and communication services. The static scheduling is directed following four parameters :

- the module *Major Frame* (MaF_m) is the duration of one cycle of the functions executions – thereafter, the cycle is repeated,
- the module *Minor Frame* (MiF_m) allows to execute several instances of one task during the MAF,
- the *offset* ($O_{m,p}$) of one partition is the gap between the MaF_m origin and the start of the partition execution,
- the *duration* ($D_{m,p}$) is the time allocated to each partition to access the processor.

A function may be implemented by a set of sub-functions or threads, scheduled locally – and maybe dynamically – to the partition (following a RM policy for example).

ARINC664 The ARINC664 standard defines a predictable communication network called *Avionics Full Duplex-Switched Ethernet* (AFDX). It uses full-duplex links to carry the packets and switches to route a packet from a sender to one or several receiver(s). AFDX implements the core concept of *Virtual Link* (VL) to share the network bandwidth while maintaining the predictability of the communications. A VL is an unidirectional logical connection from one sender to one or several receiver(s) – unicast or multicast channels. Each VL has :

- a limited bandwidth according to two parameters :
  - the *Bandwidth Allocation Gap* (bag_v) is the minimum time elapsed between two frames sending,
  - the *maximal allowed packet size* ($smax_v$),
- a predefined and static *route* ($route_v$) crossing one or several switch(es).

2.3 The Architecture Analysis & Design Language

AADL [13] is an international standard by the *Society of Automotive Engineers (SAE)*, defining the basics of an architecture description language dedicated to the design of real-time systems. AADL is component-centric and allows to specify both software and hardware parts of a system. It allows one to define consistent block interfaces and to separate them from block implementation.

An AADL model is made out of *components*. Software components (`data`, `thread`, `subprogram`, `process`) are distinguished from execution platform components (`memory`, `bus`, `processor`, `device`) and hybrid components (`system`). Each component shares the semantics of its counterpart in embedded systems terminology.

The behavior of a system (e.g. how functional blocks interact) is fully defined in the standard by means of *properties* (attributes with a dedicated semantics) to progressively refine the semantics of a system (e.g. dispatching invariants, communication patterns, non-functional properties), interface specifications and how components are interconnected. These have a deep impact on the system's behavior. Functional aspects (algorithms) can be attached separately as source code by means of properties. See [2] for a complete presentation of AADL.

AADL proposes several user-defined extension mechanisms through property sets and annex languages [14]:

- Property sets allow one to define custom properties to extend standard ones. This is the path taken by the "Data modeling annex document" that allows one to model precisely data types to be manipulated, or the "ARINC653 annex document" that defines patterns for modeling ARINC653 systems.
- AADL annex languages offer the possibility to attach additional considerations to an AADL component like behavioral specification. They bind a domain-specific language to components.

These extensions mechanisms are of particular interest to attach project-specific concerns to an architecture for further analysis such as electric power consumption, modeling of precise performance of buses, or error modeling. It is this combination of core and user-defined languages and properties that allows in-depth system analysis.

2.4 Modeling the FMS in AADL

The full model of the FMS uses AADLv2 core specifications and the ARINC653 Annex. Figure 2 gives the graphical view of the model made up of 4 modules connected through an AFDX network. We do not list the full textual models that spans over 770 SLOCs[1].

The model follows the initial specifications and AADL guidelines for ARINC653 systems : a module is a distinct `system` (containing a global `memory` and a `processor`) that hosts partitions (each is a `process`) bound to separate `memory` segments and `virtual processors` (representing spatial and temporal partitioning). `thread` components contained in partitions realize the avionics functions. Thanks to annex guidelines, we can model precisely the ARINC653 components and associated parameters (modules Major Frames, partitions durations, partitions and functions scheduling policies, etc.).

AADL does not provide specific guidelines for modeling AFDX networks. The AADL concept of `virtual bus` defines a connection supported in a `bus`. We use this concept to define AFDX virtual links. Switches are represented by `device` components bound to the virtual links. A dedicated property set has been defined to model parameters attached to virtual links, end systems and switches.

From this model, we may now consider further analysis of the full architecture. The current design could be used to validate a given architecture. Yet, the most challenging part is actually to guide the designer in finding a suitable definition of the architecture parameters in order to respect the constraints expressed in the model. This is detailed in the next section.

3 Analysis as part of the design process

We discussed how AADL allows to capture both the FMS functional and non-functional aspects as well as the IMA platform description. In this section, we first underline the difficulty of integrating those three dimensions in the same architecture model. Considering the Flight Management System, we then address how to derive AADL components and their parameters from constraints expressed in the model. For instance, we take the question of finding a suitable design for virtual links in AFDX networks that can be a difficult problem because of the interferences between VLs definitions. For further information, the issue is addressed in a more comprehensive setting in [15] and [16].

[1]The full AADLv2 textual model is part of the AADLib project, see `http://www.openaadl.org` for more details.

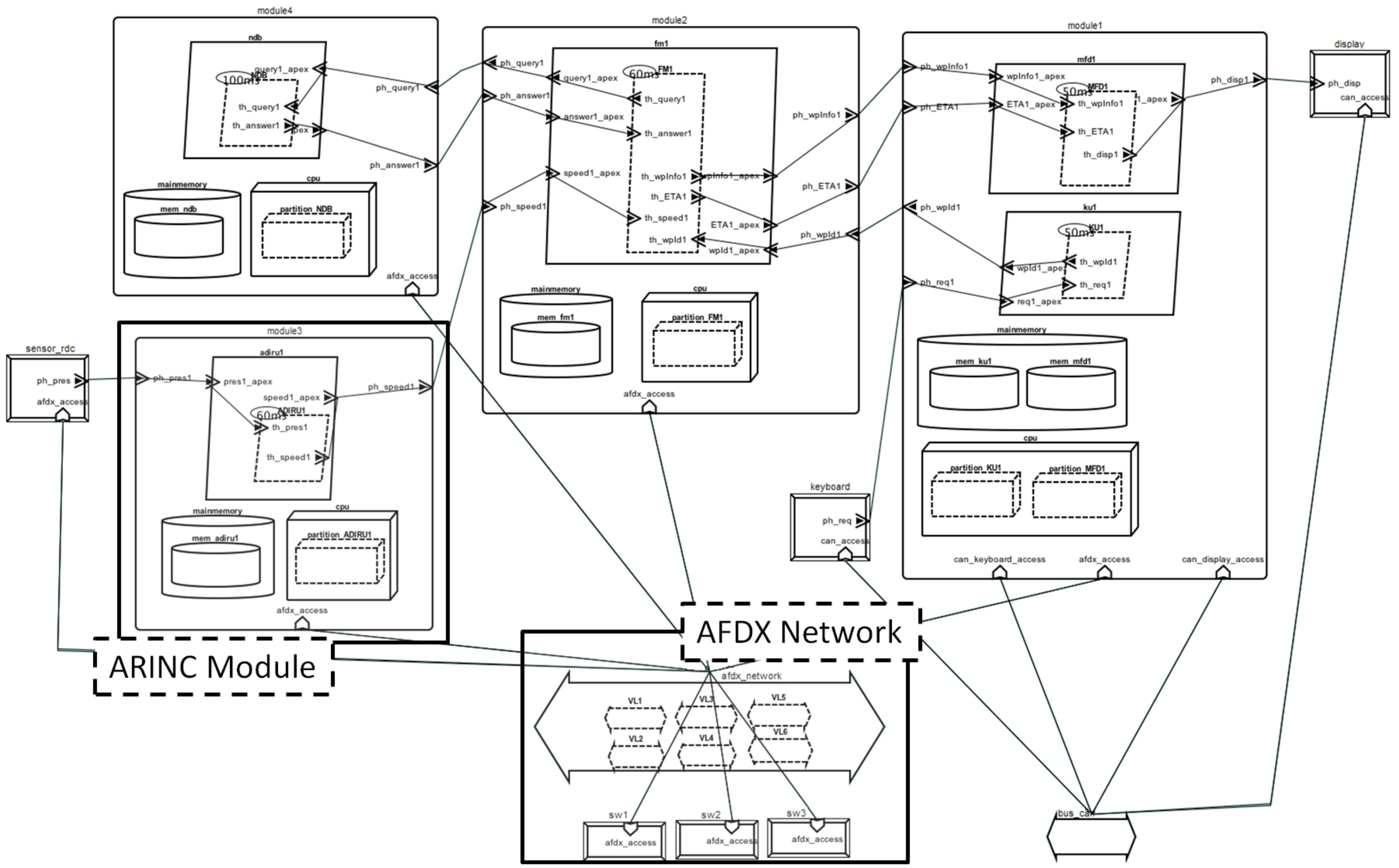


Figure 2: Overview of the FMS model in AADLv2. The model is made up of AADL *components* that describe the ARINC653 execution modules hosting the avionics functions and the AFDX network that supports the data exchanges. The full textual model is part of the AADLib project, see `http://www.openaadl.org` for more details.

3.1 Lessons learned : there are dependencies between modeling concerns

Figure 3 summarizes the three traits caught in the architecture model : the functions to realize, the hardware and software platform hosting the functions and the constraints to comply with. It implies that the architecture model components and the attached properties have to integrate and solve the dependencies between these views. In this subsection, we highlight some of the dependencies involved in a *real* IMA system that are "integrated" in its architecture model.

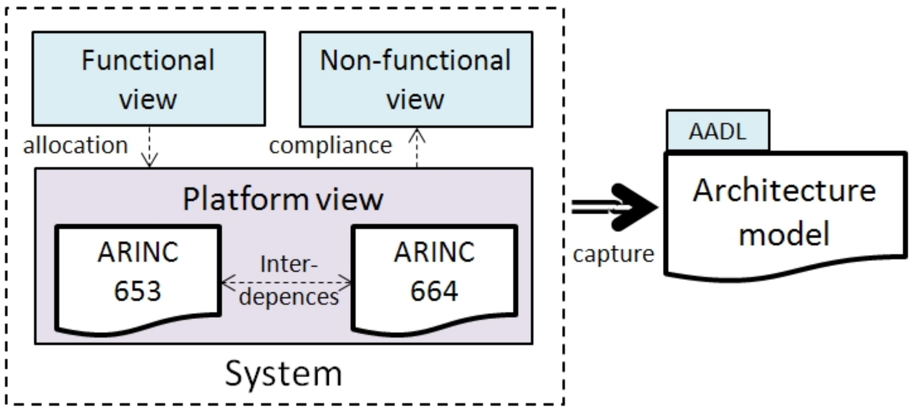


Figure 3: The architecture model captures jointly the system functional, non-functional and platform concerns and has to integrate the dependencies between these aspects.

Functions-platform allocation. The allocation "maps" the functional architecture to the platform, *i.e.* it defines how to implement the functions and communications within the ARINC653/ARINC664 execution platform:

- the MaF_m and MiF_m of a module are deduced from the periods (P_f) of hosted functions – there are, respectively, the *lcm* (less common multiple) of the periods and the shorter period,
- the duration ($D_{m,p}$) of a module partition corresponds to an upper bound of the time needed (C_f) to the related function (or sub-functions or threads) to execute all its instructions ($D_{m,p} \geq C_f$),
- the virtual links characterization (bag_v and $smax_v$) depends on several parameters :
  - the number of messages (n_f) sent during the cycle of a function,
  - the size of the messages ($m_{i,f}$, with $i \leq n_f$).

Platform components inter-dependencies. The allocation choices and interactions between platform components directly create temporal delays :

- the execution modules and their characteristics (scheduling policies, execution times, etc.) determines the functions response times,
- the configuration of the AFDX network components (choice of VLs parameters, topology and routing strategies) influences the traversal times,
- the interaction between the platform components (executions, communications) has a direct impact on delays belonging to functional chains.

```
system fms end fms;

system implementation fms.impl
        subcomponents           -- here are declared the fms' components
        -- modules and devices
        module1 : system subsystem::m1_system.impl;
        module2 : system subsystem::m2_system.impl;
        -- ... other modules and devices
        -- communication components
        afdx_network : bus fms_hardware::physical_afdx_link.impl;
        bus_can : bus fms_hardware::can;
        sw1 : device subsystem::afdx_switch;
        sw2 : device subsystem::afdx_switch;
        sw3 : device subsystem::afdx_switch;

        connections     -- connections and busses accesses
        nt_wpId : port module1.ph_wpId1 -> module2.ph_wpId1;
        --... other connections between modules
        flows           -- wpId, wpInfo, query, answer, speed flows
        wpId_fl : end to end flow module1.wpId_src ->
                                        nt_wpId -> module2.wpId_sink ;
        --... other data flows
        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;
        --... other latency constraints
        -- But how to define the bindings? For instance :
        -- Actual_connection_binding => (reference (afdx_network.??))
        --                                              applies to nt_wpId;
end fms.impl;

virtual bus VL end VL;                  -- VL is a subcomponent of afdx_network

virtual bus implementation VL.vl1       -- this is the definition of the VL1
        properties      -- but VL's properties (BAG and Smax) are missing...
        -- AFDX_properties::Bandwidth_Allocation_Gap => ?? ;
        -- AFDX_properties::Allowed_Message_Size => ?? ;
end VL.impl1;
```

Figure 4: The first AADL model ($m1$) contains the connections and the data flows between the FMS components. The bindings to support the connections and the AFDX virtual link implementations are missing. These parameters have to respect the latency constraints expressed onto the data flows.

Non-functional dependencies. Most of the time, the dependencies to non-functional aspects ”constrains” the allocation. For instance :

- a function execution may need the observation of a strict period or, conversely, may allow a jitter,
- a function execution must respect a soft or strict deadline (or no deadline at all),
- a communication between two specific tasks can be subject to timing constraints or not,
- the latency expressed upon a functional chain may be required as temporally bounded.

In the following, we show that it is possible to deal with the dependencies between the modeling concerns by performing relevant early-stage analysis. Particularly, we show how to define progressively the VLs parameters taking into account information in the model such as the constraints expressed upon the communications. From an evaluation perspective, it implies : 1) isolating model input parameters that can be combined to 2) propose a feasible solution which is 3) later assessed. Some parameters are mandatory, while other can be assumed. This is discussed in the following paragraphs.

3.2 Example : integration of the Bandwidth Allocation Gap (BAG) parameter into an incomplete AADL model

In the initial AADL model of the figure 4, we partly know the system definition : the functions hosted in the modules and their properties as well as the data exchanges between them. We also know the constraints of the system expressed onto the communications.

At this stage, dimensioning the BAG, which has a direct impact on the respect of timing constraints and the network load, may be a difficult task because the design space can be huge. Indeed, as this parameter ought to respect the formula BAG= 2^k [ms] with k integer in range 0 to 7 (see [12]), if we take as assumption that one VL is dedicated to one data flow, then there are $sol_{bag} = 8^f$ conceivable solutions, with f the number of flows.

To overcome this problem and complete the model, we execute the process pictured on Figure 5. We propose to:

1. use a *pivotal* Worst-Case Traversal Time evaluation (WCTT in the following) in order to identify the set of suitable BAGs,
2. use a *complementary* analysis method, relying on Network Calculus (NC in the following), to improve the results of the main WCTT evaluation.

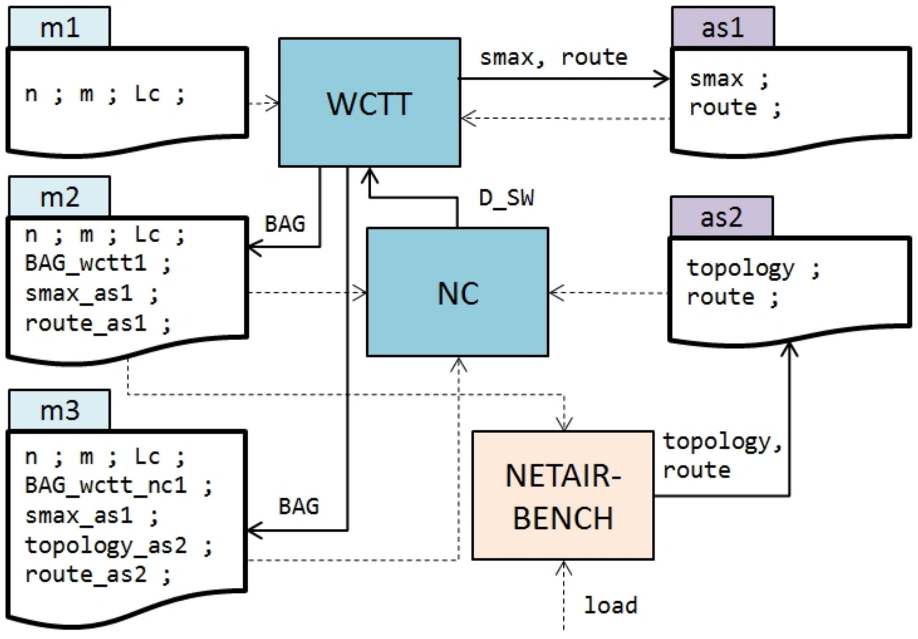


Figure 5: The BAG refinement process includes AADL models (blue-headed shapes), analysis and assumptions methods (respectively portrayed by green and brown rectangles) and assumptions models (purple-headed shapes). The dashed arrows are analysis inputs whilst the solid arrows stand for analysis results written in the models.

3.2.1 Worst-Case Traversal Time evaluation

Our *pivotal* Worst-Case Traversal Time analysis aims to assess the delay experienced by each data flow in the AFDX network (see Appendix A for more information). It amounts to add up the successive delays suffered by one sent frame throughout the traversed elements : from the source end system, through the successive switches, up to the sink end system(s). In short, the

WCTT evaluation gives the formula of an upper-bounded delay ($WL_{n,v}$) suffered by the last frame of the message n in the VL v as a function of several parameters (formulas 15, 16 and 17 of the appendix A). Hence, it is possible to compare the delay against the expressed latency constraint ($WL_{n,v} \leq LC_n$) :

$$bag_v \times (p_{n,v} - 1 + \sum_{k=1}^{n-1} p_{k,v}) + \Big(lag + 2 \times \frac{smax_v}{BW} \times (1 + r_v) + jmax\Big) + D_{sw\_v} \leq LC_n \tag{1}$$

$$\text{with} \begin{cases} D_{sw\_v} & = \sum_{k=1}^{r_v} WSCL_{n,k} \\ lag & = 2 \times WETeL + r \times WSTeL \\ sub_v - 1 & = 1 \text{ (sub-vl are not considered)} \end{cases}$$

and to calculate the suitable set of BAGs :

$$bag_v^{wctt} \leq \frac{LC_n - Dsw - \big(lag + 2 \times \frac{smax_v}{BW} \times (1 + r_v) + jmax\big)}{p_{n,v} - 1 + \sum_{k=1}^{n-1} p_{k,v}} \tag{2}$$

To figure out the bag_v^{wctt}, the model must contain :

- the maximal number of messages (n_f) that the sending function can pass to the virtual link at each execution,
- the maximal size of each message sent by the function – let m_n be the maximum value of all the messages sent by the function,
- the latency constraints expressed on the messages – in this paper, we assume a latency constraint expressed on the virtual link (LC_v), *i.e.* that is same for all the messages.

As the communication network is AFDX, the model must contain AFDX-specific parameters, defined in the standard, such as the bandwidth (BW), technological delays (lag) and a maximal transmission jitter at end systems outputs ($jmax$).

However, due to our ignorance of the whole system configuration, some properties may be still missing, in which case, we have to do assumptions :

- one virtual link is allocated to each data flow with the same source/receiver(s) couple,
- the $smax_v$ is set to :
  - $smax_v = m_v + 67$ bytes if $m_v \leq 1471$ bytes,
  - its maximum value $smax_v = 1538$ bytes else,
- all the messages can be fragmented, that means that $p_{n,v} \geq 1$ with $p_{n,v} = \lceil \frac{m_n}{smax_v - 67} \rceil$,
- if the routing strategy is missing, we assume that there is one crossed switch per VL : $r_v = Card(route_v) = 1$.

Using Equation 2, the WCTT evaluation gives a set of suitable BAGs for each VL (bag_v^{wctt}). Of course, the accuracy of the BAGs sets depends on the precision of the model and of the assumptions. In addition, at the first stage, there is no NC feedback, i.e. $D_{sw\_v} = 0$.

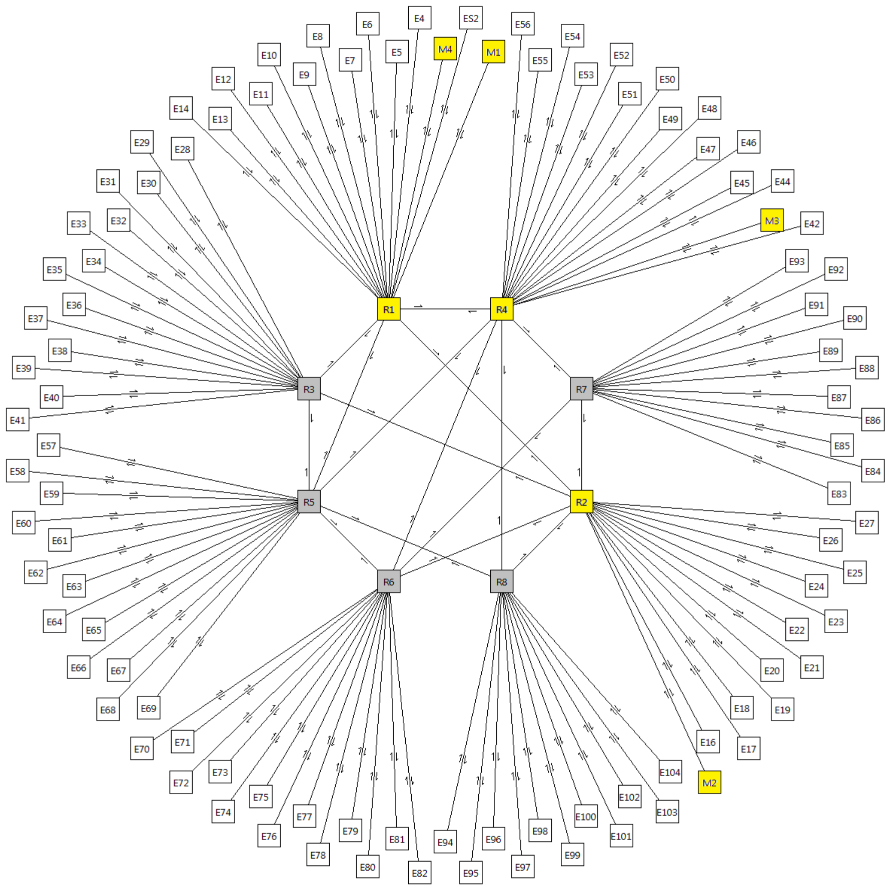


Figure 6: A typical AFDX topology can contain 100+ end systems and 8 switches [18]. The AFDX switches form the central backbone. The virtual links to study connect the yellow-colored components : the FMS functions are hosted into 4 modules connected to 3 different switches. The architecture and background traffic outside the FMS is generated by the NETAIRBENCH tool.

3.2.2 Network Calculus analysis and tool set

Network Calculus (NC) [17] is an algebra for computing and propagating constraints given in terms of envelops. NC is commonly used in avionics timing verification, and has been accepted in certification. The algebra handles incoming *flows* expressed by an *arrival curve* $\alpha(t)$ and *server* elements offering a minimal service specified through a *service curve* $\beta(t)$. Given $\alpha(t)$ and $\beta(t)$, at time t, it is possible to estimate the *backlog* – the amount of bits held in the network element – and the *virtual delay* – the delay suffered by a bit to cross the element. The worst delay experienced by a flow in a server is given by the greater horizontal deviation between the curves : $d = h(\alpha, \beta)$. Furthermore, in accordance with the input flows and the offered service expressions, the resulting output flow $\alpha^*(t)$ is given by $\alpha^*(t) = \alpha(t+d)$. Afterwards, it is possible to cascade the servers, *i.e.* to bind the output of one server to the input of one other, in order to propagate the data flow along its route and to compute the end-to-end delay ($D^{nc}_{sw\_v}$).

To perform the NC analysis, the model must detail the information needed to set the arrival curves belonging to each virtual link and the service offered by the end systems and switches :

- $\alpha_v(t)$ depends on the bag_v and the $smax_v$,
- $\beta_e(t)$ and $\beta_s(t)$ depend on : $smax_v$, BW, lag and $jmax$.

The $smax_v$, BW, lag and $jmax$ remain same as detailed for the WCTT analysis. Therewith, to obtain the least bandwidth-demanding but still correct VLs configuration, we first perform the analysis with the greater BAG for each previously refined BAGs set (BAG_v^{wctt}). To compute a precise analysis, in addition to the VL definition, the NC algebra uses :

- the network topology shaped by the execution modules, switches and links,
- the static routing table.

Either these features are exhaustively detailed in the input model or we can make assumptions on them. Particularly, we can combine model-defined parameters and assumed parameters. To use a combination of defined parameters (those of the FMS) and assumed parameters (those of the "artificial" surrounding system) has two interests : 1) it is possible to evaluate and refine the "real" input model within a realistic configuration; 2) it is possible to evaluate several routing strategies.

The latency analysis in this paper is performed using RTaW-Pegase, which is a commercial product implementing a state-of-the-art network calculus AFDX timing analysis (see `http://www.realtimeatwork.com/software/rtaw-pegase/`). We also use NETAIRBENCH, a module of RTaW-Pegase, which allows to generate avionics message sets according to user-defined parameters [18]. Figure 6 shows one instance generated by NETAIRBENCH where the FMS is included in an AFDX-realistic topology with a user-defined load.

3.2.3 Model refinement

We can see through the modeling and analysis flow (figure 5) that, as long as the required analysis inputs are present, the model is enhanced. The model refinements are done in line with : 1) the model evolution ($m1$, $m2$, $m3$), 2) the analysis methods outputs (BAG^{wctt} and D_{sw}^{nc}) and 3) the feasible assumptions ($as1, as2$).

The first execution of the pivotal analysis method takes as input the data of the incomplete model ($m1$) and the deduced assumptions ($as1$). Since we assume that one virtual link is allocated to identical data flows, there are five VLs for the FMS case study. The parameters and the results of the analysis are summarized in the table 2. The first coarse-grained WCTT evaluation reduces the space of BAGs solutions for each VL ($BAG_{vl_i}^{wctt_1}$) : $sol_{bag}^{m1} - sol_{bag}^{wctt_1} = 8^5 - 1440 = 31328$ solutions are discarded.

Virtual Link	n^{m_1}	$s_{max}^{as_1}$ (bytes)	LC^{m_1} (ms)	$BAG_{max}^{wctt_1}$ (ms)	BAG^{wctt_1} (ms)
VL_1	2	142	15	14,27456	1, 2, 4, 8
VL_2	3	692	15	7,04928	1, 2, 4
VL_3	2	192	10	9,25856	1, 2, 4, 8
VL_4	2	567	35	34,13856	1, 2, 4, 8, 16, 32
VL_5	2	567	20	19,13856	1, 2, 4, 8, 16

Table 2: The WCTT evaluation takes needed information in the model (m_1), makes feasible assumptions (as_1) for the missing parameters, and computes the maximal proper BAG. AFDX parameters (BW, lag, $jitter_{max}$) are set according to the standard, $r_{vl_i}^{as_1} = 1$ and $D_{sw\_vl_i}^{wctt_1} = 0$

Thanks to the first WCTT computation, the initial model (m_1) is enriched with a crucial missing parameter : the BAG (m_2). We are now able to perform the complementary NC analysis aiming at accurately evaluate the upper delay suffered in the switches for each VL ($D_{sw\_vl_i}^{nc_1}$). The topology is the one described by the figure 6 in accordance with an average utilization of switches ports set to 25%. The static communication routes follow a classical *shortest path* algorithm. Table 3 details the latency results ($D_{sw\_vl_i}^{nc_1}$) given by the NC analysis. This latency is passed as a refinement parameter to the WCTT method in order to precise the BAGs sets accordingly with the initial $LC_{vl_i}^{m_1}$ constraints. We can see that taking into account the calculated

$D^{nc_1}_{sw\_vl_i}$ reduces the set of eligible BAGs ($BAG^{wctt\_nc_1}$) for all the VL except VL_1 and VL_2 : $sol^{wctt_1}_{bag} - sol^{wctt\_nc_1}_{bag} = 1440 - 720 = 720$ solutions do not meet the latency constraints.

Virtual Link	$s^{as_1}_{max}$ (bytes)	BAG^{wctt_1} (ms)	$D^{nc_1}_{sw}$ (ms)	LC^{m_1}	r^{as_2}	$BAG^{wctt\_nc_1}$ (ms)
VL_1	142	8	2,774	15	2	1, 2, 4, 8
VL_2	692	4	2,922	15	2	1, 2, 4
VL_3	192	8	3,118	10	2	1, 2, 4
VL_4	567	32	2,774	35	2	1, 2, 4, 8, 16
VL_5	567	16	4,189	20	3	1, 2, 4, 8

Table 3: The NC analysis take into account the maximal packet size and the larger calculated $BAG^{wctt_1}_{vl_i}$ to compute the upper delay in the switches for each VL ($D^{nc_1}_{sw\_vl_i}$). This result is passed for a second WCTT analysis. Apart from the number of crossed switches ($r^{as_2}_{vl_i}$), the other WCTT inputs remains unchanged. The sets of suitable BAGs are precised to meet the initial latency constraints ($LC^{m_1}_{vl_i}$).

At the third iteration, the model ($m3$) contains the refined BAGs ($BAG^{wctt\_nc_1}_{vl_i}$). It is then necessary to calculate the delay suffered in the switches for each VL ($D^{nc_2}_{sw\_vl_i}$) and refine the BAGs sets ($BAG^{wctt\_nc_2}_{vl_i}$) if necessary. A new combined execution of WCTT and NC analysis shows : 1) the delay of each VL does not evolve ($D^{nc_2}_{sw\_vl_i} = D^{nc_1}_{sw\_vl_i}$) and 2) the computed BAGs subsets remain unchanged ($BAG^{wctt\_nc_1}_{vl_i} = BAG^{wctt\_nc_2}_{vl_i}$). Consequently, a fixed-point is reached : the model m_3 cannot be refined anymore against the Bandwidth Allocation Gap if the input parameters and assumptions stay stable.

4 Conclusions and perspectives

Our first contributions (Section 2) dealt with the architectural description of an avionics system with AADL, combining the *functional*, *non-functional* and *platform* concerns. We extended existing patterns dedicated to ARINC653 systems to also model AFDX networks.

We then addressed (Section 3) the definition and evaluation of the AADL model components and their properties. We showed that fixing one parameter upon a component can impact the overall avionics system definition because of dependencies among them. Actually, to deal with the architecture description amounts to solve the dependencies between the AADL model concerns. Starting from an incomplete AADL model, we implemented a process combining two analysis methods to evaluate and refine the Bandwidth Allocation Gap parameter. This process combines early and in-depth analysis to help the designer to narrow the design space.

The process used in Section 3 can be generalized by the figure 7 which shows a closed-loop approach :

1. the AADL model *captures* the system *functional*, *non-functional* and *platform* concerns within components defined by their *features* and *implementations*,
2. considering a facet of the system (*e.g.* functions executions, communications, etc.), relevant data are extracted and *converted* into parsable models,
3. depending on the point of interest, a pertinent method is used to *analyze* each model,
4. analysis outcomes are used to complete and *refine* the initial model.

We believe it is necessary to formalize the use of analysis methods along with modeling languages in order to tackle the early system architecture definition and evaluation. In a previous work, we defined REAL [19] – Requirement Enforcement and Analysis Language. It allows one

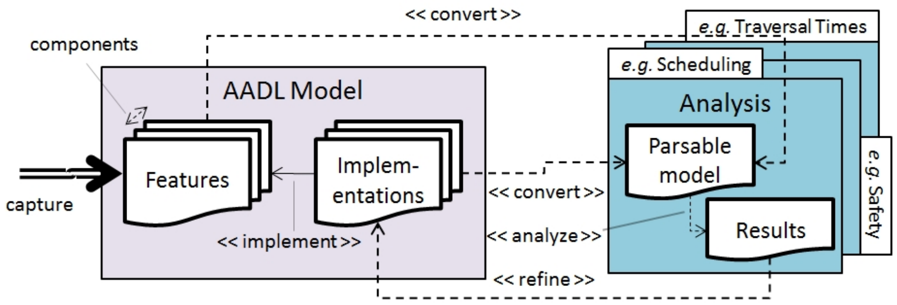


Figure 7: The AADL model details the system components *features* and *implementations*. Depending on the interest, relevant data are extracted from the model and analyzed following a pertinent methodology. Analysis outcomes allow to complete and gradually verify the model.

to define a set of predicates, and check whether a system satisfies them[2]. We plan to extend REAL to:

- define a library of predicates for *tools*. Such predicates would define conditions under which a given analysis becomes feasible. This would guide the designer to trigger analysis as early as possible in the design flow,
- detect and exploit relationships between analysis. The FMS case study illustrates that analysis parameters are mutually dependent. To guide the user, one needs some methodological support, embedded as part of the modeling framework to favor one analysis and define associated assumptions. Such relationships are actually higher-order predicates, indicating whether an analysis dominates another one.

The definition of those predicates, in the context of the FMS, is currently under implementation.

[2]This language is currently being standardized by SAE AS2-C committee

Appendices

A *Basics for AFDX Worst-Case Traversal Time evaluation*

This appendix gives the basics for evaluating the traversal time experienced by any message in an Avionics Full-Duplex Switched Ethernet (AFDX) network, i.e the elapsed time between the moment when the message is available for being transmitted through its dedicated virtual link and the moment when the message is available for being used by the receiving function(s). The traversal time evaluation consists in adding up the successive delays suffered by the last fragment (or frame) of the sent message throughout the traversed elements : from the source end system, through the successive switches, up to the sink end system(s). From the AFDX requirements and the definition of the network elements, it is then possible to derive the most unfavorable scenario under which such message suffers the Worst-Case Traversal Time (WCTT).

A.1 Introduction

Avionics Full-Duplex switched Ethernet. The ARINC664 standard [12] defines a predictable communication network called *Avionics Full Duplex-Switched Ethernet* (AFDX). It uses full-duplex links to carry the packets and switches to route a packet from a sender to one or several receiver(s). AFDX implements the core concept of *Virtual Link* (VL) to share the network bandwidth while maintaining the predictability of the communications. A VL is an unidirectional logical connection from one sender to one or several receiver(s) – unicast or multicast channels. Each VL has :

- a limited bandwidth according to two parameters : 1) the *Bandwidth Allocation Gap* (bag_v) (*i.e.* the minimum time elapsed between two transmissions of frames) and 2) the *maximal allowed packet size* ($smax_v$),
- a predefined and static *route* ($route_v$) crossing one or several switch(es).

Traversal Time evaluation. The Traversal Time evaluation aims to assess the delay experienced by each data flow in the AFDX network. It amounts to add up the successive delays suffered by the message frames throughout the traversed elements : from the source end system, through the successive switches, up to the sink end system(s). There are two kinds of delays : 1) incompressible delays and 2) contention delays. Incompressible delays (referred as IL) encompass technological delays and times needed to emit the frames bits on the physical link. End systems and switch(es) contention delays (respectively ECL and SCL) are more difficult to evaluate : there are due to the competitive access to the medium at both end systems and switches levels and depend on the overall network configuration. The total delay (TT) can be expressed by the formula :

$$TT = IL + ECL + \sum SCL$$

The Worst-Case Traversal Time considers the most unfavorable scenario and the resulting total delay ($WCTT$) experienced by a message through its dedicated virtual link. It is obtained when the last message frame suffers the worst delay in each traversed element.

The WCTT evaluation expressed in this report relies on :

1. an analysis of the AFDX communication stack and the succesive delays involved in the network components (end systems in emission, switches and end systems in reception) such as defined in the ARINC664 standard (Section A.2); this analysis neither takes into account prioritized traffics [20, 21] nor offsets [22],
2. an analytical expression of the Worst-Case Traversal Time that includes the delays expressed for each AFDX component (Section A.3).

A.2 ARINC664 standard : definition and requirement of AFDX latencies

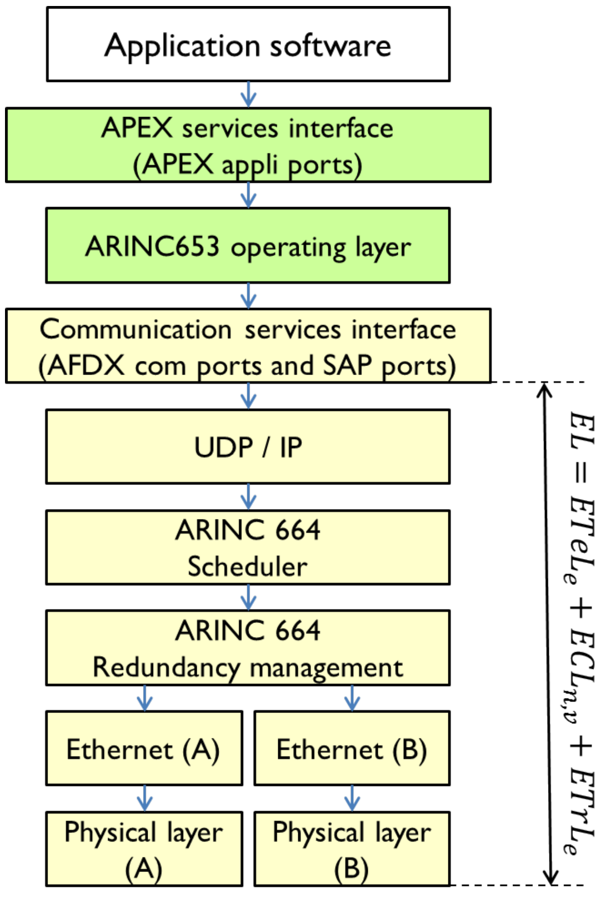
This section summarizes the latencies involved in the AFDX components : end systems (emitting and receiving) and switches. This summary rests on precise latencies definitions and requirements expressed in the ARINC664 standard [12] and derives the "parts" of the traversal time analytical expression dealt with in Section A.3. Definitions and requirements enumerated in this section are quotations from the ARINC664 standard document; as well, some figures are inspired/modified from the same standard.

A.2.1 End system latencies

Latency in emission (definition from [12]). *The latency [in emission] is defined as the duration between the following points of measurement as illustrated in [Figure 8].*

- *Start - last bit of an hosted partition data is available to the communication services of the end-system*
- *End - last bit of the corresponding Ethernet frame is transmitted on the physical media*

The **latency in emission** (EL) is the time needed by a message to go down the emission protocol stack. It encompasses a **technological latency** ($ETeL$), a **contention delay** (ECL) and a **transmission delay** ($ETrL$).



Application software
APEX services interface (APEX appli ports)
ARINC653 operating layer
Communication services interface (AFDX com ports and SAP ports)
UDP / IP
ARINC 664 Scheduler
ARINC 664 Redundancy management
Ethernet (A)
Ethernet (B)
Physical layer (A)
Physical layer (B)
$EL = ETeL_e + ECL_{n,v} + ETrL_e$

Figure 8: End system emission protocol stack – modified figure from [12]

Emission technological latency (definition from [12]). *[Emission] technological latency is defined as the time required to accept, process, and begin transmission of application data when the end system is performing no other task.*

Actually, the technological delay is the irreducible time needed to go through the sending end protocol stack, *i.e.* to cross all the “layers” of the emission protocol stack when there is no other frame in the stack. It includes ”execution times” of UDP-level, IP-level and MAC-level services as well as redundancy management.

Emission technological latency (requirement from [12]). *The technological latency of the end-system in transmission should be bounded and lower than 150 us [...].*

$$ETeL \leq 150us \tag{3}$$

Emission contention delay (definition from [12]). *The ”contention delay” is added to cover the time taken to deliver the frame to the physical layer.*

The contention phenomena are located in the *scheduler* ”layer” of the emission protocol stack (in the Figure 8) and are due to the *scheduling* stack pictured on Figure 9. It involves the following elements :

- the round-robin scheduler that empties the Sub-VL FIFO queues,
- the VL-regulator that processes the virtual link FIFO queue,
- the VL-scheduler that multiplexes the VLs handled by the end system in order to share the common physical link,

On the one hand, the contention delay suffered by a VL frame down to the physical link depends on the way on which the message will be fragmented. The fragmentation relies on :

- the maximal frame size fixed for the VL under consideration,
- the type of the used AFDX communication port (sampling or queuing).

Fragmentation (requirement from [12]). *(a) [In transmission,] sampling Com Ports should not use IP fragmentation, therefore the size of each sampling message should be less than or equal to the payload size of the associated Virtual Link.*
(b) [About Queuing Com Ports in transmission,] when fragmentation is used, the fragments should be transmitted in-order to the AFDX network.

Maximal AFDX frame size (requirement from [12]). *The End System should accommodate VL frames up to a size of 1518 bytes in both transmission and reception.*

$$smax \leq 1518 \text{ [bytes]} + headers \text{ [20 bytes]} = 1538 \text{ [bytes]} \tag{4}$$

Based on the message size (m_n) to send through the vl_v and following requirements A.2.1 and A.2.1, the number of packets ($p_{n,v}$) is given by Formula 5a for sampling ports and Formula 5b for queuing ports.

$$\text{sampling ports : } p_{n,v} = 1 \tag{5a}$$

$$\text{queueing ports : } p_{n,v} = \left\lceil \frac{m_n}{smax_v - 67} \right\rceil \tag{5b}$$

On the other hand, it is worth considering the VL settings :

- the maximal allowed packet size ($smax_v$, already defined),
- the bandwidth allocation gap (bag_v),
- the number of sub-virtual links (sub_v),
- the maximum transmission jitter ($jmax_v$).

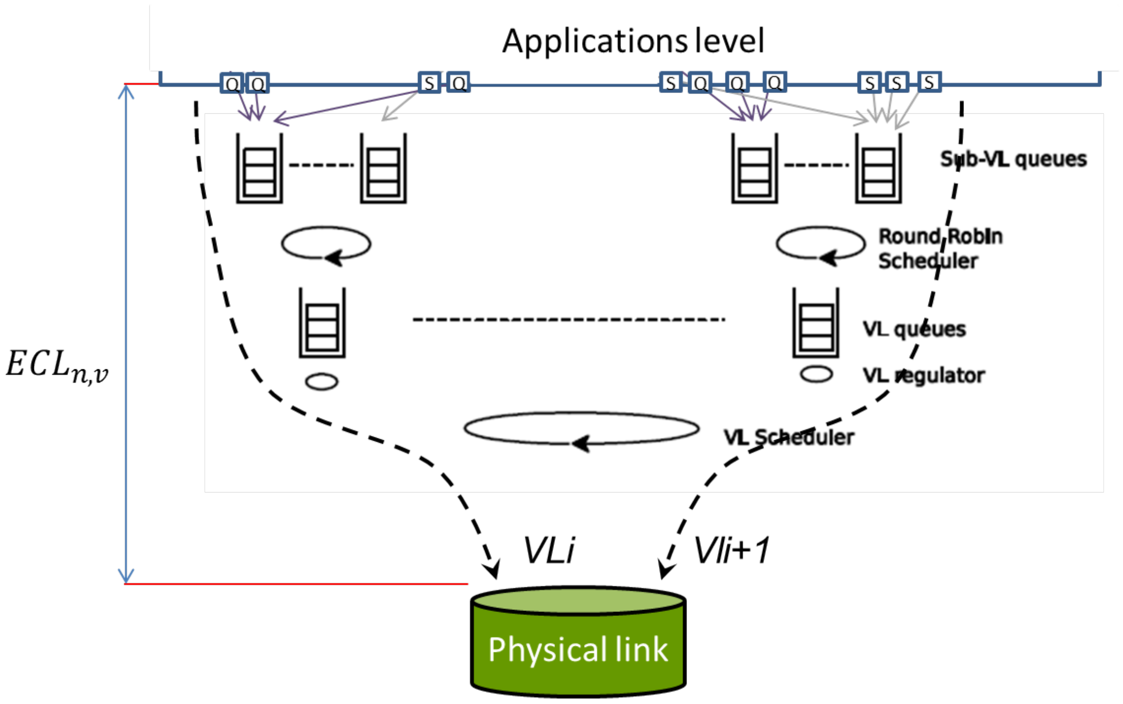


Figure 9: ARINC664 scheduling stack

Bandwidth Allocation Gap (requirement from [12]). *(a) The [VLs regulators] of the ES should be able to handle BAG values in range 1 ms to 128 ms. These values should satisfy [the formula 6].*
(b) On a per VL basis the traffic regulator [...] should shape the flow to send no more than one frame in each interval of BAG milliseconds.

$$bag_v = 2^k \ [ms] \ with \ k \ [integer] \in (0, ..., 7) \tag{6}$$

Sub-virtual links (requirement from [12]). *(a) A VL FIFO queue should be able to manage at most 4 Sub-VL FIFO queues.*
(b) Each Sub-VL FIFO queue should be read in round-robin sequence such that if any Sub-VL FIFO queue has traffic, one frame per BAG is sent to the main VL.

$$sub_v \leq 4 \tag{7}$$

Transmission jitter (requirement from [12]). *In transmission, the maximum allowed jitter on each VL at the output of the end-system should comply with both of the following formulas:*

$$jmax \leq 40 \text{ [us]} + \frac{\sum_{i \in \{\text{set of VLs}\}} smax_i \text{ [bytes]} \times 8 \text{ [bits/bytes]}}{BW \text{ [bits/s]}} \tag{8a}$$

$$jmax \leq 500 \text{ [us]} \tag{8b}$$

The worst contention delay experienced by a frame n of the VL v in the end system in emission is summarized by :

$$ECL_{n,v} = jmax_v + \underbrace{(p_{n,v} - 1) \times (sub_v - 1) \times bag_v}_{\substack{\text{time needed to process the last frame} \\ \text{of the message } n \text{ in last position} \\ \text{of the VL FIFO queue}}} + \underbrace{\sum_{k=1}^{n-1} p_{k,v} \times (sub_v - 1) \times bag_v}_{\substack{\text{time needed to process all the frames of} \\ \text{the messages enqueued ahead to the} \\ \text{message } n \text{ in the Sub-VL FIFO queue}}} \tag{9}$$

Finally, the end system transmission delay is added to cover the time needed to emit the frame bits on the physical link.

End system transmission delay (requirement from [12]). *[The MAC layer of the end-system should be able] to transmit frames back to back [at full frame rate of the medium].*

$$ETrL = \frac{smax_v}{BW} \tag{10}$$

Latency in reception (definition from [12]). *The latency in reception is defined between the following points of measurement [as shown in Figure 10] :*

- *Start - last bit of an Ethernet frame is received on the physical media attachment.*
- *End - last bit of the corresponding data is available to the end-system hosted partition.*

The **latency in reception** (RL) is the time needed by a message to go up the reception protocol stack. It only includes a **technological latency** ($RTeL$).

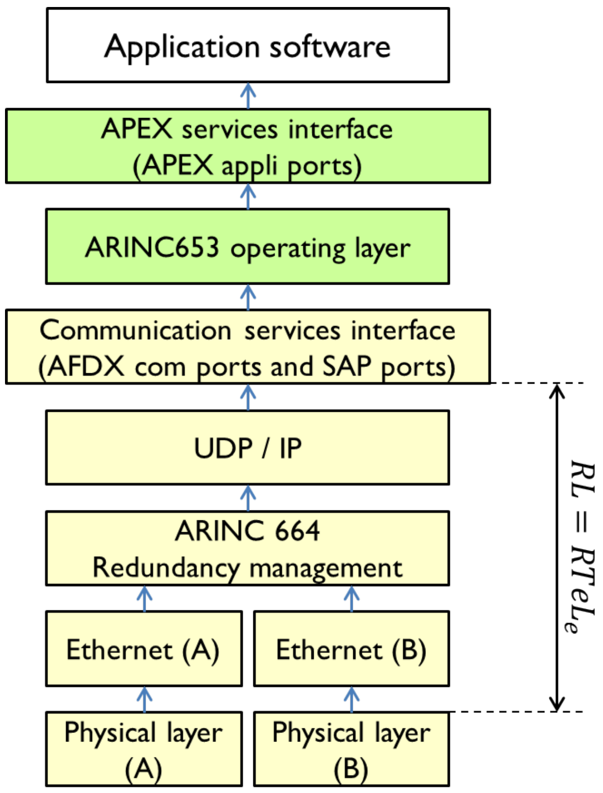

Application software
APEX services interface (APEX appli ports)
ARINC653 operating layer
Communication services interface (AFDX com ports and SAP ports)
UDP / IP
ARINC 664 Redundancy management
Ethernet (A)
Ethernet (B)
Physical layer (A)
Physical layer (B)
$RL = RTeL_e$

Figure 10: End system reception protocol stack – modified figure from [12]

The technological delay of the ES in reception is the irreducible time needed to execute the reception protocol stack (*i.e.* to cross all the “layers” of the reception protocol stack). It includes execution times of UDP-level, IP-level and MAC-level services as well as redundancy management.

Reception technological latency (requirement from [12]). *The technological latency of the end-system in reception should be bounded and lower than* $150us$.

$$RTeL \leq 150us \tag{11}$$

A.2.2 Switch latencies

Switch latencies (definition from [12]). *[Switch] latency is defined as being the elapsed time between the reception of the last bit of the frame until the transmission of the last bit of the frame [as shown in Figure 11].*

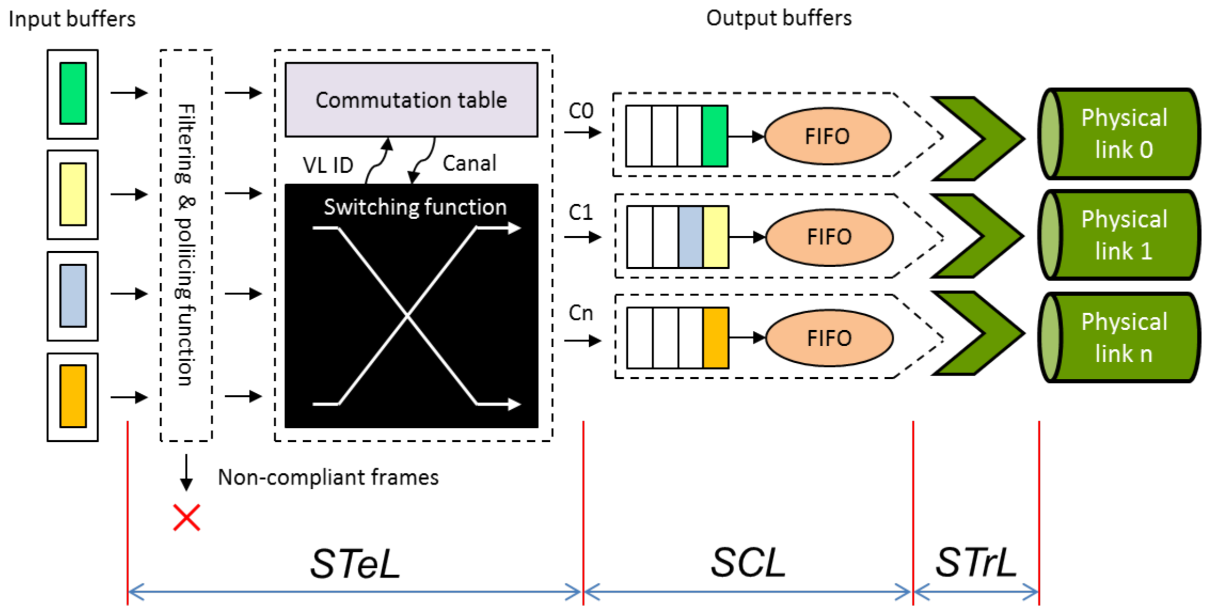


Figure 11: AFDX switch design

The switch latency (SL) is composed of three main parts:

- the **technological latency** ($STeL$) of switching function,
- the **contention latency** (SCL) due to interferences between packets crossing the same elements and competing to the same switches output ports,
- the **transmission latency** ($STrL$) required to emit the frame on the medium.

Switch technological latency (requirement from [12]). *The technological latency of the switch should be less than 100 us.*

$$STeL \leq 100us \tag{12}$$

Switch tranmission latency (requirement from [12]). *The MAC layer of each switch output port should be able to transmit frames at wire speed [100 Mbits/s].*

$$STrL_v = \frac{smax_v}{BW} \tag{13}$$

Switch contention latency (definition from [12]). *The contention in the switch and the store and forward capacity results in the need to buffer complete frames. [...] Data contention at the output ports is resolved by buffering.*

In the ARINC664 standard, there are no design constraints expressed upon the switches contention delays. However, delays suffered in output buffers of the switches obviously contribute to the traversal time. Hence, switches contention delays must be temporally bounded depending on the operational needs.

$$SCL_v \leq \mathcal{B} \tag{14}$$

A.3 AFDX (Worst-Case) Traversal Time Evaluation

Traversal time The traversal time of a message m_n corresponds to the elapsed time between the message *release time* and the message *reception time*. The *release time* is the moment when the message is written by the sending function in the emission AFDX port and available for being transmitted. The *reception time* is the moment when the message is written in the reception AFDX port and available for being used by the receiving function.

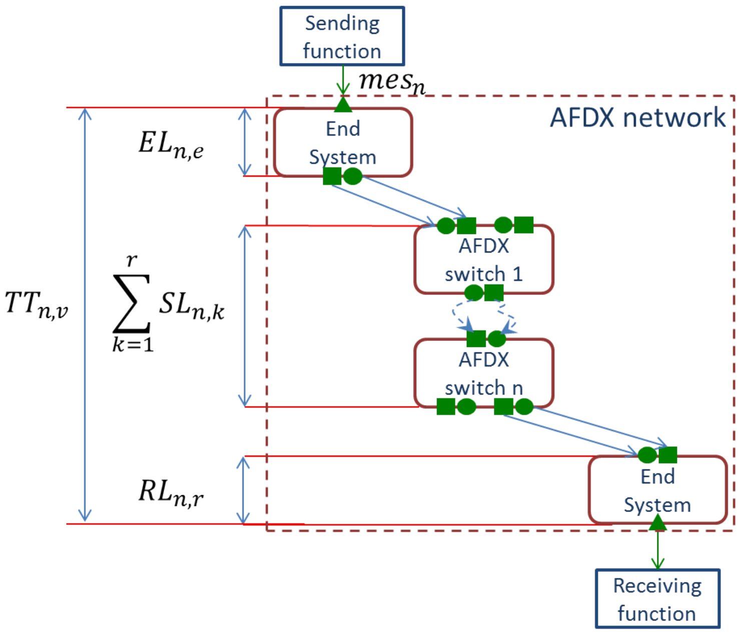


Figure 12: AFDX network – modified figure from [23]

The traversal time $TT_{n,v}$ experienced by a message mes_n through its dedicated virtual link vl_v is :

$$TT_{n,v} = EL_{n,e} + \sum_{k=1}^{r_v} SL_{n,k} + RL_{n,r} \tag{15}$$

where :

- $EL_{n,e}$ is the delay experienced in the sending end system ES_e, with $ES_e \in route_v$,
- $SL_{n,k}$ is the delay experienced in the switch SW_k, with $SW_k \in route_v$,
- $\sum_{k=1}^{r} SL_{n,k}$ is the sum of the delays experienced in the switches, with $Card(route_v) = r_v$,
- $RL_{n,r}$ is the delay experienced in the receiving end system ES_r, with $ES_r \in route_v$.

and, accordingly with the ARINC664 requirements :

- $EL_{n,e} = ETeL_e + ECL_{n,e} + ETrL_{n,e}$ (see Section A.2.1),
- $SL_{n,k} = STeL_k + SCL_{n,k} + STrL_{n,k}$ (see Section A.2.2)
- $RL_{n,r} = RTeL_r$ (see Section A.2.1)

It is possible to express the traversal time in terms of incompressible and congestion delays :

$$TT_{n,v} = IL_{n,e,k,r} + ECL_{n,e} + \sum_{k=1}^{r_v} SCL_{n,k} \tag{16}$$

where :

- $IL_{n,e,k,r}$ is the incompressible delay experienced along the network elements ES_e, SW_k and ES_r with $[ES_e, SW_k, ES_r] \in route_v$
- $ECL_{n,e}$ is the contention delay suffered in the sending end system ES_e, with $ES_e \in route_v$,
- $\sum_{k=1}^{r} SCL_{n,k}$ is the sum of contention delays experienced in the switches SW_k, with $SW_k \in route_v$ and $Card(route_v) = r_v$,

and :

- $IL_{n,e,k,r} = ETeL_e + ETrL_{n,e} + [\sum_{k=1}^{r} STeL_k + STrL_{n,r}] + RTeL_r$
- $ECL_{n,e} = jmax_v + (p_{n,v} - 1) \times (sub_v - 1) \times bag_v + \sum_{k=1}^{n-1} p_{k,v} \times (sub_v - 1) \times bag_v$

Worst-case traversal time The worst-case traversal time considers the most unfavorable scenario and the resulting delay $WD_{n,v}$ that can be experienced by a message mes_n using a virtual link vl_v. It is obtained when each term of the formula 16 is upper bounded. Let the letter W denote the upper bound of the involved terms :

$$WL_{n,v} = WIL_{n,e,k,r} + WECL_{n,e} + \sum_{k=1}^{r_v} WSCL_{n,k} \tag{17}$$

where :

- $WIL_{n,e,k,r}$:
  - $WETeL_e = WRTeL_r = 150us$ (see equations 3 and 11),
  - $WSTeL_k = 100us$ (see Equation 12), and
  - $WETrL_e = WSTrL_k = \frac{smax_v}{BW}$ (see equations 10 and 13); consequently :

$$\begin{aligned} WIL_{n,v} &= 2 \times WETeL_e + r_v \times WSTeL_k + (1 + r_v) \times WETrL_e \\ &= 0,3 + r_v \times 0,1 + (1 + r_v) \times \frac{smax_v}{100} \end{aligned}$$

- $WECL_{n,e}$:
  - $WECL_e = (p_{n,v} - 1) \times (sub_v - 1) \times bag_v + \sum_{k=1}^{n-1} p_{k,v} \times (sub_v - 1) \times bag_v$ (see Equation 9), and
    - $jmax_v = 500us$ (see Equations 8b),
    - bag_v, $smax_v$ and sub_v depend on the VL configuration,
    - m_n and m_k depend on the message(s) to process(es); consequently :

$$\begin{aligned} WECL_{n,v} &= jmax_v + (p_{n,v} - 1) \times (sub_v - 1) \times bag_v + \sum_{k=1}^{n-1} p_{k,v} \times (sub_v - 1) \times bag_v \\ &= 0,5 + (sub_v - 1) \times bag_v \times \left[\left(\left\lceil \frac{m_n}{smax_v - 67} \right\rceil - 1 \right) + \sum_{k=1}^{n-1} \left\lceil \frac{m_k}{smax_k - 67} \right\rceil \right] \end{aligned}$$

- $WSCL_{n,k}$:
  - defining $\sum_{k=1}^{r_v} WSCL_k$ is now well-mastered : several simulation-based and analytical approaches have been proposed to estimate delays suffered by AFDX frames along virtual link routes. Considering well-known analytical approaches, analysis based on Model-Cheking gives an exact worst case delay whilst Network Calculus and Trajectory approaches compute an upper bound of the worst-case delay. Since the establishment of fundamental theories, numerous works have been devoted to reduce the pessimism of the calculated upper bounds ; interested readers may consult [24] for more information.

References

[1] D. Redman, D. Ward, J. Chilenski, and G. Pollari. Virtual Integration for Improved System Design. In *Proceedings of The First Analytic Virtual Integration of Cyber-Physical Systems (AVICPS) Workshop*, San Diego, California, USA, November 2010.

[2] P. H. Feiler and D. P. Gluch. *Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language.* Addison-Wesley Professional, 1st edition, 2012.

[3] G. Lasnier, B. Zalila, L. Pautet, and J. Hugues. Ocarina : An Environment for AADL Models Analysis and Automatic Code Generation for High Integrity Applications. In *Proceedings of the 14th Ada-Europe International Conference*, Brest, France, June 8-12 2009.

[4] J. Delange, L. Pautet, A. Plantec, M. Kerboeuf, F. Singhoff, and F. Kordon. Validate, simulate, and implement ARINC653 systems using the AADL. In *SIGAda annual international conference on Ada and related technologies*, SIGAda '09, pages 31–44, New York, NY, USA, 2009.

[5] Y. Ouhammou, Grolleau E., Richard M., and Richard P. From Model-based design to Real-Time Analysis. In *The Fourth International Conference on Advances in System Testing and Validation Lifecycle (VALID)*, October 2012.

[6] M. González Harbour, J.L. Medina, J.J. Gutiérrez, J.C Palencia, and J.M. Drake. MAST: An Open Environment for Modeling, Analysis, and Design of Real-Time Systems. In *1st CARTS Workshop*, Aranjuez, Spain, October 2002.

[7] A. Khoroshilov, D. Albitskiy, I. Koverninskiy, M. Olshanskiy, A. Petrenko, and A. Ugnenko. AADL-Based Toolset for IMA System Design and Integration. *SAE International Journal of Aerospace*, 5:294–299, December 2012.

[8] J. Hansson, B Lewis, J. Hugues, L. Wrage, P.H. Feiler, and J Morley. Model-Based Verification of Security and Non-Functional Behavior using AADL. *IEEE Security & Privacy*, PP(99):1–1, 2009.

[9] M. Bozzano, A. Cimatti, J.-P. Katoen, V.Y. Nguyen, T. Noll, and M. Roveri. Safety, Dependability and Performance Analysis of Extended AADL Models. *The Computer Journal*, 54(5):754–775, May 2011.

[10] 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.

[11] Aeronautical Radio Incorporated. *ARINC Report 653P0 Avionics Application Software Standard Interface, Part 0, Overview of ARINC 653.*

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

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

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

[15] F. Frances, C. Fraboul, and J. Grieu. Using network calculus to optimize the AFDX network. In *European Congress on Embedded Real-Time Software*, Toulouse France, 2006.

[16] A. Al Sheikh, O. Brun, M. Chéramy, and P.-E. Hladik. Optimal design of virtual links in AFDX networks. *Real-Time Systems*, 49(3):308–336, 2013.

[17] J.-Y. Le Boudec. Application of network calculus to guaranteed service networks. *IEEE Transactions on Information Theory*, 44(3):1087–1096, 1998.

[18] M. Boyer, N. Navet, and M. Fumey. Experimental assessment of timing verification techniques for AFDX. In *European Congress in Embedded Real Time Software and Systems (ERTSS)*, Toulouse, France, February 1-3 2012.

[19] O. Gilles and J. Hugues. Expressing and enforcing user-defined constraints of AADL models. In *Proceedings of the 5th UML& AADL Workshop*, Oxford, United Kingdom, 22-26 March 2010.

[20] H. Bauer, J.-L. Scharbarg, and C. Fraboul. Applying trajectory approach with static priority queueing for improving the use of available AFDX resources. *Real-Time Systems*, 48:101–133, 2012.

[21] M. Boyer, N. Navet, M. Fumey, J. Migge, and L. Havet. Combining static priority and weighted round-robin like packet scheduling in AFDX for incremental certification and mixed criticality support. In *Proceedings of the 5th European Conference for Aeronautics and Space Sciences (EUCASS)*, Munich, Germany, July 1-5 2013.

[22] X. Li, J.-L. Scharbarg, and C. Fraboul. Improving end-to-end delay upper bounds on an AFDX network by integrating offsets in worst-case analysis. In *IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–8, 2010.

[23] J.J. Gutiérrez, J.C. Palencia, and M. González Harbour. Response time analysis in AFDX networks with sub-virtual links and prioritized switches. In *XV Jornadas de Tiempo Real*, Santander, Spain, January-February 2012.

[24] J.-L. Scharbarg and C. Fraboul. Methods and tools for the temporal analysis of avionic networks. In Joo Er Meng, editor, *New trends in technologies: control, management,computational intelligence and network systems*, chapter 21, pages 413–438. InTech - Open Access Publisher, http://www.intechweb.org, novembre 2010.