Integration of heterogeneous formal techniques for the design of avionics systems

Yamine AIT AMEUR¹, Frédéric BONIOL², Rémi DELMAS², Emmanuel GROLLEAU¹, Nathalie TORRECILLAS¹ and Virginie WIELS²

¹ LISI-ENSMA, Université de Poitiers, 86960 Futuroscope Cedex, France. email: {yamine, grolleau, torrecna}@ensma.fr

²ONERA-CERT, 2 Av. E.Belin, 31055 Toulouse Cedex, France. email:{boniol,rdelmas,wieels}@cert.fr

1 Introduction

In the industry in general and in the aeronautics domain in particular, the development of embedded software for new aircrafts rarely starts from scratch but rather proceeds from the assembly of basic or existing subsystems and from evolutions of systems already developed for previous aircrafts. As a result, the largest part of a new system development is the validation part, all the more when the system is critical and must comply with certification regulations. Moreover, the complexity of these systems require the use of domain oriented views, where specific classes of properties are studied using dedicated techniques. In this setting, the reuse of components together with their validation would provide a great benefit. Therefore, the component descriptions used in the development process should include their properties as well as information on the technique used for their validation. The resulting heterogeneous development process shall be studied and mastered.

1.1 A methodology. Why ?

To handle the above-mentioned system developments, it is necessary to provide a methodology addressing activities such as system design, specification, modelling, reuse, evolution, verification, validation, etc. This methodology must support not only these activities but also the plans and documents resulting from these activities.

In this paper, we propose a methodology that makes a first step in that direction. We opted for a clear separation of concerns, by defining two levels of methodological support. The first is a structural and modular description level. The second level is the semantic level, where designers model the dynamic behavior of the components to establish specific
properties using dedicated techniques. We have not yet worked on the vertical dimension of the methodology (refinement process) which is part of future work.

Two categories of approaches to building this kind of representation can be distinguished:

- generic methodologies, which claim adaptability to various engineering areas. The generality of such methodologies yields a reduced support for formal techniques in specification, design, validation and verification stages. Moreover, these approaches are not aimed at capturing application field specific aspects of systems, resulting in the specific knowledge being hard-encoded in the descriptions. Usually, domain oriented profiles are built from these methodologies in order to support domain specific developments;

- specific, application domain targeted approaches. They allow the precise description of systems in a given engineering context, but mostly suffer from their closure to a particular application domain. Moreover, these methodologies require the designers to have a deep and precise working knowledge of the field’s formal techniques and tools, in order to put them into practice. When the system to be developed is a complex one, the use of such methodologies leads to a multiple heterogeneous usages

Contrarily to generic and domain specific methodologies, our proposal separates the system static description in the central model from the domain specific dynamic descriptions in the views.

![Figure 1. Different parts involved in the methodology.](image)

As depicted in figure 1, our work consists in providing a methodology for bridging the gap between:

1. the formal notation of the central architectural model (CAM) and the formal verification techniques used in the views, in order to link modelling, verification and validation to a given structural system description. This problem is addressed through the notions of *components* and *composition* presented in this paper;

2. two or more different formal verification techniques. Indeed, properties established using different modelling and verification techniques on different models of the system must be combined to validate system-level requirements. The notion of *view* is presented in this paper, the particular issue of coherence between different views is more specifically addressed in [3].
Our methodology for the design and development of complex systems supports features usually required by the designers. A formal notation is used to define a central model of the system, holding component attributes and defining component compositions, describing the connections between components. The incremental design of the system is represented as a sequence of composition operations in the central model. The views are formally connected to the central model. The relevant properties the system must fulfill can be verified at different steps of the composition process, depending on whether a property is local to a component or to a group of components within view, or is rather a global property of a composite whose validation spans more than a single view. A given property can be verified as soon as enough information is present in the current refinement, before going on to the next refinement or composition/decomposition step. This enables designers to decompose property verification and system validation activities.

1.2 The application domain

Nowadays, in embedded systems and particularly IMA systems (Integrated Modular Avionics), design spreads across a wide variety of engineering areas. When involved in critical applications, these systems must meet requirements whose validation can be achieved using compositional approaches and methodological rules to handle complexity. Such systems consist of critical applications distributed over a network of communicating computers, based on an open architecture.

Software engineering for information processing and command and control operations plays a major role in the design of such IMA systems. Each development activity is usually supported by one or more technique, leading to a heterogeneous development of a single system. We say that each activity leads to a particular view: functional view, performance view, fault tolerance view, deployment view etc. Designers use results issued from views to assert that the whole designed system is valid.

Starting from this situation, a methodology handling the development of IMA systems was proposed in the context of an ONERA study, financed by the DPAC\textsuperscript{1}. The approach uses the component as the central notion of the development process. A composition operation allows to build complex components from atomic or simpler components. This approach separates the component and composition descriptions from the views used to model the described components.

1.3 Organisation of the paper

This paper presents the methodology and its application on a real world IMA case study. This methodology gathers and synthesizes the necessary knowledge for describing IMA components (variable abstraction, incremental design, component reuse, etc.). Its backbone is a formal component calculus with composition, which includes the generic notion of domain oriented view. The next section presents the IMA case study used to testdrive our approach. Section 3 presents the central model and its notation, and presents the notion of view. The section 4 briefly describes the application of the methodology on different views of the case study.

\textsuperscript{1}French government’s National Civilian Aviation Program Committee.
The goal of the study is to validate the reconfiguration mechanism of a command and slaving subsystem, in charge of controlling three aerodynamic surfaces of an aircraft, as depicted on figure 2. The subsystem consists of three computers, connected to a switch component thanks to digital buses, and connected to the aerodynamic surfaces thanks to analog buses.

Figure 2. Informal view of the system.

A slaving command for the actuator of an aerodynamic surface is elaborated from numerous parameters in multiple steps. First, a command law determines an angular position set point \( \alpha \) for the surface, taking into account parameters from the aircraft environment such as: the pilot’s stick position, auto pilot orders, current speed and acceleration of the aircraft, current angular position of the surface, etc. This \( \alpha \) is then passed on to a slaving law, in charge of producing the correct slaving command to be sent to the actuator of the mobile surface to achieve the desired angular position \( \alpha \). This computation also involves aircraft environment parameters. This system is obviously critical, so it is triple-redundant, and it is monitored by three distributed reconfiguration laws. Their duty is to detect eventual failures of the computers executing the command and slaving laws, and to trigger a reconfiguration mechanism whenever needed, to make sure a mobile surface never stays idle.

Each one of the three computers runs the following groups of tasks, or partitions:

1. **The Command Laws Partition** (\( \text{com}_i \)), which produces angular set points \( \langle \alpha_1, \alpha_2, \alpha_3 \rangle \), one for each of the three mobile surfaces;

2. **The Slaving Laws Partition** (\( \text{slaving}_i \)), which takes \( \langle \alpha_1, \alpha_2, \alpha_3 \rangle \) as input and produces the triple \( \langle \text{slav}_1, \text{slav}_2, \text{slav}_3 \rangle \), holding a slaving command for each one of the three actuators;

   (a) in nominal mode, computer \( \text{Comp}_i \) is in charge of surface \( \text{Surf}_i \) only, but can also, in case of failure of both of its neighbors, inherit the charge of the three surfaces;

   (b) the \( \text{slav}_j \) set points are sent to the actuators via an analog bus;

   (c) each \( \text{slav}_j \) emitted by computer \( \text{Comp}_i \) is accompanied by a broadcast of a notification \( \text{delivered}_{ij} \) through the network, meaning “computer \( \text{Comp}_i \) informs its environment it is currently slaving surface \( \text{Surf}_j \)”. The delivery of
notifications from partition $slaving_i$ to $Comp_i$ is achieved directly into $Comp_i$ without using the network, and is thus achieved without noticeable latency.

3. The Reconfiguration Partition ($reconf_j$), which receives and monitors the $delivered_{i,j}$ notifications, to detect eventual failures:

   (a) each reconfiguration partition $reconf_i$, running on $Comp_i$, monitors notification events coming from the three $slaving_j$ slaving partitions;

   (b) the absence of a notification event $delivered_{i,j}$, confirmed for a certain amount of time, reveals the failure of $slaving_i$ to slave $Surf_j$. This failure event is detected by both remaining computers, and a distributed priority scheme determines which computer backs up the faulty one.

3 The methodology

In this section, we present the central model (components and composition) and the notion of view.

3.1 Components and composition

The vision of components and composition defined thereafter is greatly inspired from the observation of industrial practices in the aeronautics domain (ASAAC standards) [2, 4]. Engineers use the notion of view to perform their own domain oriented design.

Components. A component, as depicted in figure 3(a), consists of three layers ($A, E, F$). Each layer is composed of a set of ports and a set of functions.

- The $A$ layer holds attributes describing the hardware execution and communication resources of the component: performance attributes, fault probabilities, etc.

- The $E$ layer holds attributes describing the executive functions of the component: scheduling policy, context switching jitter, communication ports and protocols, etc.

- The $F$ layer holds attributes describing the avionics functions of the component: name, number, type and range of input-output parameters, period, best case execution time $bcet$, worst case execution time $w cet$, etc.

Static mappings of ports and functions between layers can be defined using interaction hypotheses $H_{AE}$ and $H_{EF}$, representing for instance:

- at the $A$ – $E$ interface: allocation of hardware communication ports to a $E$-layer port;

- at the $E$ – $F$ interface: allocation of $E$-layer ports to function parameters;

- at the $E$ – $F$ interface: mapping of function names to entries of the scheduler process table.
Composition. The composition operation produces a component $C$ from two components $C_1$ and $C_2$, in two steps (cf. fig3(b)):

1. definition of the interaction hypotheses $HI$ between $C_1$ and $C_2$. $HI$ usually describes port mappings or resource sharing between $C_1$ and $C_2$;

2. fusion of the two components into the composite $C = \circ(C_1, C_2)$.

The description of components and composition and the links to the views can be used as a formal documentation of the design shared by all the actors involved in the development process.

3.2 View description

The components used in the central model are characterized by a set of attributes. Attribute values are not sufficient to perform analysis of any kind on the components, but they are used as data to build models of the components in various domain oriented views.

The definition of a view is done along the following steps:

1. choose the formal technique suited to the view;

2. model the component in the chosen technique, using attribute values taken from the corresponding component of the central model;

3. among the set of required properties of the component, encode the ones that can be verified using the chosen technique. The verification of a set of properties in a given view might use verification hypotheses, that must be explicitly stated. These hypotheses shall be validated within the same view, or considered as properties defined and verified within another view, using other techniques. This approach yields the possibility to perform an assume/guarantee reasoning.

This process is iterated until the whole set of requirements is validated.

In the next section, we will give examples of specific views that can be built in the avionics domain, this set of examples is not exhaustive, work is ongoing to define a safety view and other kinds of views are possible in our framework.
3.3 Use of formal techniques

There is a variety of formal techniques to choose from to validate the requirements expressed by the properties: model checking techniques, proof oriented techniques, abstract interpretation, scheduling analysis, probabilities calculus, static and dynamic analyses etc.

The techniques are chosen according to their ability to support the component paradigm (using an encapsulation mechanism of some kind); their ability to perform verification under a set of assertions; their expressiveness and verification power with respect to the class of properties the view is focused on.

4 Application to the case study

This section gives a brief review of:

1. the different component descriptions and the sequence of composition performed in order to build the system of the case study (represented by the final component of the composition sequence);

2. examples of views that can be defined to validate the user requirements on the components.

The whole details of the development can be found in [2].

4.1 Central architectural model

Using the notation described in section 3, a composition scenario involving all the basic components of the case study (Switch, Comp, Bus etc.) was described. Each composition step produces an intermediate component representing the current state of the system design. Figure 4 shows the different compositions and the intermediate components used to describe the whole case study.

![Figure 4. A composition scenario involving the components of the case study.](image)

Models of this component can be encoded at any step of the composition scenario (basic, intermediate or final step) in any view, as soon as enough information is found in the attributes of the central model.
4.2 Application architecture view

The Event-B method [1] was used to model component calculus of the structural level given in section 4.1 with Event-B semantics. Verifying this model under the hypothesis that the communication services were robust allowed to prove that the distributed system was logically non blocking. Requirements such as “at least one computer Comp_j will run the Com_i, Slav_i and Reconf_i processes” were validated using this formal development.

4.3 Functional view

The Lustre formal technique [6] was used to define the functional view of the case study. It gives the semantics of the actual computations performed by the distributed command, slaving and reconfiguration laws.

Requirements such as “the distributed reconfiguration laws must be such that, in case of failure of up to two of the three computers, and assuming that the communication services are robust:

1. an aerodynamic surface is always slaved by at most one computer;
2. no aerodynamic surface remains idle for more than n milliseconds.”

were validated using model checking in this view.

4.4 Real time performance view

In this view, the technique UPPAAL [7] was used to verify global scheduling properties at the system level. A network of timed automata models the dynamics of the whole distributed system execution, using the locally validated scheduling policy (imported from the deployment view for each computer). Requirements such as “the data transfers between computers must be deterministic and robust, and data must always be delivered from a computer to another with a latency under a fixed upper bound” can be verified, with non pessimistic upper bounds.

4.5 Deployment view

The command-reconfiguration subsystem shares its execution and its communication ressources with other distributed applications, integrated on the same computers and network. These other applications constitute the environment of our system. The local real time scheduling constraints (on a given computer) on our system can be verified only by taking into account this environment. The PENSMArts [5] formal technique, performing scheduling analysis using Petri nets, was used to determine and verify the local scheduling policy for a total of 248 processes running on the shared computers.

4.6 Synthesis

The system semantics is defined by a collection of models encoded in different views, all built from a common set of data taken in the central model. In a given view, the
verification of properties is usually performed under a set of hypotheses. These hypotheses are in turn justified by the properties established on other models, in different views of the same system. This whole process allowed to verify that the system, as defined by this collection of models, satisfies the global requirements. Moreover, it supports assume/guarantee reasoning.

Figure 5 summarizes the different development steps involved in our case study. All the views are built using common information issued from the central model which is available during the whole development (dashed line). Each view corresponds to the use of a given formal technique (continuous line).

5 Conclusion

This paper presented a methodological notation allowing to:

- incrementally build complex components starting from (or reusing) basic components;
- take into account, through hypotheses, the properties of the components environment.
- give the conditions which ensure the consistency, with respect to the considered environmental hypotheses, of the produced components;
- extract and build different views of a component for validating specific domain oriented requirements.

The case study showed that the methodological notation we propose can be used as a neutral representation of compositional developments that can be encoded thereafter in several formal techniques with tools support. Indeed, theorem proving oriented techniques like EVENT-B and model checking oriented techniques like LUSTRE, PENSMArts or UP-PAAAL have been successfully used to encode and verify the whole case study presented at the beginning of this paper.
Currently, we are developing a formal representation of the central model. Category theory is set up as the underlying theoretical framework and the EXPRESS [8] data modelling language is used to encode it.

However, other investigations need to be performed in the future. Among them, extending the central notation in order to handle other composition operators such as component refinement, component weakening etc.

References


