CONSTRAINT-BASED DESIGN AND ALLOCATION OF SHARED AVIONICS RESOURCES

Laurent Sagaspe, ONERA, Toulouse, France
Pierre Bieber, ONERA, Toulouse, France

Abstract

We have studied tools and techniques to assist Integrated Modular Avionics (IMA) platform design. We propose an approach that helps to decide whether a set of systems can be implemented on an IMA architecture while enforcing safety requirements. To support the dialogue between teams in charge of defining system architectures and the avionics architecture designers the approach is based on the exchange of allocations constraints. The approach is made of three main steps: system designers describe formally how failures propagate in the system under study and derive segregation constraints, IMA designers collect the constraints and use a constraint solver to generate safe allocations, system designers use this allocation to combine the system failure propagation model with an avionics platform model in order to check quantitative safety requirements. The approach is supported by a set of tools including fault-tree analysers and constraint solvers.

Supporting the Dialogue between System Designers and IMA Platform Designers

Avionics systems of modern civil and military aircrafts tend to rely on IMA architectures where resources such as computers with real-time operating systems or Local Area Network communication switches can be shared by several systems such as Navigation, Flight Controls, Cabin Pressure Management. The IMA architecture is replacing the previous so-called federated architecture where a system design team was responsible for the development of the underlying avionics architecture that supported the system applications. In that context, resource sharing between systems was rather limited and the dependencies between systems were well-understood.

IMA is supposed to have several benefits, one of them is that the weight of the aircraft should be reduced as less pieces of equipment and wiring are needed. But, the IMA architecture has an important impact on system development for at least two reasons that we have taken into consideration:

- A new actor is added in the design process. The IMA design team is in charge of selecting adequate computing and communication resources and proposing an avionics architecture. So System design teams are no longer responsible for the definition of the avionics architecture.
- Resource sharing adds new dependencies between systems. For instance, the failure of a shared network node could lead to the loss or erroneous behaviour of several systems using the network.

New methods that support the design of avionics architecture should assist the dialogue between system designers and IMA designers. They should help the system designers to assess the safety and performances of a system under design while taking into account the impact of resource sharing. They should help the IMA designer to propose a platform architecture and to allocate safely system functions and data flows onto the platform resources.

The approach is based on formalized descriptions of system functional architecture, avionics platform architecture and the allocation of the functions on the resources of the platform. The descriptions are shared by system designers and IMA designers.

In the following of this paper we introduce the Terrain Following/Terrain Avoidance system as an example that illustrates the notion of formalized descriptions. Then we first explain how system designers can use these descriptions to perform a preliminary safety assessment of the system. Secondly we explain how the IMA designers can use the same descriptions to compute allocations of avionics resources to the system functions. Finally we propose an avionics platform design process that tries to combine efficiently the techniques introduced in this paper.

Shared Description of the case-study

The Terrain Following/Terrain Avoidance (TF/TA) system provides the flight controls or pilot of an aircraft with climb or dive signals such that the aircraft will maintain as closely as possible a selected height above the ground.

The following Figure shows the main tasks and data flows of the TF/TA system. We describe a functional architecture as a graph with the convention that boxes denote computation functions (called tasks) and hexagons denote communication functions (called data flows) .

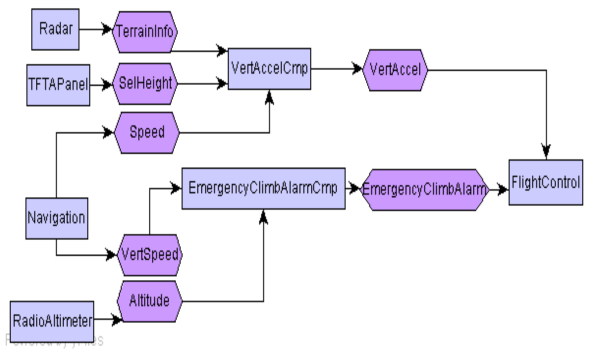

Figure 1. TF/TA Tasks and Data flows.

Before using the TF/TA system, the pilot enters through a dedicated panel (task TF/TA panel) the selected height of the aircraft. Using this information as well as Terrain information provided by the Radar and Speed data provided by the navigation system, a vertical acceleration is computed and sent to the flight control system. In parallel, an emergency climb alarm is computed based on the vertical speed data provided by the navigation system and on Altitude data provided by the radioaltimeter. This alarm is sent to the flight control so that the aircraft can climb quickly and reach a safe altitude.

An Avionics platform architecture is described by a graph with the convention that boxes denote computation resources (called CPU) and hexagons denote communication resources (called BUS).

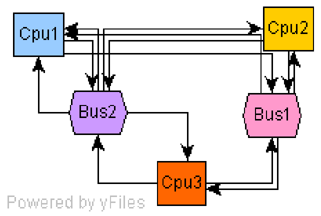

Figure 2. Basic avionics platform.

The allocation table given in table 1 describes how the previous platform architecture could support the TF/TA system.

Table 1. Allocation table

Resources	Task and data flows
Cpu1	Radar
	TFTAPanel
	VertAccelCmp
	Navigation
	TerraiInfo
	SelHeight
	Speed
Cpu2	RadioAltimeter
	EmergencyClimbAlarmCmp
	Altitude
Cpu3	FlighControl
Bus1	VertAccel
	VertSpeed
Bus2	EmergencyClimbAlarm

The descriptions presented in this section could be automatically extracted from the descriptions of system and platforms found in databases used by our Industrial partners. Furthermore the description format is compatible with new notations devoted to the description of software architectures such as AADL [1]

Formal System Safety Modelling and Assessment

Altarica Model

System Designers associate a formal model with each component of the system architecture that details its failure modes, the failure mode probabilities and the effect a given failure mode has on the component outputs. The failure modes we have considered are partial or total loss, or erroneous behaviour of a component. The formal model is based on the Altarica notation [2] developed by LaBRI (Laboratoire Bordelais de Recherche en Informatique). It was recently used by Dassault Aviation in the context of the Falcon 7X certification.

Each component, called a node, is made of the declaration of variables and events, and the definition of transitions and assertions.

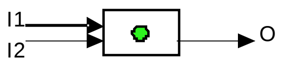

Figure 3. Node Function2 graphical representation

We illustrate these concepts with Component *Function2* that has two inputs I1 and I2 and one output O. In its correct state, this node computes a correct output when both of its inputs are correct, it does not compute any value if both of its inputs are lost otherwise an erroneous value is computed. If *Function2* node is in a correct state, an erroneous failure may occur and leads to an erroneous state. In this state, *Function2* compute erroneous values. Similarly the loss failure leads to the lost state where Function2 does not provide any value.

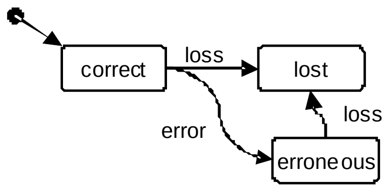

Figure 4. failure mode automaton of node Function2

The Altarica model of *Function2* node is presented in the following table.

Table 2. Function2 Altarica model

Node		
`Function2`		
State	**type**	**initial value**
`Status`	`FailureType`	`correct`
Flow	**type**	**direction**
`I1`	`FailureType`	`in`
`I2`	`FailureType`	`in`
`O`	`FailureType`	`out`
Event	**failure rate**	
`loss`	`exp 1.0e-4`	
`error`	*`exp 1.0e-5`*	
trans	**guard**	**assign**
		`Status`
`loss`	`Status != lost`	`lost`
		`Status`
`error`	*`Status = correct`*	*`erroneous`*
assert	**case**	**value**
`O`	`Status=correct and I1=correct and I2=correct`	`correct`
	`Status=lost or (Status != erroneous and I1=lost and I2=lost)`	*`lost`*
	`else`	*`erroneous`*

The row after **node** provides the node name, the rows after **state** declare state variables by giving their name (as `Status` in the previous figure), their value domain (`Status` is in user defined domain `FailureType)`and their initial value (`Status` is initially equal to `correct`). Domains we use are: Boolean, enumeration and record type built from Boolean and enumeration. For instance, domain `FailureType` is the enumeration `{correct, lost, erroneous}`.

Rows after **flow** declare variables that are used to model data exchanged with interfaced components by giving their name, type and direction : in for Inputs, out for Outputs.

Rows after **event** declare event names and their occurrence probability law.

Rows after **transition** contain the set of transitions describing events. A transition is composed of: the name of the event labelling the transition, for instance `loss`, a guard that is a Boolean expression based on state variables and input variables, defining in which conditions the transition may be triggered, and the set of state variable new assignments. In the previous table, a transition is associated with event `loss` that may only be triggered when component Function2 is not lost (i.e. `Status` is different from `lost`). The new value of `Status` is `lost.` The value of a state variable can only be modified by transitions.

Rows after **assertion** contain the set of output variables computation formulae. Computation formulae of an output variable may be based on state and input variables. The previous table defines how output `O` is computed. A nominal output is computed (`O=correct`) whenever `Status` and both inputs are equal to `correct` , When Functions2 is lost or it is in the nominal state but one of its input is lost then no output is computed (`O=lost`) otherwise an erroneous value is computed (`O=erroneous`).

From the description of the TF/TA system the Altarica model shown in figure 5 can be derived. This model is made of interconnected instances of Altarica nodes. The name of node instances and their interconnections is directly extracted from the Task and Data Flow graph of figure 1. The System Designer has to select from a node library what kind of formal behaviour should be associated with a task or a data flow. This library contains nodes that describe typical failure propagations. Function2 is associated with task EmerClimbAlarmCmp, Function3 that is similar to Function2 with one extra input is associated with verAccelCmp, Function1 that is similar to Function2 with one input less is associated with all the dataflows and Function0 that has no input is associated with all the remaining tasks but FlightControl. The node associated with FlightControl was developed ad hoc to describe precisely the various failure conditions of the TF/TA system.

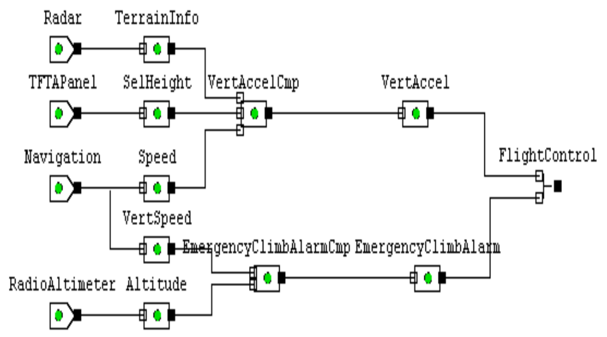

Figure 5. TF/TA Altarica model.

A second Altarica model can be built. It contains both the previous model, a model derived from the platform described in figure 2 and a model of the allocation table.

The platform architecture model is made out of the interconnection of node instances taken in a library of nodes that model CPU, BUS, Gateways.

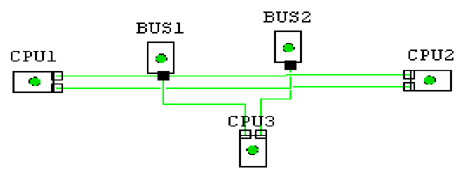

Figure 6. Platform Altarica model.

Allocation is modelled by defining global events that group events of the TF/TA and the platform models. For each resource R of the platform two global events RLoss and RErroneous are created. Event RLoss (resp. RErroneous) groups the `loss` events (resp. the `erroneous` events) of all tasks and functions that are allocated onto resource R according to table 1. For instance, event `CPU1Loss` groups events `Radar.loss`, `TFTAPanel.loss` and `VertAccelCmp.loss`.

A global event can be triggered if at least one event in the grouped events can be triggered (at least one guard should be true), when the global event is triggered all grouped events that can be triggered (their guard is true) are triggered simultaneously and related state variables are modified accordingly. So when `CPU1Loss` is triggered tasks `Radar`, `TFTAPanel` and `VertAccelCmp` become lost. The following picture shows on the combined model the effect of allocation: red icons are associated with tasks and data flows that are lost and red lines indicate flow variables that are equal to `lost`.

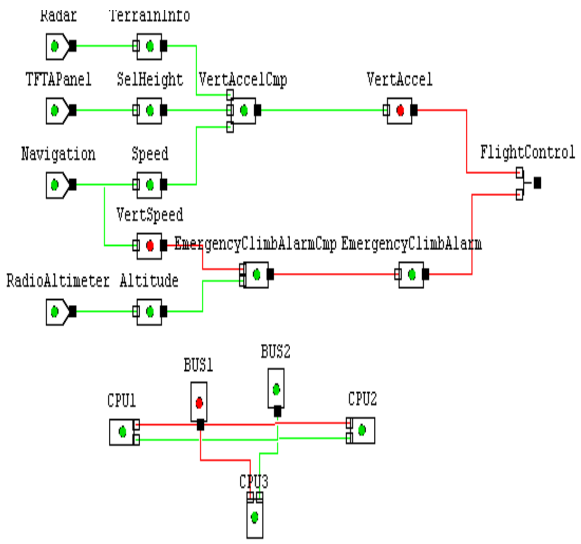

Figure 7. Combined TF/TA and Platform Altarica model when Bus1 is lost.

TF/TA Model Based Safety Assessment

The safety requirements that the system designers should assess are related with the three following Failure Conditions of the TF/TA system:

- **Undetected erroneous vertical acceleration** (TFTA_erroneous): In the Altarica model, `VertAccel.O` is equal to `erroneous` and `EmergencyClimbAlarm.O` is not equal to `correct`. This is the most safety critical situation as this could lead to the crash of the aircraft
- **Loss of the TF/TA system** (TFTA_loss): In the Altarica model, both `VertAccel.O` and `EmergencyClimbAlarm.O` are equal to `lost`. This situation is less critical as the pilot can disengage the TF/TA mode and take control of the aircraft. Depending on the workload of the crew this could lead to the loss of the aircraft or decrease the chance of success of the aircraft mission.
- **Spurious Emergency Climb Alarm** (TFTA_spurious): In the Altarica model, `VertAccel.O` is equal to `correct` and `EmergencyClimbAlarm.O` is not equal to `correct`. This situation is not safety critical but it could decrease the chance of success of the aircraft mission.

Safety requirements are of the form: "Failure Condition FC is classified SEVERITY. The failure rate of FC shall be less than QUANTITATIVE OBJECTIVE. No QUALITATIVE OBJECTIVE combination of events shall lead to FC" where:

- *SEVERITY* is a classification category, for safety the categories are: Catastrophic, Hazardous, Major, Minor and No Safety Effect. Other categories can be used, for instance a classification related with the fulfilment of a mission could include categories as Mission unsuccessful, Mission aborted, Mission delayed and No effect on Mission.
- *QUANTITATIVE OBJECTIVE* is a failure rate value that can be stated per flight hour or per take-off. Typical values for aircrafts are 10^{-9} for Catastrophic FC, 10^{-7} for Hazardous FC and 10^{-5} for Major FC. Other objectives can be defined with respect to mission success depending on financial penalties that have to be paid in case of mission delay or failure.
- *QUALITATIVE OBJECTIVE* describes the kind of combinations that have to be considered. In this paper the qualitative objective is given by a number *N* of individual failures in combinations of events. So we consider requirements of the form "*No combination of events with less than N individual failures shall lead to FC*" with N = 3 for Catastrophic FC, N=2 for Hazardous and Major FC and N=1 for Minor FC.

Table 3. TF/TA Safety Requirements

FC	**Safety**	**Mission**	**Qual**	**Quant**
erroneous	Haz	Fail	2	1e-7
loss	Maj	Abort	2	1e-5
spurious	No	Delay	1	1e-3

Fault tree generation tools associated with Altarica automatically produce the set of minimal scenarios of failure events that lead to the Failure Conditions. Fault tree tools also compute the probability of the set of minimal scenarios based on the failure rates associated to failure events in Altarica nodes. The reader could consult references [3] for a technical presentation of the principles of these tools.

Table 4 summarizes the safety assessment results. The first column gives the name of the failure condition considered, the second column indicates the type of safety result reported (*single* for the number of single failures leading to FC, *double* for the number of double failures leading to FC, *proba* for the probability of FC). The third column provides the results computed using the Altarica model restricted to the functional view of the TF/TA , whereas the two remaining columns give the results computed on the basis of global models that include the avionics platform model and two different versions of the allocation.

Table 4. TF/TA Safety Assessment Results

TFTA FC	**Result Type**	**TFTA Funct.**	**Global Alloc1**	**Global Alloc2**
Erroneous	*single*	1	2	1
	double	73	0	26
	proba	4.8 e-8	2.0 e-5	1.0 e-5
Loss	*single*	1	2	1
	double	77	0	6
	proba	1.4 e-6	2.0 e-4	1.0 e-4
Spurious	*single*	10	4	2
	double	0	0	0
	proba	5.5 e-4	2.2 e-4	1 e-4

Results for the TF/TA functional model indicates that quantitative requirements are met but qualitative requirements are not met as single failure `Navigation.error` leads to TFTA_erroneous and single failure `Navigation.loss` leads to TFTA_Loss. This problem could be solved by splitting the Navigation task into two independent tasks one would compute Speed and the other would compute VertSpeed.

Results for the global model indicate that the proposed allocation is not correct. The single failure `BUS1Loss` (resp. `BUS1Erroneous`) leads to TFTA_loss (resp. TFTA_erroneous) because loss (resp. erroneous behaviour) of Bus1 causes loss (resp. erroneous behaviour) of data flows VertAccel and TerrainInfo. To solve this problem the IMA design team should look for an allocation that preserve the independence of data flows VertSpeed and TerrainInfo.

Constraint-Based IMA Architecture Design and Allocation

Formalization of Allocation Constraints

Let *Task* be the set of tasks and *Data* the set of data flows that appear in the functional description, we use two constant functions to formalize the functional description: *orig: Data → Task* associates a data flow with its origin task and *dest : Data → Task* associates a data flow with its destination task. Let *Cpu* and *Bus* be the set of, respectively, computing and communication resources that appear in the platform description. We note *Res= Cpu U Bus* the set of platform resources. *Path* is the set of all finite paths in the graph of resources, a path is a set $\{r_1, \ldots, r_n\}$ such that resource r_i is connected to r_{i-1} and r_{i+1} in the platform architecture. The length of a path is the number of connected resources. The set of paths of length 1 is *Res*.

The main variable used to formalize allocation constraints is *allo: (Task U Data) → Path* , *allo(x)* is equal to *y* whenever *x* is allocated on resource *y*. We first explain various allocation constraints based on this variable that can be applied to any system.

Unique Allocation: Any task (resp. data flow) has to be allocated to one and only one computation resource (resp. communication path).

forall t:Task, exists c:Cpu, allo(t)=c

forall d:DataFlow, exists b:Path, allo(d) = b

We use auxiliary variable *uscnx: Res*Res → {0,1}* that is equal to 1 whenever the connection between two resources is used.

Used connections: A connection from cpu c to path b is used iff there exists a data flow d such that d is allocated to b and its origin task orig(d) is allocated to c. A connection from path b to cpu c is used iff there exists a data flow d such that d is allocated to b and its destination task dest(d) is allocated to c.

forall b:Path,c:Cpu, uscnx(c,b) = {c,b} : Path and (exists d: Data, allo(d)=b and allo(orig(d))=c)

forall b:Path,c:Cpu, uscnx(b,c) = {b,c} : Path and (exists d: Data, allo(d)=b) and allo(dest(d)=c)

uscnx is an auxiliary variables because it is defined in terms of the *allo* variable, it does not constrain the allocation of resources. It can be used to provide a quality measure on allocation solutions and to guide the search of good solutions. For instance, we could use as a search criterion the minimisation of the number of used connections: $\Sigma_{b:Path,c:Cpu}$ *length(b)*(uscnx(b,c) + uscnx(c,b))* as this should help to find architectures with the smaller number of useful communication and computing resources.

The system designers use the following families of allocation directives to constrain the possible allocations.

Segregation: Two segregated tasks (resp. data flows) shall not be allocated to the same cpu (resp. path)

forall, t1,t2: indep, not(allo(t1)= allo(t2))

Co-location : Two co-located tasks (resp. dataflows) shall share a common cpu (resp. a common path)

forall t1,t2: coloc, (allo(t1)= allo(t2))

Allocation : All tasks (resp. dataflows) in *F* shall be allocated to a *Cpu* (resp. a *Path*) in *R*.

forall t: F , exists r:R, allo(t)= r

Exclusion: All tasks (resp. dataflows) in *F* shall not be allocated to a *Cpu* (resp. a *Path*) in R.

forall t: F , forall r:R, not (allo(t)= r)

Tool support

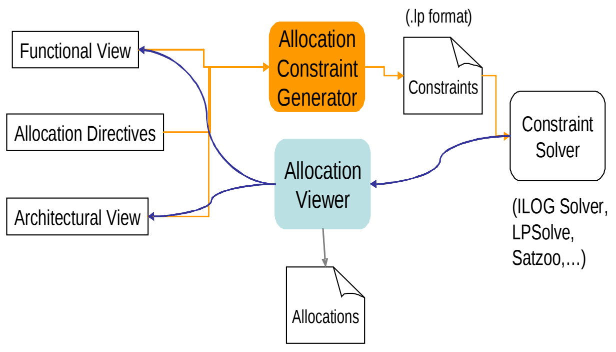

Figure 8 : Tool support for Allocation search and visualisation

The three inputs of the allocation search and visualisation tool are: A functional description and description of the platform architecture in GRAPHML format, and a set of allocation directives (segregation, co-location, allocation and exclusion) in a XML format that we defined.

The Allocation Constraint Generator reads the inputs and use them to generate a set of {0,1} Linear Programming Constraints as well as an optimisation criteria in the .lp format. The resulting file is sent to a Constraint Solver (either a general purpose solver as Ilog Solver or a specialized solver as Satzoo [4]) that tries to solve the constraints and find an allocation. The Allocation viewer function takes as input the allocation and shows graphically its effect on the function and architecture descriptions.

The Graphical User Interface has two main windows: one for the architecture description and the other for the functional description. When the user launches the allocation search, a pop-up window let the user select the allocation directive to be used. If no solution is found then the all functions and data flows of the functional view are coloured in Grey. Otherwise, the result of the optimisation criteria is displayed. Functions and data-flows are coloured with the colour of the resource that was allocated to them.

For instance, the following picture shows the allocation described in table 5. We can quickly see that Bus2 is allocated to data flows VertAccel, TerrainInfo, SelHeight, and Speed because they are coloured in purple. To investigate allocations in more details, the user can point with the mouse on a resource of the architecture and this highlights the function or data flow that was allocated to it. In the following picture, by clicking on Cpu2 box the user highlights the box Navigation, RadioAltimeter and EmergencyClimbAlarm. Finally, unused connections and resources of the architecture are coloured in Grey. For instance, the connection from CPU3 to CPU2 is not used in this allocation.

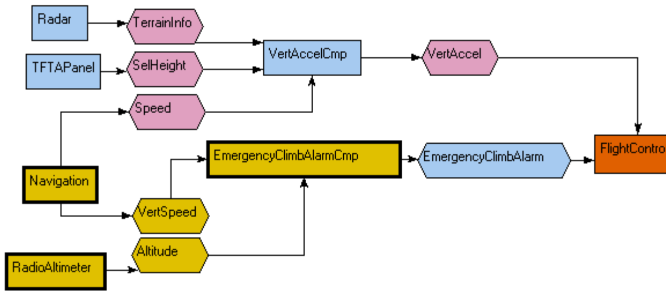

Figure 9 : Visualisation of an allocation – Functional view

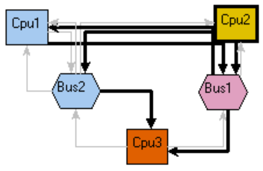

Figure 10 : Visualisation of an allocation Platform view

Combined System and Platform Design Process

We propose a design process that combines the model based safety assessment technique and the constraint based allocation generation we have presented in the previous sections. The three main steps of this combined process are:

1. System designers first generate from the formalized functional description a Functional safety model that addresses failure propagation between functions and data flows. Safety assessment techniques are applied in order to generate safety results (list of failure scenarios, probabilities, …). System Designers extract a set of allocation directives from these results. The directives are sent to the IMA design team.
2. IMA designers generate a set of constraints from the platform description and from the allocation directives it has received. A constraint solver is used to generate allocations of tasks and data flows of the system to the computer and network nodes of the architecture.
3. The allocation computed at the previous step is sent to the System designers that build a global safety model that extends the functional model with an architecture model derived from the formalized architecture description. The global model also contains a modeling of the allocation that was computed by the IMA designers. Safety assessments tools are used once again in order to provide safety results that can be compared with the results obtained with the functional model. From this comparison the System designer decides whether she accepts or rejects the proposed allocation. If the allocation is rejected, then new directives can be sent to the IMA design team in order to help find a more interesting allocation.

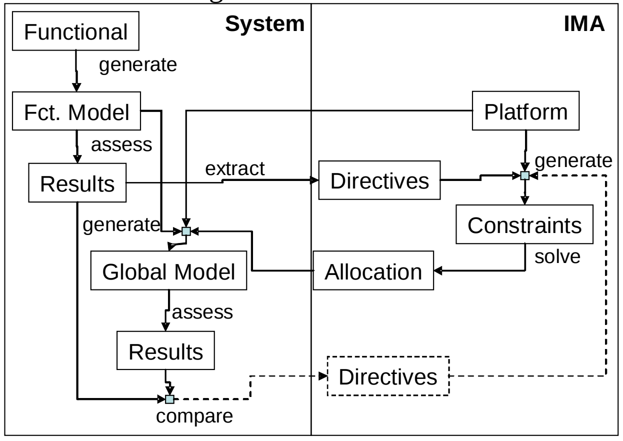

Figure 11 : Combined Design Process

In [5] we presented a method based on model-checking that allows to extract segregation directives from an Altarica model. In the following we present an alternative technique that automatically extracts segregation directives from a set of minimal scenarios generated by the Safety assessment tools.

For requirement TFTA_erroneous, the result file contains one single failure that we do not consider in this section and 73 double failures of the form ($\mathtt{Comp}_1.\mathtt{evt}_1$, $\mathtt{Comp}_2.\mathtt{evt}_2$) where $\mathtt{Comp}_i$ is the name of a node in the functional model and $\mathtt{evt}_i$ is either `loss` or `error`. If we want the computed allocation to preserve qualitative safety requirements then we have to provide directives that forbid the transformation of all these double failures into single failures. So $\mathtt{Comp}_1$ and $\mathtt{Comp}_2$ should be segregated, they should not be allocated to the same resource of the architecture.

Using this techniques we have generated 26 segregation directives that where used to compute the allocation in table 5. This allocation is better than the one proposed in table 1 with respect to qualitative requirements as it is evidenced in the last column of the Safety results table (Table 4). This allocation does not add any new single failure but the probability are still not good. To correct this problem one could add a new CPU and exclusively allocate Navigation on it.

Table 5. Allocation table

Resources	Task and data flows
Cpu1	Radar TFTAPanel VertAccelCmp
Cpu2	Navigation RadioAltimeter EmergencyClimbAlarmCmp VertSpeed Altitude
Cpu3	FlighControl
Bus1	TerraiInfo SelHeight Speed VertAccel
Bus2	EmergencyClimbAlarm

Using this process it can be the case that no allocation can be derived because the number of resources in the platform is not sufficient. We have investigated ways to indicate to the IMA designers which resources should be good candidates for change. We remove the segregation directives and use a new auxiliary variable *conflict(r,t1,t2)* that is equal to 1 when two independent tasks or data flows *t1* and *t2* are both allocated to resource *r*. For all the new allocations found and for each resource *r* , we compute $\Sigma_{t1,t2:indep}$ *conflict(r,t1,t2)* and we consider that resource with the highest score should be examined for splitting them into several independent resources. Another option, if the IMA design team is unable to cope with the constraints, is to ask the system designers to relax the segregation constraints of pairs of tasks and data flow that are always in conflict.

Conclusion

The proposed approach is consistent with industrial practices. In the context of civil aircraft certification, new safety analysis were required for IMA design that are similar to what can be performed using the technique described to assess the safety effect of a given allocation. Constraint based allocation generation is also under study at several industrial companies (aircraft and automotive manufacturers).

We are currently applying the proposed approach to a civil aircraft case-study in order to test the approach scalability. The first results indicate that the model-based approach can cope with industrial-size models (for instance, a platform made of 100 computing resources and 100 communication paths). Limitations of constraint solving tools are likely to require us to adopt an incremental approach in the allocation generation phase. For instance, one could start with a platform made of a small number of virtual communication and computing resources that are progressively refined if needed.

References

[1] P. H. Feiler, D. P. Gluch, J. J. Hudak, The Architecture Analysis & Design Language (AADL): An Introduction, *Technical Note,* CMU/SEI-2006-TN-011

[2] Arnold, A., A. Griffault, G. Point, A. Rauzy (2000). The AltaRica formalism for describing concurrent systems. In: Fundamenta Informaticae p109-124.

[3] Rauzy, A., (2002). Modes automata and their compilation into fault trees. Reliability Engineering and System Safety, 78:1-12, 2002.

[4] N. Eén, N. Sörensson, An Extensible SAT-solver, *SAT'03* proceedings, 2003

[5] L. Sagaspe, G. Bel, P. Bieber, F. Boniol, Ch. Castel, Safe Allocation of Shared Avionics Resources, High Assurance System Engineering (HASE), 2005.

Email Addresses

Laurent.Sagaspe@onera.fr , Pierre.Bieber@onera.fr