

A Framework for Heterogeneous Formal Modeling and Compositional Verification of Avionics Systems

Rémi Delmas, Virginie Wiels
ONERA-CERT, 2 Av. E.Belin, 31055 Toulouse Cedex, France.
email : {rdelmas,wiels}@cert.fr

Yamine Aït-Ameur
LISI-ENSMA, Université de Poitiers, 86960 Futuroscope Cedex, France.
email : yamine@ensma.fr

Abstract

This paper presents a component oriented framework dedicated to the specification of embedded systems in the aeronautics domain. A component is an entity with three internal layers (hardware, operating functions and applicative functions) together with a collection of models in different domain-oriented views. A composition operation allows the expression of composition scenarios, yielding a component calculus for representing composite systems. An institutional framework supports this component calculus, allowing the expression of coherence criteria between heterogeneous views. This framework can be seen as a formal documentation of a system development and analysis, supporting heterogeneous modeling and compositional verification. The approach is illustrated on a non trivial case study.

1 Introduction

In this paper, we consider the problem of global requirements validation, in the field of avionics systems. This class of reactive systems is subject to multiple and drastic safety requirements, due to the criticality of their mission.

The current and ever growing level of complexity of these systems forces designers to resort to multiple domain oriented views, such as Fault Tolerance (FT), Real Time Performance (RTP) and Functional (F) views. In these views, formal techniques, by imposing a tight and precise semantic frame, enable designers to define models of the systems, express required properties using some appropriate formal language, and eventually verify the models, using automated or interactive verification tools.

Usually, each view uses a technique suited to the aspect it focuses on. Most of the time, several studies are conducted

in parallel in different views, by different design teams. Information is exchanged between views, each design group providing its neighbor with the results of its own analysis, in a globally iterative development cycle. Considerable synthesis efforts have to be made in order to determine whether the final design, as seen from the multiple heterogeneous views, meets the global, system-level requirements. This effort is all the more important when there is no synthetic view of the system, showing the relationships between the multiple designs, models and local validation tasks carried out in the different views.

In this setting, drawn from the current practices, the following questions were raised by industrials of the aeronautics domain, and were the subject of an ONERA study, financed by the DPAC¹ [1] :

- how to build a synthetic representation of a system, its different views and their interactions ?
- how to univocally express a global requirement on a system ?
- how to formally validate such a requirement ?

Our work tries to bring answers to these questions by proposing a component oriented framework, which allows the homogenous and formal representation of :

- the incremental construction of the system from components using composition,
- the different views and models used in the analysis of the system.

The framework also enables one to formally express coherence criteria between views, and hence facilitates the expression of meaningful global properties.

Our applicative field and case study involves the three FT, RTP and F views, nevertheless the approach is generic and supports the definition of an arbitrary number of views.

¹French government's National Civilian Aviation Program Committee.

Paper Outline. In section 2, the case study illustrating our approach is described. It deals with an avionics subsystem, on which global requirements are expressed. The formalization of components and composition is given in section 3, and we show how it permits the construction of a static view of the system, central to our framework. In section 4, the concept of *institution* [8, 3] is introduced, and its use in the representation of the views and their formal techniques is explained as well. The resulting approach is generic and is instantiated on the RTP and F views of the case study in sections 5 and 6. The modeling techniques used in these views are respectively UPPAAL [7] and LUSTRE [5]. In section 7, the generic method used to connect a view to the central model is described. An example on the case study is given to illustrate how the F and RTP views are connected to the central model of the case study. In section 8, the *synchronization of institutions on models* construct [9] is explained. This construct allows expressing semantic coherence criteria between views, via a *synchronization relation*. Eventually, in section 9, an example of synchronization relation between the RTP and F views of the case study is detailed, followed by an example of global property verification. Last, in section 10, we conclude this paper and give the perspectives of our work.

2 Case Study

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 1. The subsystem consists of three computers, connected to a switch component via digital buses, and connected to the aerodynamic surfaces via analog buses.

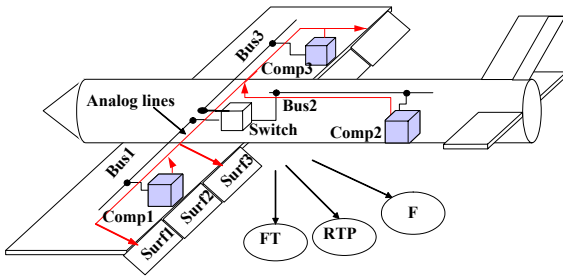


FIG. 1. Informal view of the system.

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 α 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

α is then passed on to a *slaving law*, in charge of producing the correct *slav* command to be sent to the actuator of the mobile surface, to achieve the desired angular position α . 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 $Comp_i$ runs the following groups of tasks, or *partitions* :

1. **Command Laws Partition** (com_i), which produces angular set points $\langle \alpha_1, \alpha_2, \alpha_3 \rangle$, one for each of the three mobile surfaces.
2. **Slaving Laws Partition** ($slaving_i$), which takes $\langle \alpha_1, \alpha_2, \alpha_3 \rangle$ as input and produces the triple $\langle slav_1, slav_2, slav_3 \rangle$, holding a slaving command for each one of the three actuators.
 - (a) in nominal mode, computer $Comp_i$ is in charge of surface $Surf_i$ only, but can also, in case of failure of both of its neighbors, inherit the charge of the three surfaces.
 - (b) the $slav_j$ set points are sent to the actuators via an analog bus,
 - (c) each $slav_j$ emitted by computer $Comp_i$ is accompanied by a broadcast of a notification $delivered_{i,j}$ through the network, meaning “computer $Comp_i$ informs its environment it is currently slaving surface $Surf_j$ ”. 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. **Reconfiguration Partition** ($reconf_i$), 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.

A model of the system is built in each of the following views :

- a safety and fault tolerance view, not detailed in this paper ;

- a real time performance view, in which the timed automata formalism is used, using the UPPAAL [7] environment.
- a functional view F , using the synchronous data flow language LUSTRE, and the model checker LESAR [5].

The requirements on the system are :

- **req₁** : the communication medium must satisfy a probability of 10^{-9} chances of failures per hour. The property is verified on the model of the FT view, using hypotheses on the atomic failure probabilities of the hardware components, and using the topology and redundancy of the hardware ;
- **req₂** : the data transfers between computers must be deterministic and robust, and a data must always be delivered with a latency under a fixed upper bound. This property is verified on the model of the RTP view, using hypotheses concerning the robustness of the hardware communication medium, imported from the FT view ;
- the distributed reconfiguration laws must be such that, in case of failure of up to two of the three computers :
 - **req₃₁** : in nominal mode, $Comp_i$ slaves $Surf_i$;
 - **req₃₂** : each $Surf_i$ is always slaved by at most one $Comp_j$;
 - **req₃₃** : no $Surf_i$ remains idle for more than n milliseconds.

These properties are verified on the model of the F view, using hypotheses imported from the RTP view.

The requirements on the whole distributed reconfiguration mechanism are indeed global, as both its functional correctness and its performance are studied.

3 The Central Model

The vision of components and composition defined thereafter is directly inspired from the observation of common industrial practices[1, 2]. The informal intuition and categorical formalization of these concepts are given in this section.

3.1 Intuition

Components (fig.2(a)). A component, consists of three layers (A, E, F). Each layer defines and holds the values of a number of *attributes* :

- the A layer holds attributes describing the hardware execution and communication resources of the component : performance attributes, fault probabilities, etc.
- the E layers 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 of the function, number, type and range of input-output parameters, period or triggering condition, best case execution time $bcet$, worst case execution time $wcet$, etc.

Static mappings of attributes 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 (fig.2(b)). On these components, a composition operation is defined. It produces a component C from two components C_1 and C_2 , in two steps :

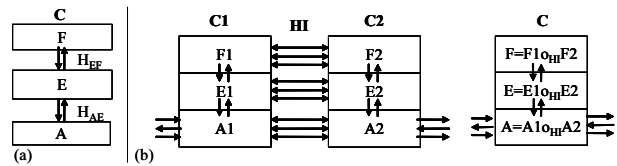


FIG. 2. Components and composition, intuitive view.

1. definition of the interaction hypotheses HI between C_1 and C_2 . HI usually consists of port mappings (both hardware and software), unions of sets of functions, function synchronizations, etc. performed during composition.
2. fusion of the two components into $C = \circ(HI, C_1, C_2)$.

3.2 Formalization

Components (fig.3(a)). The layers of a component are formally represented using a triple of *signatures*, each one defining a set of *sorts* and sorted attributes.

A *signature morphism* $h : sign_1 \rightarrow sign_2$ represents the embedding of a signature $sign_1$ in a bigger signature $sign_2$. The morphism maps a sort $s_2 = h(s_1) \in sign_2$ to each sort $s_1 \in sign_1$, and maps an attribute $att_2 = h(att_1) \in sign_2$ of sort $h(s_1)$ to each attribute $att_1 \in sign_1$ of sort s_1 .

These signature morphisms allow representing the inter-layer mappings using a *pushout* construct. The interaction hypotheses H_{AE} and H_{EF} are defined using two signatures and two pairs of morphisms, defining the sharing of sorts and attributes by the signatures A and E , and by the signatures E and F as shown on figure 3(a)..

The unique *component signature* of the component C is given by the colimit of the diagram formed by the signatures A, E, F, H_{AE} and H_{EF} and the two pairs of morphisms.

Composition (fig.3(b)). The notion of *signature morphism* is extended to the notion *component signature morphisms*, defined as 5-tuples of signature morphisms mapping A layers to A layers, E to E, F to F, H_{AE} to H_{AE}, H_{EF} to H_{EF} and satisfying additional coherence properties.

Component composition is again represented using a *pushout* of *component signatures* construct. The interaction hypotheses HI are represented using a component signature C_0 , which inner signatures hold the attributes and sorts that are going to be shared by the component signatures of C_1 and C_2 . Two component signature morphisms are defined between C_0 and C_1 , and between C_0 and C_2 . The signature of C is obtained by pushout on those morphisms, as shown on figure 3(b).

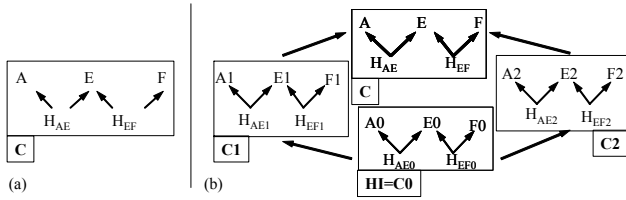


FIG. 3. Components and composition, formalization.

The *component signatures* and their *morphisms* form the co-complete category \mathbf{Sign}_{Comp} .

3.3 Case Study : The Central Model

The component signatures and the composition operation allow to represent a system as a diagram of *components signatures pushout* constructs. Figure 4 illustrates a possible composition scenario for the system of the case study. The components of the system ($Comp_i, Bus_i$ and $Switch$) are defined by component signatures and then composed incrementally. Different scenarios can yield the same system; the constraints associated with component signature morphisms guarantee the coherence of the results. This composition diagram is called the *central model* of the system, or its *static view*.

4 Formalizing Views

4.1 Intuition

The *central model* holds the *structural* information of the system. The *semantics* of the components are defined in the

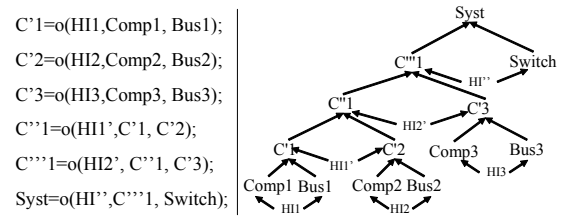


FIG. 4. A composition scenario for the system.

different *views* which, although using heterogeneous formal techniques, are built on the common basis of the *central model* (cf. sect.7).

In this section, we define the approach used to define a *view* in the framework together with the formal technique associated with it. Sections 7 and 8 show how to connect a view to the *central model*, and how to synchronize several views to achieve semantic coherence. From now on we call these views *dynamic*, as their models define and hold the multiple behaviors, interactions and properties of the system.

4.2 Formalization

The variety of formal techniques used in the different views can be dealt with in a uniform way through the concept of an *institution* [8, 3], which formalizes the notion of *logical system*. For each formal technique T we wish to represent in the framework, an *institution* must be identified :

$$I_T = \langle \mathbf{Sign}_T, \mathbf{Sen}_T, \mathbf{Mod}_T, \models_{T,\Sigma} \rangle$$

with :

- \mathbf{Sign}_T : a category whose objects are *signatures* and whose arrows are *signature morphisms*, representing the vocabulary of the property language of the formal technique T ;
- $\mathbf{Sen}_T : \mathbf{Sign}_T \rightarrow \mathbf{Set}$: a functor which associates to each $\Sigma \in \mathbf{Sign}_T$ the set $\mathbf{Sen}_T(\Sigma)$ of well formed *sentences* over the vocabulary of Σ , thus representing the property language of the technique T ;
- $\mathbf{Mod}_T : \mathbf{Sign}_T \rightarrow \mathbf{Cat}^{\mathbf{Op}}$: a covariant functor which associates to each $\Sigma \in \mathbf{Sign}_T$ a category whose objects are Σ -*structures* and whose arrows are Σ -*structure morphisms*, representing all the possible semantic structures on which the sentences of $\mathbf{Sen}_T(\Sigma)$ are interpreted ;
- $\models_{T,\Sigma} \subseteq |\mathbf{Mod}_T(\Sigma)| \times \mathbf{Sen}_T(\Sigma)$ for each $\Sigma \in \mathbf{Sign}_T$, a relation called Σ -*satisfaction*, such that $\forall \phi : \Sigma \rightarrow \Sigma' \in \mathbf{Sign}_T$, the *satisfaction condition* holds for each $m' \in \mathbf{Mod}_T(\Sigma')$ and for all $\varphi \in \mathbf{Sen}_T(\Sigma)$: $m' \models_{T,\Sigma'} \mathbf{Sen}_T(\phi)(\varphi) \Leftrightarrow \mathbf{Mod}_T(\phi)(m') \models_{T,\Sigma} \varphi$

A pair $\langle \varphi, m \rangle$ belongs to $\models_{T, \Sigma}$ if and only if the model m satisfies the formula φ . The satisfaction condition expresses the fact that *the truth is invariant under vocabulary change*. If a formula $\text{Sen}_T(\phi)(\varphi)$ is satisfied by a model m' , then φ is satisfied by $m = \text{Mod}_T(\phi)(m')$, reduction of m' to the symbols of Σ .

4.3 Our Approach

The main steps involved when introducing a view together with a formal technique T in the framework are summarized :

1. **Step 1** : for each new technique T , identify an institution able to represent it, and establish a formal connection between the technique semantics and the institution semantics (defined in sect.4, illustrated in sect.5 and sect.6);
2. **Step 2** : model the components in the view using the technique T (illustrated in sect.5.2 and sect.6.2)
3. **Step 3** : define the signature-level constraints to be satisfied when embedding a component from the *central model* into the view (defined and illustrated in sect.7);
4. **Step 4** : define the eventual synchronization relations between this institution and those already in the framework (defined in sect.8 and illustrated in sect.9).

5 The Real Time Performance View

In this section the approach defined in section 4.3 is instantiated on the RTP view of the case study.

5.1 Step 1 : Institution

Two logics able to represent the technique UPPAAL in the framework were identified : L_ν [6], a fragment of μ -calculus, and XSL , a fragment of L_ν . L_ν is a branching, dense time logic of actions of the future tense, in which quantitative safety properties can be expressed with reference to the initial state. XSL is the linear fragment of L_ν . In the following subsection, we give a short description of the signatures, models and satisfaction relation of both logics, with an emphasis on their models. But first, we must recall the definition of a timed automaton.

Timed Automata. A timed automaton² is defined by :

$$\mathcal{A} = \langle A, N, n_0, C, E, v \rangle$$

with :

²due to space restrictions, the definition of a timed automaton given here is kept basic, and does not include *urgent* or *committed* states, *state invariants*, *broadcast* or *urgent channels*...

- A : a set of *actions*. There is two types of actions, *actions* $\{a, b, c, \dots\}$, and *delay actions* or ε -*actions* $\varepsilon(d)$, where $d \in \mathbb{R}$ represents the value of the delay; ε -actions consume time, whereas *actions* do not ;
- N : a set of nodes ;
- $n_0 \in N$: the initial node ;
- C : a set of clocks ;
- $E \subseteq N \times N \times A \times 2^C \times B(C)$: a set of edges ;
 $e \in E = \langle n, n', a, r, b \rangle$ is an edge between nodes n and n' , labeled with action a , r is the set of clocks to be reset when the transition is fired, and b is a condition on clocks of C guarding the transition.
- a valuation $v : C \rightarrow \mathbb{R}$ associates a real value with each clock of C . The valuation $[v + d]$, $d \in \mathbb{R}$ is such that $\forall c \in C, [v + d](x) = v(x) + d$, the valuation $v' = [r \rightarrow 0]v$ is such that $v'(c) = 0$ if $c \in r$, or else $v'(c) = v(c)$. The set of valuations is noted \mathbb{R}^C ;

A state s is defined as a pair $\langle n, v \rangle$, with $n \in N$ and v a valuation. The set of states is noted S .

Signatures $\text{Sign}_{L_\nu/XSL}$ and Sentences $\text{Sen}_{L_\nu/XSL}$. Signatures of L_ν and XSL consist of the set of actions of timed automata given above together with a set of clocks variables. More details can be found in [6].

Models $\text{Mod}_{L_\nu/XSL}$. Models of L_ν/XSL are the traces of timed labeled transition systems $\mathcal{M}_A = \langle \Sigma_A, \sigma_0, \rightarrow_A \rangle$, with :

- \mathcal{A} : a timed automaton ;
- $\Sigma_A \subseteq N \times \mathbb{R}^C$: set of states ;
- $\sigma_0 \in \Sigma_A$: initial state ;
- \rightarrow_A : transition relation such that $\langle n, v \rangle \xrightarrow{a}_A \langle n', v' \rangle$ iff $\exists r, b \langle n, n', a, r, b \rangle \in E \wedge b(v) \wedge v' = [r \rightarrow 0]v$.

A trace τ_{L_ν} (resp. τ_{XSL}) is a sequence $\{\sigma_0, \dots, \sigma_n, \dots\} \in \Sigma_A^\infty$ such that $\forall i, \langle \sigma_i, \sigma_{i+1} \rangle \in \rightarrow_A$. The set of traces generated by \mathcal{M}_A is noted $L(\mathcal{M}_A)$. More details can be found in [6].

Satisfaction Relation $\models_{L_\nu/XSL}$. The satisfaction relation is inductively defined on the elements of $L(\mathcal{M}_A)$, more details can be found in [6].

5.2 Step 2 : Modeling With Uppaal

A network of parameterized timed automata is used to model a combination of the A and E layers of the system components.

An automaton *part_sched_i* models the partition sequencer of *Comp_i*, which periodically emits *start_slaving_i!* and *start_reconf_i!* messages, to

trigger the execution of the slaving and reconfiguration partitions³.

An automaton $part_name_i$ models the execution of a partition. It leaves its *idle* state on reception of a $start_part_name_i?$ message to enter its *running* state (after a variable *jitter*) and runs for an amount of time of at least the partition *bcet*, and at most the partition *wcet*. On completion, the automaton broadcasts a $comp_part_name_i!$ message.

A generic automaton vl_in_i , triggered by the $comp_part_name?$ message, is used to model the time consumed by the preparation and delivery of the data produced by a partition (here the $delivered_{i,j}$ notifications of the slaving partitions) to the switch component. A message $switch_delivery_i!$ is emitted on completion.

The switch component is modeled as the parallel composition of the following automata : three input automata $switch_in_i$ (one for each $Comp_i$), which, when triggered by a $switch_delivery_i?$ message, copy the identifier i in one or more of three queues (each queue is modeled by an automaton). Three vl_out_i automata (one for each $comp_i$), scheduled by an automaton com_sched , check for the presence of data waiting to be sent in the queues, and broadcast a $delivered_c[i][j]!$ message when data from $Comp_i$ has actually been delivered to $Comp_j$. One or several other automata can interact with the switch to represent the global network load.

The property $prop_i_j$ verified on this model is encoded by the automaton on figure 5.

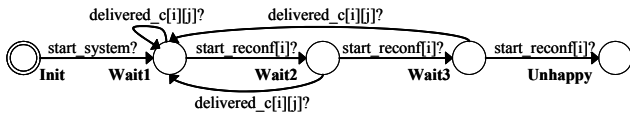


FIG. 5. Property observer $prop_i_j$.

Six $prop_i_j$ observers are added to the model. They allow to verify, in response to req_2 , that for all $1 \leq i, j \leq 3$, $i \neq j$, the following statement always holds : *there is a $delivered_c[i][j]!$ event at least every second $start_reconf[j]!$ event* (the *Unhappy* states of the $prop_i_j$ automata are unreachable). In this model the failure of the communication medium is considered as improbable, from the conclusions of the FT analysis. The full implementation is detailed in [2].

6 The Functional View

In this section the approach defined in section 4.3 is instantiated on the F view of the case study.

³the command laws partitions are not modeled, as they're not directly involved in the reconfiguration problem

6.1 Step 1 : Institution

The formal technique of the functional view is the synchronous data flow language LUSTRE [5]. The *Safety Logic* or *SL* [4] is a candidate for representing the technique in the framework.

SL is a propositional, linear, past tense temporal logic allowing the expression of safety properties, without reference to the initial state. A formal connection between *SL* models and boolean LUSTRE programs is given in [4]. In the following subsection, we give a short description of the signatures, models and satisfaction relation of the logic, with an emphasis on its models.

Signatures $Sign_{SL}$ and Sentences Sen_{SL} . *SL* specification signatures consist of a set of propositional symbols *Prop* (intuitively, LUSTRE boolean input data flows) and of a set of auxiliary propositional symbols *Aux* (intuitively, LUSTRE boolean internal and output data flows).

The sentences are defined using the **pre** operator, which refers to the previous truth value of any proposition, using boolean connectors and negation, and using a two level syntax : first, *past sentences* are defined, ranging over *Prop* and *Aux* symbols ; then, *safety sentences* are defined as quantified *past sentences*, in which every *Aux* symbol is under the scope of an existential quantifier.

Models Mod_{SL} . Let *Prop* be a set of propositional symbols. The models of *SL* are the sequences generated by the following transition systems :

$$\Pi = \langle Q, q_0, O, \lambda \rangle$$

with :

- Q : set of states
- $q_0 \in Q$: initial state
- $O = Prop^2$: set of *observable propositions*, here the power set of *Prop*,
- $\lambda \subseteq Q \times O \times Q$: transition relation.

A trace τ_{SL} is a sequence $\{\omega_0, \dots, \omega_n, \dots\} \in O^\infty$ such that there is a sequence $\{q_0, \dots, q_n, \dots\} \in Q^\infty$ such that $\forall i, (q_i, \omega_i, q_{i+1}) \in \lambda$. The set of traces generated by Π is noted $L(\Pi)$.

Satisfaction Relation \models_{SL} . The satisfaction of a formula by a propositional trace is defined using a fixed point criterion, involving the propositional trace and an auxiliary trace. More details can be found in [4].

6.2 Step 2 : Modeling With Lustre

The LUSTRE model of the system focuses on the *F* layer of the components. It consists of three LUSTRE programs $reconf_i_{1 \leq i \leq 3}$, taking as input the boolean data

flows $rec_i_j_k_{1 \leq i, j, k \leq 3}$, representing the event “ $Comp_i$ has received a $delivered_{j,k}$ notification”, taking the value *true* if a notification was received since the last execution, and *false* otherwise), and producing three boolean data flows $send_i_k_{1 \leq i, k \leq 3}$ taking the value *true* if $Comp_i$ must slave $Surf_k$, and *false* otherwise. LUSTRE allows the use of formal *assertions*, whereby a designer can express hypotheses in his design. In a verification perspective, *assertions* express known properties of the environment and are used to perform constrained verification.

The following reasoning is conducted to detect a failure : assuming that the communication layer actually meets its performance requirement, $reconf_i$ (running on $Comp_i$) considers that $Comp_j$ has failed if the notifications $delivered_{j,k}$, as seen from computer $Comp_j$, appear perturbed ; that is, if the data flow $rec_i_j_k$ remains *false* for more than k cycles (to be determined) of $reconf_i$ ⁴.

The following observer node returns true as long as the flow A never contains two consecutive *false* values : $never_false_twice(A : bool)$.

The fault of $Comp_j$ can then be diagnosed by $reconf_i$ as follows (arbitrarily taking $k = 2$) :

$$diag_i_fault_j = \neg never_false_twice(rec_i_j_k);$$

The details of the reconfiguration algorithm are not given here (but can be found in [2]), as they are not crucial to the illustration of our approach. However we detail the use of assertions in the verification.

Fixing $k = 2$ amounts to assuming that a notification, if emitted at all, always reaches its destination with at most one cycle (in the receiver time frame) of latency. Hence a computer can enter a fail state (interrupting its notification flow) and still have the other computers receive data from it within a duration equivalent to a partition cycle. This we must simulate in the verification program.

The data flows $x_i_j_k_{1 \leq i, j, k \leq 3, i \neq j}$, $fail_i_{1 \leq i \leq 3}$ and $fail_i_j_{1 \leq i, j \leq 3, i \neq j}$ are declared as free variables of the verification program, and the following assertions are stated :

- $assert(never_false_twice(x_i_j_k))$: asserts that the flow $x_i_j_k$ simulates the expected behavior of the communication layer ;
- $assert(\neg fail_1 \text{ or } \neg fail_2 \text{ or } \neg fail_3)$: here, $fail_i$ represents the *failure state* of computer $Comp_i$, and we assert that there is always at least one computer running.
- $assert(\neg pre(fail_i) \text{ or } fail_i)$: failures are definitive.

The input data flows of $reconf_i$ are then defined :

⁴The $reconf_i$ routine also monitors $Comp_i$, and even if *slaving_i* running on $Comp_i$ starts broadcasting notifications again after a temporary perturbation, it is forced to silence by $reconf_i$.

- $rec_i_i_k = \neg fail_i : Comp_i$ receives its own data with no latency as long as it has not failed ;
- $rec_i_j_k = x_i_j_k \text{ and } \neg fail_i_j$: here, $fail_i_j$ represents the *failure mask* used to simulate the perception of $Comp_i$ failure state by $Comp_j$; $Comp_i$ receives notifications from $Comp_j$ if the latter has not failed.

Extra assertions are necessary, to correlate *failure masks* to *failure states* :

- $assert(fail_j \text{ or } \neg fail_i_j)$: a computer cannot be seen faulty from the other’s perspective if it is actually not.
- $assert(\neg pre(fail_j) \text{ or } fail_i_j)$: states that the failure state of $Comp_j$ is propagated to other computers with a latency of at most one cycle.

The following properties are verified by model checking the verification program :

1. $\neg fail_i \Rightarrow send_i_i$: corresponds to **req31**, in nominal mode, $Comp_i$ slaves $Surf_i$,
2. $\forall j, (\neg send_1_j \wedge \neg send_2_j \wedge \neg send_3_j) \text{ xor } (send_1_j \text{ xor } send_2_j \text{ xor } send_3_j)$: corresponds to **req32**, mutual exclusion property on $Surf_j$;
3. $\neg(\neg(send_surf_i) \wedge \neg(pre(send_surf_i))) \wedge \neg(pre(pre(send_surf_i)))$
with $send_surf_j = send_1_i \text{ or } send_2_i \text{ or } send_3_i$: corresponds to **req33**, an aerodynamic surface never remains idle for more than three reconfiguration partition periods (inclusive).

7 Connecting Views to the Central Model

The connection of a dynamic view, which uses technique T , to the central model is represented using a class of mappings

$$embed_{I_T} : \mathbf{Sign}_{Comp} \rightarrow \mathbf{Sign}_{I_T}$$

associating component signatures in $Sign_{Comp}$ to I_T signatures. To be correct, a mapping must fulfill a set of constraints, the definition of which is left to the designers of the view and corresponds to **Step 3** of the approach. These constraints represent the way in which attributes from the static views are transferred and used for modeling the system in the technique T . A number of sets of constraints can be defined, to capture different modeling approaches in a given technique.

Following **Step 3** of our approach, designers are free to define any embedding mapping well suited to capture correctly their modelling approach. For illustration, two alternative embeddings of components in the RTP view could be defined :

1. a mapping is correct if it maps

- (a) a *broadcast channel* to each inter-partition communication port embedded in the view ;
- (b) an *urgent* or *standard channel* to each intra-partition communication port ;
- (c) the value of the *bcet* attribute of a partition P (in the F layer of a component) is mapped to the *bcet* parameter of the automaton A_P representing the partition.

2. a mapping is correct if

- (a) every communication port embedded in the view is mapped on a global shared UPPAAL variable ;
- (b) the value $wcet - bcet$ of a partition P is mapped to the parameter *jitter* of the A_P automaton.

When embedding components in the F view, a single modeling approach could be used : a mapping is correct if a partition P is mapped to a LUSTRE node, and that a communication port at the $E - F$ interface is mapped to a pair $\langle data, fresh \rangle$ of data flows, with *fresh* a boolean freshness tag for the value *data*. Figure 6(b) depicts those mappings.

8 Synchronization of Institutions on Models

The main purpose of our approach is to be able to conduct a formal verification of global properties spanning a number of heterogeneous views. In this institutional framework, this comes down to conjointly using two institutions $I_1 = \langle \mathbf{Sign}_1, \mathbf{Sen}_1, \mathbf{Mod}_1, \models_1 \rangle$ and $I_2 = \langle \mathbf{Sign}_2, \mathbf{Sen}_2, \mathbf{Mod}_2, \models_2 \rangle$ to perform heterogeneous reasoning, thanks to the *synchronization of institutions on models* construct [9, 10], whereby a new institution I is defined :

$$I = I_1 + I_2 = \langle \mathbf{Sign}, \mathbf{Sen}, \mathbf{Mod}, \models \rangle$$

such that :

- $\mathbf{Sign} = \mathbf{Sign}_1 \times \mathbf{Sign}_2$. New signatures are pairs of signatures ;
- $\forall \langle \Sigma_1, \Sigma_2 \rangle \in \mathbf{Sign}, \mathbf{Sen}(\Sigma) = \mathbf{Sen}_1(\Sigma_1) \setminus \{\top_1\} \cup \mathbf{Sen}_2(\Sigma_2) \setminus \{\top_2\} \cup \{\top\}$. A formula $\varphi \in \mathbf{Sen}(\Sigma)$ is either a $\mathbf{Sen}_1(\Sigma_1)$ formula, or a $\mathbf{Sen}_2(\Sigma_2)$ formula, and the formula “true” \top is appropriately redefined in the new language $\mathbf{Sen}(\Sigma)$;
- $\forall \langle \Sigma_1, \Sigma_2 \rangle \in \mathbf{Sign}, \mathbf{Mod}(\Sigma) = \mathbf{Mod}_1(\Sigma_1) \times \mathbf{Mod}_2(\Sigma_2), \perp = \{\perp_1, \perp_2\}$. The models of I are pairs of models, and the improper model \perp is appropriately redefined in $\mathbf{Mod}(\Sigma)$;

The new satisfaction relation is defined as follows :

$$\forall \langle \Sigma_1, \Sigma_2 \rangle \in \mathbf{Sign}, \forall m = \langle m_1, m_2 \rangle \in \mathbf{Mod}(\Sigma), \forall \varphi \in \mathbf{Sen}(\Sigma) : m \models \varphi \Leftrightarrow \varphi = \top, \text{ or } \varphi \in \mathbf{Sen}_1(\Sigma_1) \text{ and } m_1 \models_1 \varphi, \text{ or } \varphi \in \mathbf{Sen}_2(\Sigma_2) \text{ and } m_2 \models_2 \varphi$$

The new entity I is indeed an institution, however it doesn't serve our purpose yet, as no particular semantic constraint is established between I_1 and I_2 . To achieve the desired interaction, the satisfaction relation is kept unchanged, but a new class of models $Sync_{I_1, I_2}(\Sigma) \subseteq \mathbf{Mod}(\Sigma)$ is defined :

$$Sync_{I_1, I_2}(\Sigma) = \{ \langle m_1, m_2 \rangle \in \mathbf{Mod}(\Sigma) \mid P(m_1, m_2) \}$$

Expressing the property P corresponds to **Step 4** of our approach. P formally represents the semantic bridge between the techniques represented by I_1 and I_2 . Models of the pair $\langle m_1, m_2 \rangle \in Sync_{I_1, I_2}$ are semantically matched according to P . The synchronized institution is noted $I = I_1 \bullet I_2$.

9 Synchronization : Application to the Case Study

The system and both its LUSTRE model Π and its UPPAAL model \mathcal{M}_A can be seen as a single model belonging to the institution $SL \bullet XSL$, which formal construction is depicted on figure 6(c), as a diagram in the category of institutions (the two institutions are first stuck together without constraints to form $SL + XSL$, then synchronized according to $Sync_{SL, XSL}$). In this section we propose a synchronization relation $Sync_{SL, XSL}$, well suited to our application field and engineering purpose, and we illustrate its use on the case study.

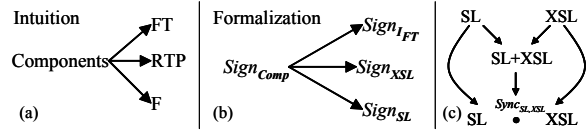


FIG. 6. Views, signatures embedding and synchronization.

The synchronization proposed here constitutes the **Step 4** of the approach on the case study, and involves a matching criterion between the two sets of traces $L(\Pi)$ and $L(\mathcal{M}_A)$.

9.1 Preliminary definitions

First, a few definitions necessary for defining the synchronization relation are given. Let \mathcal{M}_A and Π be such that $ObsAct \subseteq A \setminus \{\varepsilon\text{-actions}\}$ is a non-empty set of *observable actions*, such that $\forall a \in ObsAct$, there is a proposition $evt_a \in Prop$. The set of these *observed propositions* is noted $ObsProp \subseteq Prop$. The translation operator induced by this mapping is noted *mirror* : $ObsAct \rightarrow ObsProp$. We also consider a set of *key actions* $KeyAct \subseteq A \setminus \{\varepsilon\text{-actions}\}$, defining an ordered set of *key instants*,

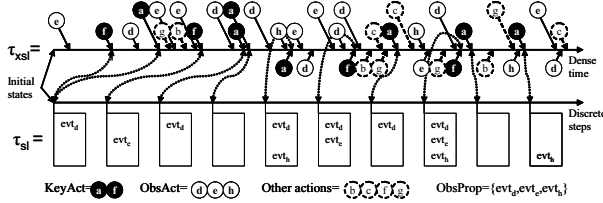
or *discrete timeline* $Dtl(\tau_{XSL}) = \{t_i\}_{i \in \mathbb{N}}$ for each trace $\tau_{XSL} \in L(\mathcal{M}_A)$.

The following operators are defined (the $-$ symbol represents argument places) :

- $[-]_{ObsProp} : L(\Pi) \rightarrow (ObsProp^2)^\infty$: restriction of τ_{XSL} traces to the elements of $ObsProp$;
- $\downarrow_{KeyAct} : L(\mathcal{M}_A) \rightarrow (A^2)^\infty$: *agglutination* operator, which, given a trace $\tau_{XSL} \in L(\mathcal{M}_A)$, associates with each $t_{i \geq 1} \in Dtl(\tau_{XSL})$ the subset of actions of A appearing in τ_{XSL} between the instants t_{i-1} and t_i .
- $[-]_{ObsAct} : (A^2)^\infty \rightarrow (ObsAct^2)^\infty$: restricts a sequence $\tau_{XSL} \downarrow_{KeyAct}$ to the actions of $ObsAct$;
- *Mirror* : $(ObsAct^2)^\infty \rightarrow (ObsProp^2)^\infty$: extends *mirror* to sequences of elements of $ObsAct^2$.

9.2 Example

This picture shows how the agglutination and mirroring operations transform the upper timed trace into the lower temporal trace. Key actions are mapped to discrete steps (dotted arrows). The first and second key actions occurring after the initialization of the timed model are mapped to the initial transition of the synchronous model. All observable actions between these two key actions are mirrored to the set of propositions associated with the initial state.



Note that another interleaving of actions between key actions would have led to the same propositional trace.

9.3 Definition and Signification of Sync_{SL,XSL}

The synchronization relation is given by $Sync_{SL,XSL} = \{(\Pi, \mathcal{M}_A) \in \mathbf{Mod}(SL) \times \mathbf{Mod}(XSL) \mid \text{Mirror}([L(\mathcal{M}_A) \downarrow_{KeyAct}]_{ObsAct}) \subseteq [L(\Pi)]_{ObsProp}\}$

The idea behind this synchronization is that the environment of the transition system Π (modeled as a LUSTRE program) is modeled by the timed automaton \mathcal{M}_A . For a given τ_{XSL} , the instants of the *discrete timeline* $Dtl(\tau_{XSL})$ correspond to instants at which Π fires its own transitions in the global time frame of \mathcal{M}_A . The firing of an a -labeled transition in the model \mathcal{M}_A appears, after agglutination by \downarrow_{KeyAct} and translation by *Mirror*, as a *true* value for the proposition evt_a to the Π system. All observable actions occurring between transitions $n-1$ and n of Π are seen as occurring simultaneously at transition n .

The relation $Sync_{SL,XSL}$ is defined as an inclusion of sets rather than an equality, as we consider the dynamics

of the *ObsProp* symbols to be governed by the timed model. The designer of the Π model may only have a partial understanding of its environment, and hence cautiously assume it less constrained than it is in reality ; Π thus tolerates a greater variety of event combinations.

9.4 Using Synchronization

A pair of models $\langle system, environment \rangle = \langle \Pi, \mathcal{M}_A \rangle$ is semantically coherent if \mathcal{M}_A produces traces matching the assertions of Π . This enables *heterogeneous assume-guarantee* reasoning to be carried out, as depicted on figure 7.

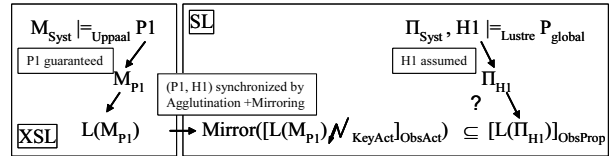


FIG. 7. Synchronization of a RTP property with an F hypothesis.

In the RTP view, a model \mathcal{M}_{Syst} of the scheduling and communication layer of the system allows us to establish a property P_1 , expressed in the XSL logic ; an automaton M_{P_1} models this property. In the F view, a model Π_{Syst} uses a SL assertion H_1 , modeled by Π_{H_1} , to establish the property P_{global} .

We consider that the property P_1 of the PTR view validates the assertion H_1 of the F view if and only if $\langle \Pi_{H_1}, M_{P_1} \rangle$ belongs to $Sync_{SL,XSL}$.

9.5 Case Study Example

In section 5, we showed by model checking that the *Unhappy* states of the observers $\mathbf{Prop}_{i,j}$ (cf. fig.5) are unreachable. Then we define, for each observer :

- $ObsAct = \{delivered_c[i][j]?\}$;
- $ObsProp = \{rec_i_j\}$;
- $mirror(delivered_c[i][j]?) = rec_i_j$;
- $KeyAct = \{start_system?, start_reconf[i]?\}$;

It is easy to see that $L(Prop_{i,j})$ is the set of traces starting with $start_system? \rightarrow \varepsilon(d)$ and followed by all the possible concatenations of the sequences :

- $delivered_c[i][j]? \rightarrow \varepsilon(d)$
- $start_reconf[i]? \rightarrow \varepsilon(d) \rightarrow delivered_c[i][j]? \rightarrow \varepsilon(d)$
- $start_reconf[i]? \rightarrow \varepsilon(d) \rightarrow start_reconf[i]? \rightarrow \varepsilon(d) \rightarrow delivered_c[i][j]? \rightarrow \varepsilon(d)$

Consequently, $Mirror([L(Prop_{i,j}) \downarrow_{Dtl}]_{ObsAct})$ is the set of sequences of elements of $ObsProp^2$ resulting from all the possible concatenations of the sequences :

- $\langle \emptyset, \{rec_{i-j}\} \rangle$
- $\langle \{rec_{i-j}\} \rangle$

These traces translate to LUSTRE data flows rec_{i-j} which never contain two consecutive occurrences of *false*, which are exactly the data-flows accepted by the *never_false_twice*($A : bool$) observer. The pair $\langle \mathcal{M}_{Prop_{i-j}}, \Pi_{never_false_twice} \rangle$ hence belongs to $Sync_{SL,XSL}$.

An UPPAAL observer was defined for each assertion of the LUSTRE model and checked for synchronization; the collection of synchronized $\langle assertion, property \rangle$ pairs validates the heterogeneous reasoning conducted in section 6. The system, as modeled here, satisfies the global requirements expressed in section 2.

10 Conclusion and Perspectives

The purpose of this work was to study the practical benefits that can be obtained through the use of formal approaches of this kind in the field of avionics systems. The *institution* paradigm provides a convenient and uniform way of dealing with the variety of formal techniques involved in complex systems modeling and analysis tasks. This approach suggests how one can tailor a framework to his needs, to document a system development (representation of the system incremental construction, together with heterogeneous views and models used for its analysis) in a clear and formal way, up to the heterogeneous semantics level.

A case study was developed in this generic framework. An avionics subsystem was specified and modeled in the real time performance and the functional views, using the heterogeneous techniques LUSTRE and UPPAAL. A synchronization relation between the views defines the semantic coherence necessary to carry out the verification of global safety properties across views. The verification was achieved in the F view using assertions, synchronized with properties from the RTP view. Figure 8 gives a representation of the case study in the framework.

In this synthetic diagram, each arrow carries a set of formal constraints, capturing how information travels between the different views. *Component signature pushouts* (plain arrows) represent component composition, *embedding mappings* (dotted arrows) $embed_{SL}$ and $embed_{XSL}$ connect the central component signatures to the model signatures. In each view, the development cycle is left unconstrained (reuse or refinement can occur), and the synchronization relation $Sync_{SL,XSL}$ explicitly defines semantic coherence between heterogeneous views, allowing the exportation of guarantees from one view to become assumptions for other heterogeneous views, in support for compositional reasoning.

We are currently working on the introduction of *consequence systems* in the framework, which would allow proof

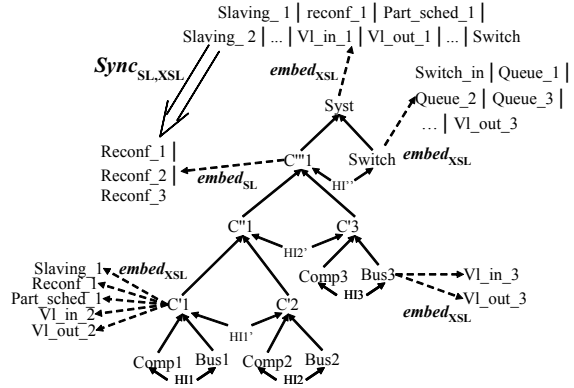


FIG. 8. Synthetic view of the case study.

oriented techniques to be used alongside model theoretic techniques.

Eventually, through the complete case study, we show the applicability of such approaches on systems of significant scale in the avionics domain.

References

- [1] Y. Ait-Ameur, B. d'Ausbourg, F. Boniol, R. Delmas, and V. Wiels. Principes méthodologiques pour le développement de systèmes critiques embarqués, rapport final n° 1.2/07361/dtim. Technical report, ONERA, august 2003.
- [2] R. Delmas. Un cadre formel pour la spécification hétérogène de systèmes avioniques : une étude de cas, rapport n° 1/31105/dtim. Technical report, ONERA, june 2003.
- [3] J. Goguen and R. Burstall. Institutions : abstract model theory for specification and programming. *Journal of the ACM*, 1(39), 1992.
- [4] N. Halbwachs, J. Fernandez, and A. Bouajjanni. An executable temporal logic to express safety properties and its connection with the language lustre, 1993.
- [5] N. Halbwachs, F. Lagnier, and C. Ratel. Programming and verifying real-time systems by means of the synchronous data-flow programming language lustre. *IEEE Transactions on Software Engineering, Special Issue on the Specification and Analysis of Real-Time Systems*, September 1992.
- [6] F. Laroussinie, K. G. Larsen, and C. Weise. From timed automata to logic - and back. In *Proc. 20th Int. Symp. Math. Found. Comp. Sci. (MFCS'95), Prague, Czech Republic, Aug.-Sep. 1995*, volume 969, pages 27–41. Springer, 1995.
- [7] K. G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a nutshell. *International Journal on Software Tools for Technology Transfer*, 1(1-2) :134–152, 1997.
- [8] J. Meseguer. General logics. In H.-D. E. et al, editor, *Logic colloquium 87*. Elsevier Science, 1989.
- [9] A. Sernadas, C. Sernadas, and C. Caleiro. Synchronization of logics. *Studia Logica*, 59(2) :217–247, 1997.
- [10] A. Sernadas, C. Sernadas, and C. Caleiro. Synchronization of logics with mixed rules : Completeness preservation. In *Algebraic Methodology and Software Technology*, pages 465–478, 1997.