Heterogeneous models and analyses in the design of real-time embedded systems - an avionic case-study

Guillaume Brau
Université Fédérale Toulouse Midi-Pyrénées – ISAE-SUPAERO
10 Avenue E. Belin
Toulouse 31055, France
guillaume.brau@isae-supaero.fr

Nicolas Navet
University of Luxembourg, CSC Research Unit
6 Avenue de la Fonte
Esch-sur-Alzette L-4364, Luxembourg
nicolas.navet@uni.lu

Jérôme Hugues
Université Fédérale Toulouse Midi-Pyrénées – ISAE-SUPAERO
10 Avenue E. Belin
Toulouse 31055, France
jerome.hugues@isae-supaero.fr

ABSTRACT

The development of embedded systems according to Model-Driven Development relies on two complementary activities: system modeling on the one hand and analysis of the non-functional properties, such as timing properties, on the other hand. Yet, the coupling between models and analyses remains largely disregarded so far: e.g. how to apply an analysis on a model? How to manage the analysis process? This paper presents an application of our research on this topic. In particular, we show that our approach makes it possible to combine heterogeneous models and analyses in the design of an avionic system. We use two languages to model the system at different levels of abstraction: the industry standard AADL (Architecture Analysis and Design Language) and the more recent implementation-oriented CPAL language (Cyber-Physical Action Language). We then combine different real-time scheduling analyses so as to gradually define the task and network parameters and finally validate the schedulability of all activities of the system.

CCS CONCEPTS

• **Computer systems organization** → **Embedded systems**; **Real-time systems**; *Real-time languages*; • **Software and its engineering** → *Software notations and tools*;

KEYWORDS

Embedded systems, Model-Driven Development, Real-time

1 INTRODUCTION

Context. The development of embedded systems is a complex and critical task, especially because of the non-functional requirements. Embedded systems have indeed to fulfill a set of non-functional properties dictated by the environment and the application, in particular real-time constraints. For instance, missing a temporal constraint (e.g. deadline) in an avionic system can have hazardous consequences.

RTNS '17, October 4–6, 2017, Grenoble, France

ACM ISBN 978-1-4503-5286-4/17/10…$15.00
https://doi.org/10.1145/3139258.3139281

In order to develop real-time embedded systems, models can be used to first define the system and then analyze its real-time behavior. For example, languages such as AADL [1], EAST-ADL/AUTOSAR [2], SysML [3], MARTE [4], etc. provide various formalisms to describe real-time embedded systems. On the other hand, analytical frameworks for Verification & Validation activities include real-time scheduling tools [5, 6], model checkers [7], etc, in order to evaluate the fulfillment of real-time properties.

Problem. The modeling and analysis capabilities offered by integrated environments such as the AADL-based tool platform OSATE [8] are *de facto* limited. To extend their modeling and analysis capabilities, model transformation is the most-widely used technique. Typically, a model transformation translates a model used for design in a first tool into a model used for analysis in a third-party tool. For instance, several model transformations have been implemented from AADL models to real-time scheduling tools [9]. Yet, we note important limitations with this approach: multiple transformations to implement, difficulty to ensure the correctness of the transformation, etc. Importantly, a model transformation does not take into account the semantics of the analysis, and it does not answer questions such as: is the analysis applicable on the design model under study? What is the meaning of the analysis result? How to qualify the analysis (in terms of complexity, rapidity, precision, etc)? How to deal with the analysis process, for instance, is there a way to combine several analyses so as to prove a high-level property?

Contribution. In past contributions, we proposed a general approach to integrate the analysis of non-functional properties in Model-Driven Engineering for embedded systems [10]. Our approach is based on four application layers: (1) models to represent the system, (2) accessors to extract data from a model, (3) analyses to compute output data and/or properties from input data, (4) contracts to represent the analysis interfaces and orchestrate the analysis process.

The current paper presents an application of this approach in the real-time domain through an aerospace case study. The studied system is inspired by the Flight Management System case study [11] and the ROSACE case study [12]. We combine heterogeneous models and analyses so as to design and validate the avionic system. On the one hand, we use two modeling languages: AADL (Architecture Analysis and Design Language) and CPAL (Cyber-Physical Action Language, see [13, 14]). On the other hand, we combine different analyses so as to gradually define the task and network parameters

and finally validate the schedulability of all the activities of the system.

The experimental results presented in this paper can be reproduced from our tool prototype and models are available online[1].

Outline. The paper is organized as follows. We discuss related works in Section 2. Section 3 introduces our approach to combine heterogeneous models and analyses. We present the avionic system in Section 4. Section 5 explains the modeling of the system with the help of two different languages, AADL and CPAL, whereas Section 6 deals with the real-time scheduling analysis of these models. Finally, Section 7 concludes the paper.

2 RELATED WORKS

We distinguish between three types of approaches providing both modeling and analysis features: integrated environments, extensions based on model transformations and integration frameworks.

Integrated environments offer both modeling and analysis features in the same environment. These environments are built on diverse concepts and theories: the real-time scheduling theory, the synchronous approach, the cyber-physical systems approach, model checking, architecture description languages, etc. In this category, we can cite the standard MATLAB/Simulink [15] and SCADE [16] in the industry, the Ptolemy project in academia [17], or AADL-based environments OSATE [8] or MASIW [18], etc. We can also mention real-time scheduling tools (e.g. MAST [6] or Cheddar [5]) and various model checkers such as Spin [19] and Uppaal [7]. We note that the modeling and analysis capabilities provided by these environments are *de facto* restricted to a specific and closed environment.

The modeling and analysis capabilities can be extended through model transformations. In this case, a model used for design in a first tool must be transformed into a model used for analysis in another tool. For example, many transformations have been implemented to connect AADL models to analysis tools: real-time scheduling tools as MAST and Cheddar with the OCARINA tool suite [20] and MoSaRT [21]; model checkers as Uppaal [22], TINA [23]; dependability analyses [24]. A more exhaustive list of transformations applied to AADL models is available in a survey paper [9]. Yet, this approach suffers important limitations: multiple transformations to implement, difficulty to ensure the correctness of the transformation, etc. In addition, we observe that the semantics of the analysis is not central in any of these approaches (i.e. integrated environments and transformation-based extensions) and they do not answer questions as: is the analysis applicable on the design model which is considered? If not, is it relevant to execute a transformation? What is the meaning of the analysis result? How to qualify the analysis (in terms of complexity, rapidity, precision, etc.)? How to deal with the analysis process, for instance, is there a way to combine analyses so as to prove a high-level property?

The last class of works to which we aim to contribute seeks to define better and more powerful integrations between modeling and analysis techniques. We can cite three main initiatives that have inspired our works. MoSaRT (Modeling-oriented Scheduling analysis of Real-Time systems) [25] is an intermediate framework between real-time design languages and real-time analysis tools. MoSaRT consists of a Domain Specific Modeling Language providing the core concepts of real-time systems, and an Analysis Repository to analyze the models with the help of the real-time scheduling theory. A main novelty is the automatic selection of real-time scheduling analyses based on the notion of *real-time context* (i.e. a set of assumptions related to the tasks). Real-time contexts are then automatically checked on the models to choose a suitable analysis. Gaudel [26] builds on *architectural design patterns* to select real-time schedulability tests in the Cheddar tool. The authors define an architectural design pattern as a set of applicability constraints applying on architectural models. They implement their own algorithms to select the schedulability tests in the Cheddar tool. This algorithm aims at detecting the design patterns which are present in a model and analyze their composition if multiple patterns are represented. Ruchkin et al. [27] deal with the integration of analyses for Cyber-Physical Systems in the context of the OSATE/AADL tool environment. They acknowledge that properties of AADL models can be computed by tools coming from different scientific domains (e.g. schedulability, power consumption, safety or security). Thus, they firstly propose to use analysis *contracts* to capture the semantics of analysis domains. Then, at analysis time, they can verify the contracts through methods involving model checking and SMT solving so as to avoid the execution of conflicting tools (i.e. not to invalidate properties computed by a tool with one another).

In our integration approach [10], we aim at unifying these solutions in a more general framework and organize them according to a layered architecture: modeling, access, analysis and orchestration layers (see Section 3 for a more detailed presentation of the approach). Thus, previous related works can be seen as specific and separate implementations of these layers. Model transformation can be seen as a particular way to implement accessors, i.e. extract data from a model. The verification of real-time contexts, applicability constraints or contracts can be seen as a kind of *pre* and/or *post* analysis. And, at the topmost level, contracts enable to represent the analysis interfaces and then orchestrate the analysis process. In addition, we provide different implementations for these layers: heterogeneous languages (e.g. AADL and CPAL) to model the system, Python to support the whole prototyping and Alloy to formally define and evaluate the contracts.

3 OVERVIEW OF THE INTEGRATION APPROACH

This section introduces our approach to combine heterogeneous models and analyses. The approach is implemented in a tool prototype called MAIWEn (Modeling and Analysis Integration Workbench for the Engineering of embedded systems).

Figure 1 represents the four layers of the approach. The next paragraphs present the layers in a few words.

Models. Models enable to represent the system. Typically distinct models written in different languages provide different points of view on the system, e.g. architectural view, behavioral view, etc. In Section 5, we use two different modeling languages to represent an avionic system at different levels of abstraction: overall and operational architecture of the system in AADL, functional architecture of the applications in CPAL.

[1]The tool with the models can be found on the git repository https://github.com/gbra/maiwen

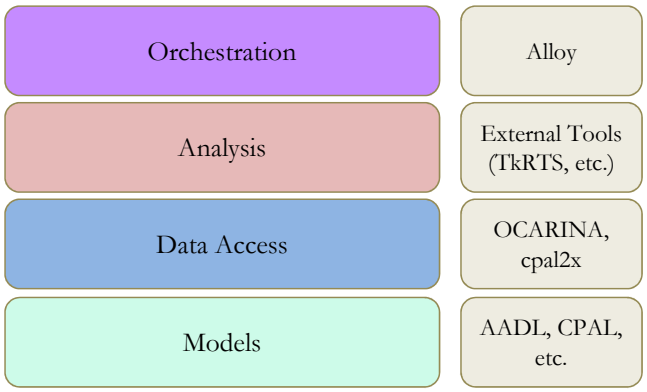

Figure 1: Modular and layered integration approach. The modules from modeling to orchestration are organized in layers. In our tool, these modules may use the resources in light gray.

Accessors. The interaction between models and analyses is managed via accessors. These accessors enable to extract data from a model in order to analyze them. The expected benefit is that an analysis can be associated to any kind of model as soon as an implementation of accessors to model internals is provided. This layer includes a *data model* and the *accessors*:

- the data model holds analysis-specific data about the system. For example, Figure 2 details a subset of the data model to describe real-time systems: tasks, processors, shared resources and scheduling algorithms are some of the data structures required to analyze timing behavior,
- accessors to model internals must then be implemented according to the mapping between the analysis data structures and the model concepts. In our tool, we use for example OCARINA to parse AADL models, and the `cpal2x` tool to extract data from CPAL source files.

Analyses. An analysis is a program to compute output data and/or derive properties from input data. Analyses can be performed using simulation, model checking or, more broadly, any analytical techniques (queuing theory, dependability analysis, etc). For example, Algorithm 1 details a simple procedure to compute the processor utilization factor of a set of tasks as for instance used in schedulability tests. This algorithm relies on the `Task` data structure in Figure 2.

In our tool, analyses are implemented using the Python programming language or we rely on existing specialized external tools. For instance, there are bridges to REAL, TkRTS, Cheddar, MAST, etc, through the OCARINA toolchain.

Orchestration. This module orchestrates the analysis process by taking into account three parameters: (1) input models, (2) the repository of analysis, and (3) the analysis goals. The orchestration relies on the core concept of contract introduced in [28]. A contract completely defines the interfaces of an analysis in terms of processed data and properties:

- Inputs/Outputs (I/O) describe input and output data,

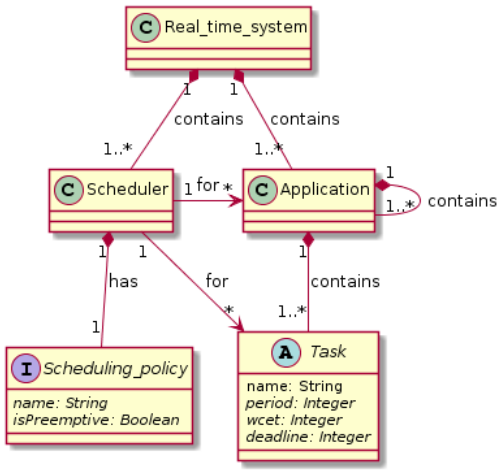

Figure 2: Subset of the data model from which the system load can be derived. Accessors enable to retrieve these data in a model (e.g. AADL) and analyze them in a program.

Algorithm 1: Compute the processor utilization factor of a set of real-time tasks

```
1 compute_processor_utilization_factor is
      Input  : Task[] list_of_tasks
      Output : Float utilization_factor
2     utilization_factor ← 0.0 ;
3     foreach task ∈ list_of_tasks do
4         utilization_factor ←
            utilization_factor + task.wcet/task.period
5     end
6     return utilization_factor;
7 end
```

- Assumptions/Guarantees (A/G) describe input and output properties.

Listing 1 describes a simple contract in the Alloy language [29]. This contract specifies the interfaces of the fixed-priority non-preemptive schedulability analysis `rts_periodic_np` coming from [30] in terms of the data model defined in Figure 2. For example, in input, the analysis requires a precise hierarchy of data structures which consists of a real-time system with a scheduling policy and tasks, and properties attached to the components, e.g. `periods` are required, `offsets` are *not* required.

We can then use methods to automate the reasoning about these interfaces, and answer complex questions about the analysis process: which analysis can be applied on a given model? Which are the analyses that meet a given goal? Are there analyses to combine to meet a high-level objective? Are there interferences between analyses? Etc. We use SAT resolution methods provided by the Alloy analyzer to derive an *analysis graph*. The analysis graph provides all the analysis paths to execute in order to fulfill a goal for an input model. The orchestration module finally visits the graph, and execute the analyses. The strength of the approach is that it

```
/* An example of analysis contract */
one sig rts_periodic_np extends Contract{}{
    input={S:Component |        S.type=system and
                                    (some sub:S.subcomponents | sub.type =processor and
                                          (scheduling_protocol+preemptive_scheduler) in sub.properties) and
                                    (some sub:S.subcomponents | sub.type =process and
                                          thread in sub.subcomponents.type and
                                          ( let th=sub.subcomponents & thread.~type |
                                                (dispatch_protocol +period +compute_execution_time +priority+deadline) in th.properties
                                                    and
                                                (not (offset) in th.properties)
                                          )
                                    )
            }
    output={thrd:Component|     thrd.type=thread and
                                thrd.subcomponents=none and
                                    thrd.properties=response_time
            }
    assumption=mono_processor+periodic_tasks+no_jitter+implicit_deadlines+independent_tasks+fixed_computation_times+non_self_suspension+
          no_preemption+no_overheads+fixed_priority
    guarantee=is_schedulable
}
```

Listing 1: Specification of an analysis contract in Alloy. `Input/output` fields are defined with respect to the data model. Here, the analysis takes as inputs a system with a scheduling policy and tasks; with properties attached to the components, e.g. `periods` are required while `offsets` are *not* required.

ensures that if there is an analysis path that can fulfill a goal, it will necessarily be found (see [28]).

Section 6 presents an application of this approach to orchestrate the schedulability analyses for an avionic system. The analysis graph in Figure 7 involves different analyses (WCET, task scheduling, communication delays, simulation, etc.) so as to gradually define the task and network parameters and finally validate the schedulability of the system.

4 AVIONIC SYSTEM

This section presents the avionic system that we study in this article. It also introduces the Integrated Modular Avionics (IMA) platform that hosts the avionic functions.

4.1 Application

The avionic system comprises a Flight Management System (FMS) [11, 31] and a Flight Control System (FCS) [12, 32].

Flight Management System. The primary task of a Flight Management System is in-flight management of the flight plan. The Flight Management System uses values measured from various sensors to compute the flight plan during flight and guide the aircraft. The crew interacts with the FMS by means of a Multi-Function Control and Display Unit (MCDU).

Figure 3 describes the functional architecture of the Flight Management System. This system is made up of five main functions. The *Keyboard and cursor control Unit (KU)* handles requests from the crew while the *Multi Functional Display (MFD)* displays data from the flight plan such as *waypoints* or the *Estimated Time of Arrival*. The *Flight Manager (FM)* computes the flight plan by querying static data (waypoints, airways, etc.) from the *Navigation Data Base (NDB)* and dynamic data (altitude, speeds, position, etc.) from the *Air Data Inertial Reference Unit (ADIRU)*.

Flight Control System. The Flight Management System also interfaces with several other avionic systems in order to accomplish these functions, in particular the Flight Control System. The aim

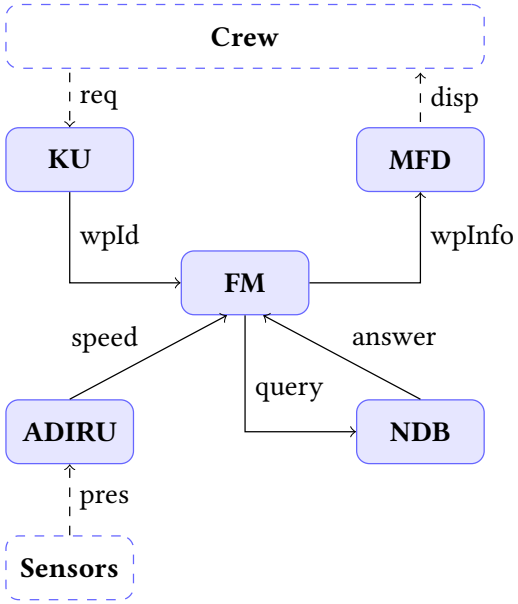

Figure 3: Functional architecture of the Flight Management System. The functional architecture shows the set of functions and the data flows among the functions.

of the FCS is to control the altitude, the speed and the trajectory of the aircraft from the flight plan [32]. In this paper, we model the functional architecture coming from the ROSACE (Research Open-Source Avionics and Control Engineering) case study [12] that implements a longitudinal flight controller.

4.2 Integrated Modular Avionics

The *functions* are executed on an *Integrated Modular Avionics (IMA)* platform. The IMA defines the use of the hardware and software resources with two main standards:

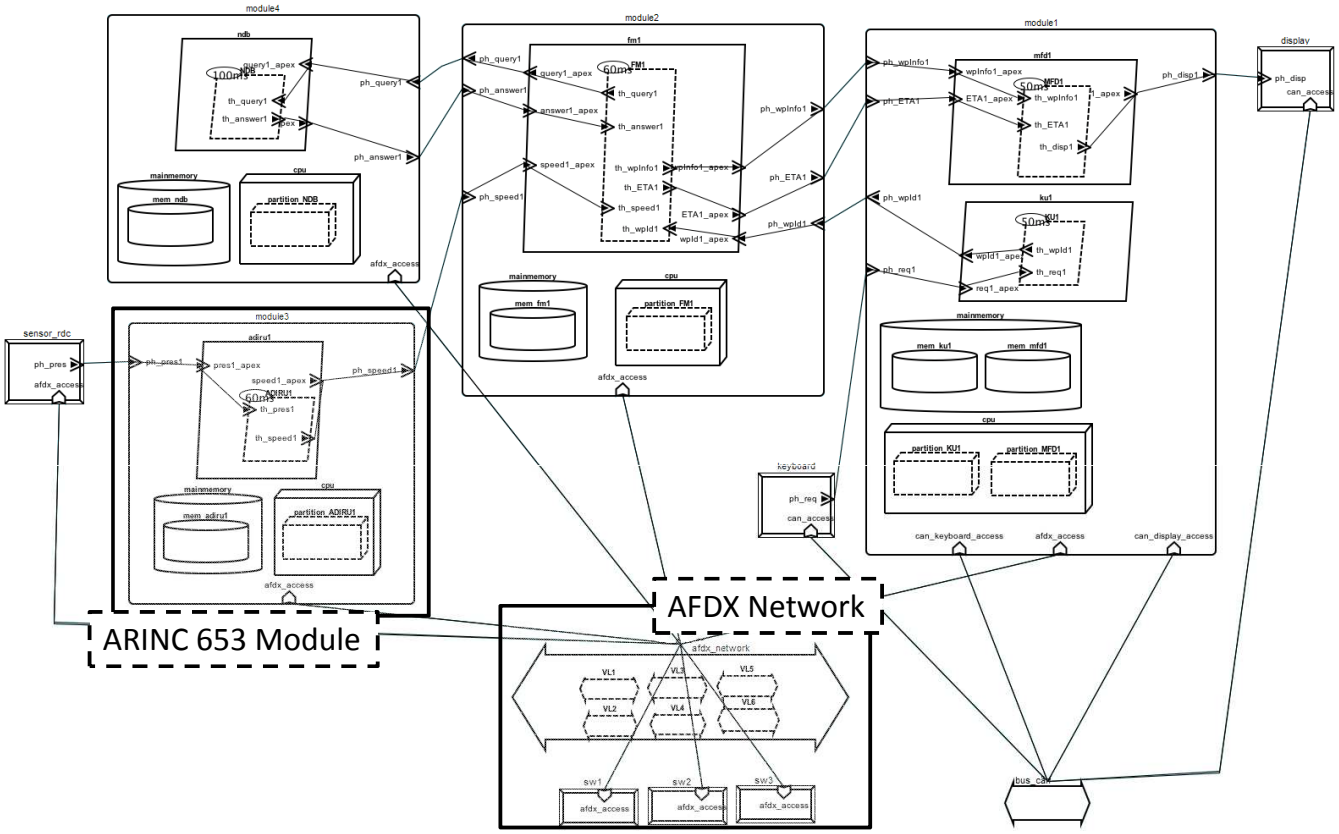

Figure 4: Overview of the operational architecture of the Flight Management System in AADLv2. AADL components represent the ARINC653 calculators and the AFDX network.

- ARINC653 [33] for computational resources,
- ARINC664 (part 7) [34] for communication resources.

One important objective of the IMA is to ensure timing predictability. In the following, we review some important concepts of its two core standards. This description emphasizes the parameters that are to be modeled and analyzed in this article.

Calculators – ARINC653. The ARINC653 is a standard to share processing and memory resources between several functions in a hardware *module*, or calculator. According to ARINC653, each function is hosted in a specific *partition* with a statically defined strict access to processing and memory resources:

- *temporal* partitioning ensures that partitions are executed during specific time slots,
- *spatial* partitioning guarantees that each partition has a reserved memory space.

Hence, an ARINC653 schedule is both static and cyclic. Partitions are scheduled according to several parameters:

- at module level: a *major time frame* is defined for each module (MAF_m); possibly, a minor cycle can also be defined (MIF_m).
- at partition level: an *offset* ($O_{m,p}$) that is the delay between the MAF_m origin and the start of the partition execution; and a *duration* ($D_{m,p}$) that is the time allocated to each partition to access the processor.

During the major cycle, each partition is scheduled once or several times. This major cycle is then repeated indefinitely. In a partition, a function is implemented through one or several processes. These processes are scheduled at the partition level according to a specific scheduling algorithm (e.g. FIFO or NPFP).

Networks – ARINC664. This standard defines a predictable communication network based on Ethernet called *Avionics Full Duplex-Switched Ethernet* (AFDX). It uses full-duplex links to transmit the packets and switches to route packets from a source to one or several destinations. AFDX defines the core concept of *Virtual Link* (VL) to share the network bandwidth between the data flows. A VL is a unidirectional logical connection from one sender to one or several receiver(s) (i.e. unicast or multicast VLs). In particular, each VL has:

- a limited bandwidth (ρ_v) according to two parameters: the *Bandwidth Allocation Gap* (bag_v) that is the minimum time interval between two successive transmissions of frames of the same flow; and the *maximal allowed packet size* ($smax_v$); $\rho_v = \frac{smax_v}{bag_v}$,
- a predefined and static *route* ($route_v$) crossing one or several switch(es).

5 ARCHITECTURAL AND BEHAVIORAL MODELING

In the following, the architecture of the system, and a subset of its functional behavior, is modeled with AADL and CPAL.

5.1 Operational architecture in AADL

We represent the highest-level operational architecture of the avionic system in AADL.

AADL: the Architecture Analysis and Design Language. AADL is an architecture description language dedicated to "the specification, analysis, automated integration and code generation of real-time performance-critical (timing, safety, schedulability, fault tolerant, security, etc.) distributed computer systems"[2]. AADL is an SAE International standard [1]. AADL originates from the former MetaH language [35] and has been improved and revised several times[3].

AADL is a textual language first, but also has a graphical representation. It represents both the static and dynamic architecture of a system:

- the static architecture consists of a hierarchy of interacting software and hardware components,
- the dynamic architecture describes operational modes, connection configurations, fault tolerant configurations, behaviors of individual components, etc.

AADL model. The Flight Management System (FMS) is first specified in AADL. The model uses AADLv2 core specifications and the ARINC653 Annex. Figure 4 shows a graphical view of the model. The model includes four ARINC653 calculators to host the avionic functions connected through an AFDX network[4].

The model follows the FMS specifications and the AADL design patterns for ARINC653 systems: each module is a distinct `system` with a global `memory` and a `processor`. A module hosts partitions modeled as `processes`. Each partition has its own `memory` segment and must be executed on a `virtual processor`. Finally, `thread` components contained in partitions realize the avionic functions. The ARINC653 Annex guidelines provide support for the precise modeling of the ARINC653 components and associated parameters (e.g. major time frames for modules, durations of partitions, scheduling policies within the partitions).

AADL on the other hand does not provide specific support for modeling AFDX networks. The AADL concept of `virtual bus` defines a connection supported in a `bus`. We rely on this concept to define AFDX virtual links. Switches are represented by `device` components bound to the virtual links. A dedicated property set is defined to model parameters attached to virtual links, end systems and switches: bandwidth allocation gaps, transmission jitters, technological delays, etc.

[2]http://www.aadl.info/ accessed September 2016

[3]AADLv2.1 is the latest version to date, from September 2012. AADLv2.2 and AADLv3 are in the planning stage.

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

5.2 Functional architecture in CPAL

A functional description of the calculators completes the highest-level operational architecture. We model the functions (i.e. processes) of the Flight Control System with the CPAL language. The CPAL models of the FCS come from [36].

CPAL: the Cyber-Physical Action Language. CPAL is a Domain-Specific Language (DSL) to model, simulate, verify and program Cyber-Physical Systems [13, 14, 37]. The language is inspired by the synchronous programming approach [38], Promela [19] and time-triggered architecture description languages as Giotto [39] and Prelude [40]. The syntax of CPAL is close to the syntax of the C language but provides abstractions specific to embedded systems, especially for non-functional concerns specified via an internal DSLs, together with a formal execution semantics. In addition, CPAL is a real-time execution engine: CPAL models are *interpreted* with the property that a model will have the same behavior in simulation mode on a workstation and in real-time mode on an embedded target.

CPAL model. Figure 5 shows the functional architecture of the FCS in the CPAL graphical representation. The functional architecture describes the processes, their activation pattern and the data flows among them. For instance, the process `az_filter` executes at a rate of 100Hz (i.e. $T_{az\_filter} = 10ms$), computes, from input variables `Az_Filter_Conf` and `az`, an output variable `az_meas` that is used by another process named `vz_controller`.

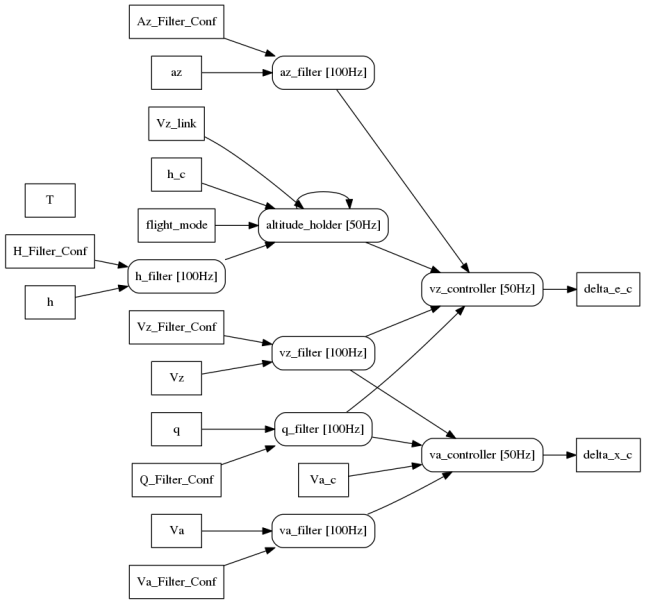

Figure 5: Functional architecture of the flight controller in CPAL. The functional architecture describes the processes, their activation pattern and the data flows among them.

The CPAL model describes the logic of each process under the form of a Finite-State Machine (FSM) with conditional and timed transitions between states. For example, the FSM in Figure 6 implements two distinct functioning modes for the `altitude_holder`

process: `Manual` and `Auto`. The operations in each state are specified in a textual syntax close to C but providing higher-level constructs such as *loop over* to iterate over a collection, and communication channels that can be found in Promela (see [41] for a comparison a both languages).

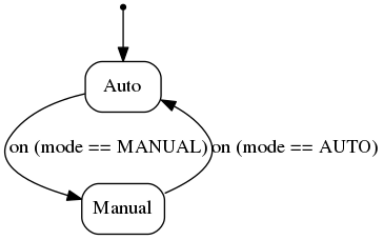

Figure 6: Finite-State Machine to describe two running modes of the `altitude_holder` process: `Manual` and `Auto`.

6 REAL-TIME SCHEDULING ANALYSIS

We use the approach introduced in section 3 to analyze the schedulability of the avionic system from the AADL and CPAL models presented in Section 5. Throughout the process, several parameters of the avionic system are derived so as to meet the timing constraints expressed at tasks and network levels.

6.1 Analysis graph

The analysis graph in Figure 7 is automatically generated from the contracts using their Alloy specification. The graph depicts the analysis process that will enable us to check whether the avionic system, as specified with the AADL and CPAL models (respectively `aadl_model` and `cpal_model` nodes in the graph), can respect the timing constraints (`isSched` node in the graph).

The analysis graph comprises three analysis flows that converge towards the same goal:

(1) the right-hand analysis flow, starting from the `cpal_model`, enables to check the schedulability of the tasks described in CPAL models, which are part of the ARINC653 processes to be represented in AADL,
(2) the analysis flow appearing in the middle uses task parameters defined in the CPAL model to define the ARINC653 parameters in the AADL model. Then, the ARINC653 parameters must be validated,
(3) the left-hand analysis flow, starting from the `aadl_model`, includes several analyses in order to iteratively define parameters of the AFDX network and finally validate them,

In the following sections, we explain the analysis steps in greater depth and provide experimental results.

6.2 From the analysis of CPAL processes to the definition of ARINC653 modules

This first analysis flow aims at fully validating the timing behavior of the software, that is to verify that all the processes will meet their deadlines at run time.

❶ *WCET analysis.* The first step is to measure the Worst-Case Execution Times experienced by the CPAL processes on several target platforms with the help of the CPAL-interpreter option `--stats`. In particular, we measure the WCETs on two execution platforms:

- *Embedded Linux 64-bit*: a laptop with a processor Intel Core i7-4710HQ @2.50GHz (4 cores), 7895 MiB of RAM, and running under Ubuntu 14.10 operating system,
- *Raspberry Pi*: a single-board embedded computer Raspberry Pi 2 - Model B V1.1 with a ARM Cortex-A7 processor @900MHz (4 cores), 1 GiB of RAM, and running under Raspbian operating system.

Table 1 summarizes the WCET measured on the Raspberry Pi platform in each running mode of the Flight Control System, i.e. the *vertical speed*, *airspeed* and *climb* modes. The WCETs measured on an Embedded Linux platform can be found in [10].

Process	WCET (µs)		
	Vertical Speed	Airspeed	Climb
va_filter	498.210	241.769	259.894
vz_filter	188.797	252.915	192.916
q_filter	440.518	218.801	209.739
az_filter	3402.323	371.920	190.832
h_filter	543.221	303.957	238.227
altitude_holder	162.448	164.531	262.551
vz_controller	194.634	263.957	216.561
va_controller	208.125	232.967	241.405

Table 1: WCET measured on a Raspberry Pi platform (*wcet_analysis*) in the different running modes of the Flight Control System: *vertical speed, airspeed* and *climb* modes.

Although this has not been done in this study, it is possible on the basis of the measurements to provision for a safety margin, typically using probabilistic arguments [42]. This margin can for instance account for cache latencies linked to the scheduling which have not been considered here. Another option is to make use of an analytic WCET analysis, generally considered as safer than measurement-based techniques although much more conservative.

❷ *Schedulability of the CPAL processes.* In a second step, we evaluate the schedulability of the FCS processes in each functioning mode, and analyze it with two non-preemptive scheduling policies available in CPAL: First-In First-Out (FIFO) and Non-Preemptive Fixed Priority (NPFP) (see the corresponding paths on Figure 7). Schedulability analysis is performed with two different techniques depending on the policy: simulation for FIFO (i.e. `cpal_simu`) and a schedulability test based on response time analysis for NPFP (i.e. `rts_periodic_np`). Simulation cannot in general be used as a schedulability test for non-preemptive policies because of scheduling anomalies [43]. This is however not the case for FIFO which is sustainable with respect to execution times [44] and simulation provides an exact test for strictly periodic tasks with offsets.

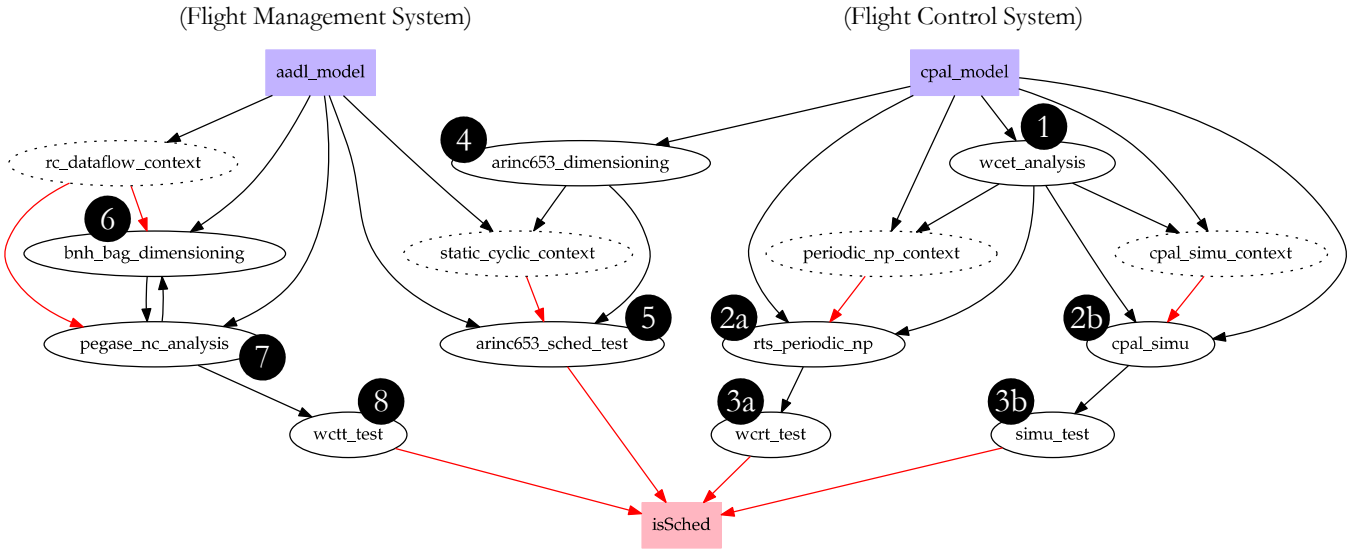

Figure 7: Analysis graph for the avionic case study. The graph shows the analysis process to check the schedulability of the system (`isSched` goal) according to input `aadl_model` and `cpal_model`. Black arrows conveys data, red arrows involve properties.

(a) Simulation with FIFO. A CPAL model can be simulated so as to evaluate the timing behavior of the software. The CPAL simulator uses the following data from the CPAL model:

- the scheduling algorithm,
- the task activation model that usually consists of a few tasks' parameters, e.g. periods and offsets, although arbitrary complex activation patterns are possible,
- timing annotations that may be execution times, jitters, priorities or deadlines.

The processes execute in zero time when the code is not annotated. Timing annotations defined within a `@cpal:time` block specifies the timing behavior that a CPAL program must have at run-time. We thus inject the WCETs previously measured as timing annotations in the CPAL model in order to achieve a simulation that is timing accurate with respect to the WCETs.

Figure 8 shows the simulation result of the CPAL model of the flight controller in the *Vertical Speed* mode. The figure describes the schedule of the processes according to their periods, offsets (which are null here) and the FIFO scheduling policy. Each rectangle represents a process execution and the thickness of the rectangle denotes the process execution time. According to FIFO, the processes are executed in the exact order of their activation. In Figure 8, we observe from a simulation over more than 2 LCMs of the periods that the system meets the scheduling constraints: (1) the process activation respects the periods; (2) only one process is scheduled on the processor at the same time; (3) all the processes complete before their deadlines, i.e. before the activation of the next job.

(b) Schedulability tests with NPFP scheduling policy. We use a schedulability test based on response time analysis to evaluate the schedulability of the FCS processes under a NPFP scheduling policy.

We evaluate the task response times from the CPAL model with the help of the TkRTS tool [30]. For example, Table 2 shows the

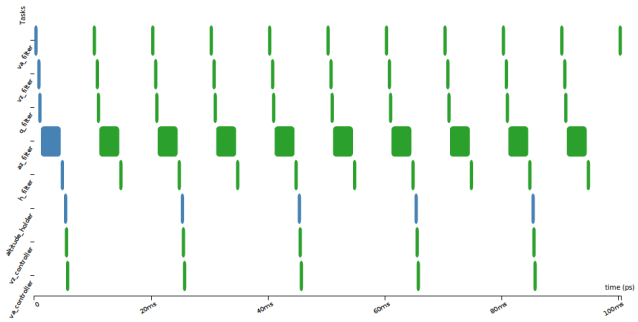

Figure 8: Timing simulation of the flight controller (*cpal_simu*) under FIFO scheduling in the *VerticalSpeed* mode.

worst-case response times (i.e. *bound*) under NPFP scheduling in the *Airspeed* scenario. The results are conclusive as every calculated worst-case response times *bound* is less than the corresponding relative deadline D. Thus, every laxity, which is the remaining time to deadline, is positive. Therefore, the task set is schedulable in the *Airspeed* scenario under the NPFP scheduling algorithm.

Complete experimental results with an evaluation of all the scheduling policies and running modes can be found in [10].

❹ *definition of ARINC653 partitions and* ❺ *validation.* This analysis aims at setting up an ARINC653 module M_5 to host the CPAL processes. The approach chosen here is to define a unique partition for all the processes. With that design choice, it is straightforward to define the parameters of the partition from those of the processes:

- the MAF_5 is equal to the *lcm* of the process periods,
- a different MIF_5 is not necessary as there is only one partition, hence $MIF_5 = MAF_5$,

Algorithm	Task	C (ns)	T (ns)	D (ns)	*bound* (ns)	*laxity* (ns)
np-fp	altitude_holder	164531	20000000	20000000	2049989	17950011
	va_controller	232967	20000000	20000000	2049989	17950011
	vz_controller	263957	20000000	20000000	1885458	18114542
	va_filter	241769	10000000	10000000	1652491	8347509
	h_filter	303957	10000000	10000000	1410722	8589278
	az_filter	371092	10000000	10000000	1146765	8853235
	q_filter	218801	10000000	10000000	842808	9157192
	vz_filter	252915	10000000	10000000	624007	9375993

Table 2: Worst-case response times computed by the *rts_periodic_np* analysis under NPFP scheduling in the *Airspeed* mode.

- the duration to execute the single partition P_1 of M_5 is $D_{5,1} = MAF_5$

In this particular case, the scheduling analysis is trivial as there is only one partition and the MAF is set to the hyperperiod of the processes.

If we choose a different partitioning of the processes (i.e. by assigning processes to different partitions), the analysis graph in Figure 7 has to be recomputed and a dedicated scheduling analysis must be available. In that case, the scheduling becomes hierarchical with a partition level schedule and a process level schedule. Here, a compositional analysis methodology as [45] could be applied to determine whether the processes are schedulable or not.

6.3 Iterative definition of the Bandwidth Allocation Gap from the AADL model

This second analysis flow aims at validating the temporal behavior of the AFDX network as specified in the AADL model. A problem at that stage is to define the parameters of the Virtual Links, in particular the Bandwidth Allocation Gap (BAG), in order to meet the latency constraints expressed on the dataflows.

As can be seen in the analysis graph in Figure 7, applying successively the two following analyses allows to set the BAGs:

❻ `bnh_bag_dimensioning` to identify the set of suitable BAG candidates for each VL in the network. This analysis is detailed in [46],

❼ `pegase_nc_analysis` that relies on Network-Calculus to compute upper bounds on communication delays (worst-case traversal times) in AFDX networks once BAGs have been set. This analysis is performed by the RTaW-Pegase tool [47].

The analysis process is here iterative as the analyses depend on each other: `bnh_bag_dimensioning` calculates the set of suitable BAGs that meets the latency constraints based, in part, on an estimation of communication delays in the Virtual Links, whereas the `pegase_nc_analysis` estimates the actual communication delays in the AFDX network based on the actual BAG definition.

Figure 9 shows the 3 iterations needed for the BAG definition process. The maximum BAG (i.e. `bnh_bag_dimensioning`) is refined at each iteration according to the traffic in the network (i.e. `pegase_nc_analysis`), which itself depends on the set of BAGs. According to the standard [34], every BAG must be set to BAG= $2^k ms$ with $k \in \{1, 2, \ldots, 7\}$. A fixed-point is reached at the third iteration, i.e. the set of parameters is identical to the previous iteration,

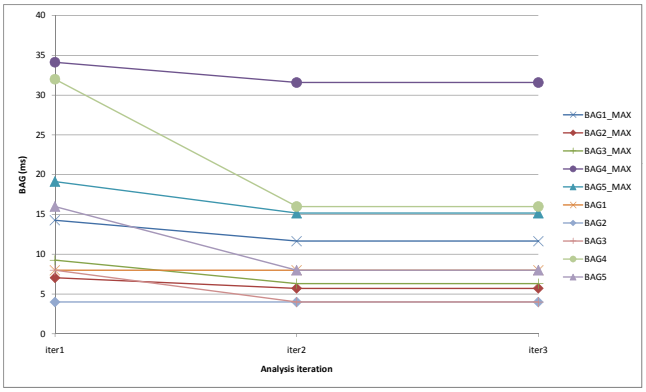

Figure 9: BAG definition process. At each analysis iteration, a BAG compliant with the AFDX standard is defined from the maximum value that meets the latency constraints expressed on the data flows. Three iterations are here required to find a stable solution.

meaning that the set of BAGs cannot be refined anymore. According to this BAG definition, the latency constraints expressed on the dataflows are met. Complete experimental results are available in [10, 46].

7 CONCLUSION

To couple models and analyses in Model-Driven Engineering for embedded systems, we propose an approach based on 4 layers: (1) models to define the system, (2) accessors to extract data from models, (3) analyses to compute output data and/or derive properties from input data, (4) contracts to specify the analysis interfaces and orchestrate the analysis process.

In this work, we present an application of the proposed approach in the real-time domain through a aerospace case study. Importantly, the approach allows to combine heterogeneous models and analyses. On the one hand, we used two modeling languages to specify the system at different levels of abstraction: overall and operational architecture of the system in AADL, functional architecture of the applications in CPAL. On the other hand, we combine different analyses (WCET, task scheduling, communication delays, various simulators, etc.) so as to gradually define important parameters of the system, and ultimately validate the schedulability of the system.

The applicability of our integration approach to the design of non-trivial real-time systems is illustrated through the case study treated in this work. From the system engineering perspective, we believe that such kind of integration approaches will facilitate the use by non-experts of the wealth of real-time scheduling results. As shown in this paper, valuable contributions from the real-time scheduling community such as real-time task models and schedulability tests can be integrated with usual descriptive modeling languages (AADL, SysML, etc.), qualified for a given context of use via contracts, and their execution can be fully automated through reasoning on contracts. From an engineering perspective, we believe that the systematic use of analyses, such as real-time scheduling analyses, with domain-specific models may substantially improve our way to develop critical embedded systems.

Future works may explore three lines of research: (1) extension of the integration approach, e.g. support of other types of analyses such as dependability analyses, investigate other languages for the specification and evaluation of contracts; (2) implementation of the approach: additional accessors, support the approach via a dedicated analysis and orchestration language, improve tooling; (3) integration with standard or emerging engineering processes and tools as SysML, CAPELLA, MATLAB/Simulink.

REFERENCES

[1] SAE International, *Architecture Analysis and Design Language (AADL) AS-5506A*, Std., 2009.

[2] P. Cuenot *et al.*, "The EAST-ADL Architecture Description Language for Automotive Embedded Software," in *Model-Based Engineering of Embedded Real-Time Systems*. Springer, 2011.

[3] B. Selic and S. Gerard, *Modeling and Analysis of Real-Time and Embedded Systems with UML and MARTE: Developing Cyber-Physical Systems*, ser. The MK/OMG Press. Morgan Kaufmann, 2013.

[4] T. Weilkiens, *Systems Engineering with SysML/UML: Modeling, Analysis, Design*, ser. The MK/OMG Press. Morgan Kaufmann, 2008.

[5] F. Singhoff, J. Legrand, L. Nana, and L. Marcé, "Cheddar: a flexible real time scheduling framework," in *ACM SIGAda Ada Letters*, 2004.

[6] M. González Harbour, J. G. García, J. P. Gutiérrez, and J. D. Moyano, "Mast: Modeling and analysis suite for real time applications," in *13th Euromicro Conference on Real-Time Systems (ECRTS)*. IEEE, 2001, pp. 125–134.

[7] K. G. Larsen, P. Pettersson, and W. Yi, "UPPAAL in a nutshell," *International Journal on Software Tools for Technology Transfer (STTT)*, 1997.

[8] Software Engineering Institute, "OSATE2 : An open-source tool platform for AADLv2," https://wiki.sei.cmu.edu/aadl/index.php/Osate_2, june 2016.

[9] B. Xu and M. Lu, "A survey on verification and analysis of non-functional properties of aadl model based on model transformation," in *5th International Conference on Education, Management, Information and Medicine (EMIM)*, 2015.

[10] G. Brau, "Integration of the analysis of non-functional properties in model-driven engineering for embedded systems," Ph.D. dissertation, University of Luxembourg, March 2017.

[11] M. Lauer, "Une méthode globale pour la vérification d'exigences temps réel - Application à l'Avionique Modulaire Intégrée," Ph.D. dissertation, Institut National Polytechnique de Toulouse, 2012.

[12] C. Pagetti, D. Saussié, R. Gratia, E. Noulard, and P. Siron, "The rosace case study: from simulink specification to multi/many-core execution," in *2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2014.

[13] N. Navet and L. Fejoz, *The CPAL Programming Language: An introduction*, v1.19 ed., July 2017, available at https://www.designcps.com/wp-content/uploads/cpal-intro.pdf.

[14] ——, "CPAL: High-level abstractions for safe embedded systems," in *Proc. of the ACM International Workshop on Domain-Specific Modeling (DSM)*, 2016, pp. 35–41.

[15] MathWorks, "Simulink - Simulation and Model-Based Design," https://www.mathworks.com/products/simulink.html (accessed January, 2017).

[16] G. Berry, "Scade: Synchronous design and validation of embedded control software," in *Next Generation Design and Verification Methodologies for Distributed Embedded Control Systems*. Springer, 2007, pp. 19–33.

[17] J. Eker, J. W. Janneck, E. A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Y. Xiong, "Taming heterogeneity-the ptolemy approach," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 127–144, 2003.

[18] A. Khoroshilov, I. Koverninskiy, A. Petrenko, and A. Ugnenko, "Integrating aadl-based tool chain into existing industrial processes," in *2011 16th IEEE International Conference on Engineering of Complex Computer Systems*, 2011.

[19] G. Holzmann, *Spin Model Checker, the: Primer and Reference Manual*, 1st ed. Addison-Wesley Professional, 2003.

[20] J. Hugues, B. Zalila, L. Pautet, and F. Kordon, "From the prototype to the final embedded system using the Ocarina AADL tool suite," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 7, no. 4, pp. 42:1–42:25, 2008.

[21] Y. Ouhammou, "Model-based framework for using advanced scheduling theory in real-time systems design," Ph.D. dissertation, Ecole Nationale Supérieure de Mécanique et d'Aérotechique de Poitiers, december 2013.

[22] A. Johnsen, K. Lundqvist, P. Pettersson, and O. Jaradat, "Automated verification of aadl-specifications using uppaal," in *14th International Symposium on High-Assurance Systems Engineering (HASE)*. IEEE, 2012, pp. 130–138.

[23] B. Berthomieu, J.-P. Bodeveix, C. Chaudet, S. Dal Zilio, M. Filali, and F. Vernadat, "Formal verification of AADL specifications in the Topcased environment," in *14th International Conference on Reliable Software Technologies Ada-Europe*, 2009.

[24] A.-E. Rugina, K. Kanoun, and M. Kaâniche, "The ADAPT tool: From AADL architectural models to stochastic petri nets through model transformation," in *Dependable Computing Conference, 2008. EDCC 2008. Seventh European*. IEEE, 2008, pp. 85–90.

[25] Y. Ouhammou, E. Grolleau, P. Richard, and M. Richard, "Reducing the Gap Between Design and Scheduling," in *20th International Conference on Real-Time and Network Systems (RTNS)*. ACM, 2012, pp. 21–30.

[26] V. Gaudel, "Des patrons de conception pour assurer l'analyse d'architectures: un exemple avec l'analyse d'ordonnancement," Ph.D. dissertation, Université de Bretagne Occidentale, november 2014.

[27] I. Ruchkin, D. De Niz, S. Chaki, and D. Garlan, "Contract-based integration of cyber-physical analyses," in *14th International Conference on Embedded Software (EMSOFT)*. ACM, 2014, p. 23.

[28] G. Brau, J. Hugues, and N. Navet, "A contract-based approach for goal-driven analysis," in *18th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC)*. IEEE, 2015.

[29] D. Jackson, *Software Abstractions: logic, language, and analysis*. MIT press, 2012.

[30] J. Migge, "Scheduling of recurrent tasks on one processor : a trajectory based model," Ph.D. dissertation, Université de Nice, 1999.

[31] C. Spitzer, U. Ferrell, and T. Ferrell, *Digital avionics handbook*. CRC Press, 2014.

[32] J. Forget, "A Synchronous Language for Critical Embedded Systems with Multiple Real-Time Constraints," Ph.D. dissertation, Université de Toulouse, 2009.

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

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

[35] P. Binns, M. Englehart, M. Jackson, and S. Vestal, "Domain-specific software architectures for guidance, navigation and control," *International Journal of Software Engineering and Knowledge Engineering*, 1996.

[36] L. Fejoz, "ROSACE Case Study: A CPAL implementation (version 1.0)," September 2016, available in the examples programs of the CPAL distribution at https://www.designcps.com.

[37] N. Navet, L. Fejoz, L. Havet, and S. Altmeyer, "Lean Model-Driven Development through Model-Interpretation: the CPAL design flow," in *Embedded Real-Time Software and Systems (ERTS)*, 2016.

[38] A. Benveniste and G. Berry, "The synchronous approach to reactive and real-time systems," *Proceedings of the IEEE*, vol. 79, no. 9, pp. 1270–1282, 1991.

[39] T. A. Henzinger, B. Horowitz, and C. M. Kirsch, "Giotto: A time-triggered language for embedded programming," in *Embedded software*, 2001.

[40] J. Forget, F. Boniol, D. Lesens, and C. Pagetti, "A real-time architecture design language for multi-rate embedded control systems," in *25th Symposium on Applied Computing (SAC)*. ACM, 2010, pp. 527–534.

[41] I. Cibrario Bertolotti, T. Hu, and N. Navet, "Model-based design languages:a case study," in *13th IEEE International Workshop on Factory Communication Systems (WFCS2017)*, Trondheim, Norway, June 2017.

[42] F. J. Cazorla *et al.*, "Proartis: Probabilistically analyzable real-time systems," *ACM Trans. Embed. Comput. Syst.*, vol. 12, no. 2s, pp. 94:1–94:26, May 2013.

[43] S. Baruah and A. Burns, "Sustainable scheduling analysis," in *2006 27th IEEE International Real-Time Systems Symposium (RTSS'06)*, Dec 2006, pp. 159–168.

[44] S. Altmeyer, S. M. Sundharam, and N. Navet, "The Case for FIFO Real-Time Scheduling," University of Luxembourg, Tech. Rep., 2015.

[45] J. P. G. C. Craveiro, "Real-Time Scheduling in Multicore Time-and Space-Partitioned Architectures," Ph.D. dissertation, Universidade de Lisboa, 2013.

[46] G. Brau, J. Hugues, and N. Navet, "Refinement of AADL models using early-stage analysis methods – An avionics example," Laboratory for Advanced Software Systems, Tech. Rep. TR-LASSY-13-06, 2013.

[47] M. Boyer, J. Migge, and M. Fumey, "PEGASE - A Robust and Efficient Tool for Worst-Case Network Traversal Time Evaluation on AFDX," in *SAE AeroTech Congress & Exhibition*, Toulouse, France, October 18-21 2011.