New Challenges for Future Avionic Architectures.

Pierre Bieber, Frédéric Boniol, Marc Boyer, Eric Noulard, Claire Pagetti

P. Bieber, F. Boniol, M. Boyer,
E. Noulard, C. Pagetti
(Onera)

E-mail: frederic.boniol@onera.fr

New Challenges for Future Avionic Architectures

Electronic sets operated on aircraft are usually summarized as “avionic architectures” (for “aviation electronic architecture”). Since the 70s, avionic architectures, composed of digital processing modules and communication buses, are supporting more and more avionic applications such as flight control, flight management, etc. Hence, avionic architectures have become a central component of an aircraft. They have to ensure a large variety of important requirements: safety, robustness to equipment failures, determinism, and real-time.

In response to these requirements, aircraft manufacturers have proposed several solutions. This article has a twin objectives: firstly to survey the state of the art of existing avionic architectures, including the IMA (for Integrated Modular Avionic) architecture of the most recent aircraft; and secondly to discuss two challenges for the next generation of avionic architectures: reconfiguration capabilities, and integrating COTS processing equipment such as multi-core processors. We believe that these two challenges will be central to the next generation of IMA architectures (called IMA-2G for IMA 2d generation).

Introduction

Aerospace systems

Avionic systems represent a growing part of aircraft costs: 35 to 40% in civil aircraft, and more than 50% in military aircraft. These systems are responsible for various applications, such as navigation, guidance, stability, fuel management, air/ground communications, passenger entertainment, etc. Their complexity is continuously growing (more and more functions to integrate, the aircraft becomes a real information system). In parallel, communication and information management technologies are evolving and new solutions for avionics are being invented. The implementation of the avionic systems of modern civil (B787, A350) and military (Rafale, Gripen, A400M, etc.) aircraft tends to rely on an IMA (for Integrated Modular Avionic) architecture instead of the more classical federated architecture. In a federated architecture, each system has private avionic resources, whereas in an IMA architecture avionic resources can be shared by several systems. The types of avionic resources that are generally considered are computers with real-time operating systems or local area network with real-time communication protocols.

An important consequence of the emergence of IMA in aerospace architectures is to allow development of x-by-wire[1] distributed applications, composed of a great deal of equipment (sensors, actuators, physical devices, software modules, memory modules, etc.) supported by the IMA platform. That leads to more and more complex systems, which are becoming increasingly difficult to validate.

Aerospace systems requirements

Aerospace systems are often characterized by two properties. Firstly, they are an information processing subsystem of their embedding systems, i.e., the aircraft. Second, they are reactive, i.e., they interact with their physical environment (e.g., the crew, the physical devices, etc.) at a speed imposed by the environment. Consequently, they have to meet a wide variety of constraints imposed by the embedding system and by the environment.

[1] x-by-wire is a generic term used to describe control systems that depend on a real-time communication network to connect different electronic components. Historically, these control systems relied on mechanical or hydraulic linkages, and the goal of x-by-wire is to replace nearly every automotive hydraulic/mechanical system with ultra dependable electronic systems. The “x” in “x-by-wire” denotes any safety-related application, such as steering, braking, flight control.

•*Safety constraints:* by nature avionic systems may cause severe damage if they malfunction. This possibility is dramatically increasing with the emergence of x-by-wire technologies, where critical applications (guidance and stability for instance) are totally managed by software modules. It is therefore of great importance to guarantee that such a system works correctly in all situations. For instance, let's consider a landing gear control system. The system is in charge of maneuvering landing gears and associated doors by controlling a set of physical devices, such as hydraulic jacks. Obviously, this system has to satisfy a list of requirements, such as "if the landing gear command button has been down for a given amount of time t, then the gears will be down in less than t time units". These properties characterize a real-time behavior of the control software together with the communication network between software modules and equipments. Due to the critical nature of the system it is necessary to guarantee that, whatever the behavior of the rest of the system (i.e., other applications), the right orders are sent at the right time to the right actuators (hydraulic jacks, gears, doors, etc.).

•*Dependability constraints:* by nature, embedded equipment may fail (with a given probability). Obviously, catastrophic failures are not acceptable at run time above a given probability level. It is therefore necessary to ensure that the system continues to work well (possibly in a degraded mode) even in the event of equipment failure.

•*Real-time constraints* are of great importance when dealing with x-by-wire dynamic systems. The total time delay is important for stability and performance of closed loop control of the flight control system, the automatic pilot system, the flight management system, the braking system, etc. The time lag between data input and the corresponding output (orders to actuators for instance) are due to (a) computing, (b) communication latency through the "wires" (i.e., the buses, the communication switches...), and (c) storage time. Typically, the inputs are pre-processed before being used in the main computational cycle, and the output will be post-processed after being computed. The pre- and post-processing may involve unit conversion, reasonableness checks, comparison with input from other computers (voting, etc.), and integrity checks on the communication links. In the IMA context, this pre- and post-processing are supported by shared computing resources. Similarly, the main computational cycle may involve several software modules running concurrently on IMA-shared resources and managed by real-time scheduling policies. Due to resources sharing, the time required for these activities may vary but could increase the effective lag between input and output orders above the value corresponding to the specified iteration rate. That could impact the performance of the closed loop control system, and consequently the stability of the aircraft. Guaranteeing and verifying real-time constraints are major tasks when designing and validating aerospace systems.

Safety, dependability and real-time constraints have a direct impact on architecture design at aircraft and system level, and on the validation/certification process. Hence, aircraft manufacturers have to show compliance with international regulations using means that have been accepted by the certification authorities. This includes showing that safety requirements are enforced, establishing the predictability of communication and computing real-time performances and developing software and hardware according to strict development guidelines. This context makes designing and verifying aerospace systems, and particularly avionic architecture, a substantially different and more difficult task than for classical systems and software.

In response to this difficulty, aerospace researchers and engineers have developed two successive families of avionic architectures: federated architectures and IMA architectures.

Brief history: from federated architectures to IMA

From federated architectures…

The first avionic devices to be embedded in aircraft were radios for communication and navigation in the 1940s. Since this period, analog and digital electronic controllers began to replace mechanical aircraft functions and equipment. One of the most popular examples of digital embedded systems is the flight control system (FCS), automatically controlling the trajectory of the aircraft according to the pilot's navigation orders. Until the 1990s the global architecture of the avionic system (including the FCS) was designed in accordance with the "federated architecture" principle: "one function = one computer". Figure 1 depicts an example of a federated architecture: the architecture of the A340 FCS. This system is composed of several software functions (FCPC, FCSC, etc.). Each function runs on a dedicated resource connected to its dedicated sensors and actuators. In other words, each function owns its dedicated equipment, including computing resources, sensors and actuators. Consequently, each function + resources set could be considered as a standalone subsystem. The main advantage of this so-called federated architecture is that it has quite limited resource sharing, and the dependencies between subsystems were well understood.

Up to the end of the 1990s, most of the civil aircraft, including the Airbus family A320, A330 and A340, were based on the federated architecture principle. However, since the 1990s, the airlines want more and smarter functionality, such as precise flight management capabilities, on board maintenance systems, larger entertainment systems for passengers, etc. The concept "one function = one dedicated subsystem" could no longer be maintained. The concept met its natural limit when the weight and volume of the dedicated subsystems hit the envelope restrictions of the aircraft. Another drawback became obvious: the huge number of different resources significantly increased the maintenance costs for airlines in terms of worldwide computer spares provisioning and handling. So a new approach was needed.

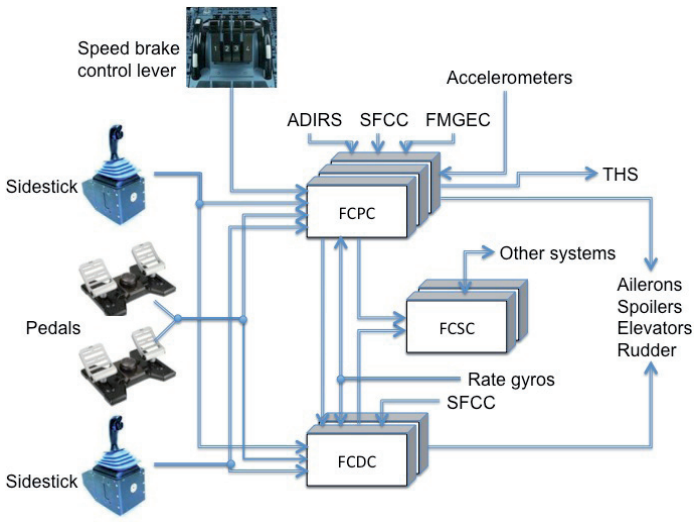


Figure 1. Example of a federated architecture: the A340 Flight Control System architecture

… to IMA-1G architectures

This new approach, called IMA (Integrated Modular Avionics), is based on two complementary principles. The first principle is to integrate multiple software functions with possibly different criticality levels on single avionic computing resources in order to keep the weight, volume and cost of the avionic architecture within reasonable limits. However, due to resource sharing, this first idea leads to side-effects and non-functional dependencies between avionic applications. Troubleshooting and modifications become very difficult. Moreover, proving that the system behaves safely requires knowledge of the whole system. It cannot be done in an incremental way function by function, as in previous federated architectures, and design and the certification process become a nightmare.

So a second principle is needed to simplify the design process and receive certification: strict and robust partitioning, i.e. a set of principles implemented by hardware and software means (such as middleware) which prevent interference between functions, exactly as if each function runs on its own virtual resources with guaranteed performances whatever the behavior of other functions.

IMA-1G (for IMA first generation, developed and certified for A380 and B787) is based on these two principles.

IMA-1G: description and main principles

Resource sharing and robust partitioning are the central ideas of the IMA concept. They are based on two standards: ARINC 653 [1] which defines partitioning principles in processing modules, and ARINC 664 [2] which defines partitioning principles for communications between functions.

Processing module partitioning

ARINC 653 specifies the management of avionic applications on common processing modules. As has already been explained, an avionic application is composed of several software functions running on different processing modules. According to ARINC 653 principles, functions from different applications resident in a processing module are partitioned with respect to space (resources partitioning) and time (temporal partitioning).

- *Resources partitioning*

Each partition is allocated a set of spatial resources (memory, non-volatile memory, I/O resources, etc.) in a static manner, that is to say that the module integrator has the task of assigning maximum allowed resources to each partition while respecting space segregation between them. Low-level mechanisms (at operating system level) provide protection for partition data against any modification from the other partitions. They monitor function activity with reference to allowed resources which are statically allocated through configuration tables.

- *Temporal partitioning*

The scheduling of functions on each module is defined off-line by a periodic sequence of slots statically organized in a time-frame named the MAjor time Frame (MAF). Each function is allocated a time slot for execution. At the end of this time slot the partition is suspended and execution is given to another function (from another application). Thus, each function periodically executes at fixed times.

Functions become in this manner totally independent, where faulty ones can be isolated without affecting much of the system integrity.

Communication resources partitioning

Arinc 664 describes the management of communication resources (i.e., the communication network). Communication flows are statically segregated into Virtual Links (VL). Each VL is dedicated to a single function and implements a traffic shaper. It is characterized by a Bandwidth Allocation Gap (BAG), i.e., the minimal time interval separating two successive messages on the VL. This principle has been implemented in the Avionics Full Duplex Ethernet (AFDX) architecture. AFDX constitutes one of the major technological breakthroughs in the avionics of the A380. In effect, and for the first time for such an aircraft category, the avionic system is based on a redundant and reliable Ethernet network. Key criteria for the choice of AFDX technology were avionic-specific constraints (security, temporal problems), the arrival of Ethernet switching and the size of the computer market. The final choice is therefore the switched Ethernet (full-duplex mode). The AFDX standard defines the electrical and protocol specifications for the exchange of data between Avionics Subsystems. One thousand times faster than its predecessor in the old federated architectures: the ARINC 429 bus.

A typical IMA platform is described in figure 2. Its hardware architecture consists of a set of computing processing modules (called CPM) that are connected to ARINC664 communication switches. CPM are grouped into clusters so that all the CPM in a cluster are connected to the same communication switch. Avionics applications (labeled A1 to A3 on CPM1, B1 to B3 on CPM2, etc.) are hosted in the partitions running on the computing modules. Data flows exchanged by applications hosted on different computing modules are transmitted through communication switch paths that connect the two computing modules.

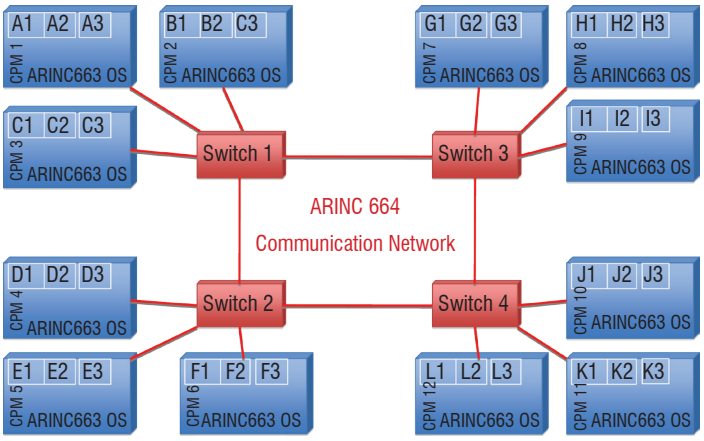


Figure 2. Typical IMA-1G architecture

The two standards ARINC 653 and ARINC 664 globally define the IMA concept that has been implemented in the Airbus A380 and the Boeing B787 for instance.

IMA-1G: benefits and impacts

Several benefits and impacts of IMA are discussed in [12]. The benefits are mainly weight and power consumption reduction. However, new difficulties arise: understanding and side-effects and dependencies between applications due to resource sharing.

Weight and power consumption reduction

Since IMA makes use of shared computing resources, the circuitry that once was contained within each federated resource (dedicated

to a given function) is now contained within a common IMA platform. For instance, computing processors that were duplicated in the federated case for fault-tolerance are replaced by a common set of IMA processors (and the associated infrastructure such as power, cooling, and redundancy mechanisms). Similarly, dedicated communication links are replaced with common communication channels (and the associated wiring). And finally, dedicated I/O interfaces are replaced by common I/O interfaces (and the associated wiring).

Globally, IMA results in a reduction in the required physical resources. Reduced physical resources translate to global weight and power savings for the aircraft. For instance, the number of processing units in the A380 is half that of previous generations. Reductions in airline operating costs are expected to be significant, with the decrease in the number of computers and cables (for power supply or communication) contributing to a reduction of aircraft weight leading to better fuel consumption efficiency. In the same way, fewer types of equipment will mean the airline has to buy and store fewer types of spare parts, which should lead to savings in maintenance costs.

Side-effects and dependencies between applications

The IMA architecture has a considerable impact on system development, as it is no longer possible to develop a system or a subsystem without considering its dependencies due to resource sharing with other systems. Of course, in federated avionic systems, functional dependencies between applications must be identified and taken into account. For instance, several aircraft applications, such as flight controls, depend on the navigation system or on the radioaltimeter system for altitude data. Hence, the impact of the loss of navigation or radioaltimeter data is taken into account when developing the flight control system. But resource sharing adds new indirect dependencies, similar to side-effects, between systems or subsystems. With regard to system safety, shared resources might cause common-cause failures. In the same way, resource sharing might cause unpredictable (or at least difficult to predict) overloads, implying delays and possibly a missed deadline or loss of information at run time. Let's consider the example of navigation data, produced by the navigation function for the flight control system. Assume that navigation data is allocated to a communication bus that is also in charge of transmitting diagnostic and maintenance information. It could then happen that, if a failure occurs somewhere in the aircraft (even on a non-critical equipment), the alarm and maintenance information overloads the communication network, leading to delays or to the navigation data not being transmitted. Such a scenario might lead to a catastrophic situation, although the initial single failure is of minor importance.

More generally, multiplexing all avionics communication flows on standard COTS communication resources would not have maintained the guarantees (in terms of guaranteed bandwidth, segregation, determinism) provided by dedicated avionic buses (like the ARINC 429 standard used in the federated architectures). For instance, traffic confluence inside Ethernet switches lead to variable latency. Hence, without any further assumption on a limitation of the communication flows, the occupation rate of each output port of each communication resource in the network is not predictable. Therefore, end-to-end latencies through the network are not predictable.

The second important benefit of the AFDX architecture is that, if functions are statically allocated in modules and if all functions respect their communication contract (i.e. the bandwidth allocation gap associated with each virtual link), then it is possible to mathematically prove that

- *no message is lost during the communication (i.e. no queue will overflow)*,
- the end-to-end delay of any message is bounded, and it is possible to evaluate an over-approximation of this bound.

These assertions do not guarantee absolute behavior determinism, but only a weaker form of determinism, which is sufficient to bring the guaranteed service required by essential avionics systems. Moreover, a "last but not least" benefit is that the certification process of the IMA communication network is (partly) based on this mathematical proof. This proof is based on the network calculus theory [10] (see also Chap. 15 of [18]). Network calculus is a recent analytic technique dedicated to switched communication networks. Input and output flows of the network are defined by positive increasing functions. Each node in the network (the switches) is then defined by its service curve [8]. Informally, the service curve determines the quantity of information, which may have been served until a given duration. Such a theory allows the formal determination of upper bounds, such as queue capacity required in each switch, and global maximum response time needed for each flow to cross the network. Network calculus has been used for certification of the A380 avionic network, and is now integrated in the Airbus network development process.

Next steps: two new challenges for IMA architectures

Towards reconfiguration capabilities

As explained above, IMA architectures have been defined to design avionics platforms that share communication and computation resources according to the two standards ARINC 653 and ARINC 664; these two standards impose static and fixed allocations. However, it could be interesting, in the event of a hardware failure for example, to be able to reconfigure the system, which means reallocating functions to safe modules. To meet this objective, the next generation of IMA platforms are to include reconfiguration capabilities in order to limit the effect of hardware failures on aircraft operational reliability. Such a reconfiguration capability is one of the next great challenges for avionic architectures.

Onera, in collaboration with Thales and Airbus, has explored the reconfiguration issue for IMA architectures in the European SCARLETT project [2]. The solution proposed for introducing reconfiguration capabilities in the next generations of IMA architectures is developed in section 3.

Towards high performance IMA architectures

The last decade has seen the emergence of multicore and manycore architectures, i.e. chips integrating several cores. These architectures are replacing the "old" monocore processors in all commercial domains, including avionics. The integration of monocore processing modules in IMA architectures is becoming more and more expensive because of the ongoing scarcity of these components.

[2] http://www.scarlettproject.eu/, point of contact: didier.hainaut@fr.thalesgroup.com

As a consequence, multi and manycore processors will be unavoidable in the next IMA-2G architecture.

They also offer promising opportunities for airframers due to their high level of computing power. The use of more flexible and lighter structures, for instance, means embedding far more complex flight control systems needing short response times and huge computations. It is expected that suitably controlled multi or manycore systems will provide the appropriate increase in computation power needed by these complex applications, paving the way towards greener aircraft by reducing aircraft structure weights.

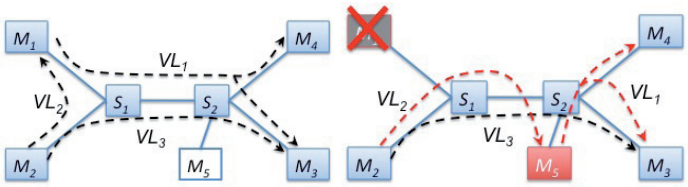
However, embedding multi or manycore architectures is a real challenge because they make it hard to ensure time predictability due to intensive resource sharing and because they do not satisfy the same fault model as monoprocessors.

Onera, in collaboration with Thales and Airbus, has explored this challenge. A first solution has been proposed for embedding predictable multicore architectures in avionic systems. This first step towards high performance IMA architectures is discussed in § "Second challenge".

First challenge: towards a reconfigurable architecture

Objectives and main ideas

Reconfigurable IMA should be able to change the configuration of the platform by moving applications hosted on a faulty computing module to spare computing modules.



(a) Initial configuration (b) Configuration after losing module M1

Figure 3. Reconfiguration example

Let's look at the notion of reconfigurable IMA with a simple example: the platform is composed of five modules ($M_1 \ldots, M_5$) and two communication switches ($S1, S2$). The initial configuration is drawn in figure 3(a): module M_5 is a spare initially shut down and free of application (in white in the picture). If some failure occurs on module M1 then the applications initially hosted on this module can be reconfigured on the spare M_5, and all communications from and to M_1 ($VL1, VL2$) have to be rerouted according to this new allocation. The configuration obtained is shown in figure 3(b) (reallocations are shown in red).

The main goals to be achieved by the reconfigurable IMA platform are to:

- Improve the operational reliability of the aircraft while preserving current levels of safety;
- Avoid unscheduled maintenance and associated costs;
- Limit the impact of reconfiguration on certification practices and effort.

Aircraft systems have to enforce stringent safety requirements that address the effects of failures on the life of passengers. To satisfy these requirements a minimum level of redundancy is associated with an application on the basis of the severity of the effects of its loss. For instance, three occurrences of an application managing cabin air pressure would be required because loss of cabin pressurization is catastrophic whereas no occurrence of an application managing in-flight entertainment is required, as it is considered as a comfort application the loss of which has no safety effects.

Operational reliability addresses the effect of failures on economical aspects of flight operations. One source of improvement is to decrease the number of flight delays or cancellations caused by faulty computing modules. Before each flight, the health status of all equipments is assessed in order to check whether for all applications the correct level of redundancy is available. If this is not the case the aircraft cannot be used (NOGO). It should be possible to restore the minimum level of redundancy by moving the applications running on the faulty module to a non-faulty one. This should also help to defer maintenance operations until the aircraft has reached an appropriate location.

Reconfiguration constraints and principles: a first proposal

According to the IMA standards, several functions are available in order to manage the platform, they include a:

- *Data-Loading* function that stores all the application software and loads the application software in accordance with the allocation of applications onto the computing modules;
- *Monitoring and Fault Detection* function that constantly receives information about the health of the hardware components and is able to detect that a component is faulty;
- *Power Supply Management* function that is able to switch on and off the power supply of the various hardware components of the platform.

For reconfigurability purposes, a new function, called the *Reconfiguration Supervisor* (supervisor for short), needs to be embedded in the aircraft. The role of the supervisor consists in determining when a reconfiguration can occur and in performing a "correct by construction" modification of configurations in order to reach a better and safe state. The supervisor behavior is described by the following:

1 - Triggering a reconfiguration

When a computing module fails, a reconfiguration can be launched if this failure has an operational reliability impact, meaning that the aircraft becomes NOGO. The Monitoring and Fault detection function detects a NOGO module failure and sends this event to the supervisor. First, the supervisor applies usual maintenance procedure to check that the failure cannot be repaired by a simple reset of the module. For this it interacts with the Power Supply Management and the Monitoring and Fault detection to check if the reset has repaired the module. If the reset works then the module restarts the hosted avionics applications and the failure is corrected; otherwise the supervisor performs the next steps.

2 - Selection of a correct configuration

When the failure is confirmed the supervisor must determine the current state of the platform in order to select the next configuration. The sorting takes into account the side-effect on aircraft and the duration of the reconfiguration execution. The selection process is the following:

• *Set of configurations*

We note *Application* the set of applications hosted by the platform, *Cluster* the set of clusters of the platform, *Basic_Module* (resp. *Spare_Module*) the set of modules (resp. spare modules) in a cluster and *Basic_Partition* (resp. *Spare_Partition*) the set of partitions (resp. spare partition) running on a module. Let us denote $bm=|Basic\_Module|$, $sm=|Spare\_Module|$, $c=|\text{Cluster}|$, $f=|\text{Application}|$, $bp=|Basic\_Partition|$ and $sp=|Spare\_Partition|$.

Definition 1 (Configuration)
A configuration is an allocation of avionics applications on computing elements and it is represented by a function

$$Conf: Application \rightarrow Computing\_Element$$

where the set $Computing\_Element$ *is defined by* $Cluster \times (Basic\_Module \cup Spare\_Module) \times (Basic\_Partition \cup Spare\_Partition)$.

For module level reconfiguration, the identifier of the partition for an application remains unchanged wherever the application is allocated. Therefore, we can optimize the set to be $Computing\_Element=Basic\_Module \cup Spare\_Module$. In figure 1, avionics application *A1* is allocated in its initial partition on the first module of the first cluster. Thus $Conf(A1)=(1,1,1)$. For module level reconfiguration, there are exactly $(c(bm+sm))f$ configurations. This corresponds to the number of integer functions $[1,f]\rightarrow[1,c(bm+sm)]$. For partition level reconfiguration, there are exactly

$$A^{f}_{(c(bm+sm)(bp+sp)} = \frac{(c(bm+sm)(bp+sp))!}{(c(bm+sm)(bp+sp)-f)!}$$

configurations. This corresponds to the number of injective integer functions $[1,f]\rightarrow[1,c\times(bm+sm)\times(bp+sp)]$. All these configurations are known at design time. Note that we only consider nominal configurations and not degraded ones where an avionics application may not be allocated because there are not enough fail-free computing elements. The sequences of possible reconfigurations starting from an initial configuration can then be represented by a directed acyclic graph.

• *Reconfiguration policy*

The reconfiguration policy defines generic rules to be followed. It is chosen off-line and impacts the on-line selection process of the next configuration. For instance, we can decide that there is no priority among avionics applications and then, once a spare has been occupied, no other application can be hosted on this spare. Or we could associate a priority level with the applications and then a reconfigured application can be removed to leave the spare to a failed application with higher priority level. Another rule can give an order on the spares. The policy would then consist in reallocating on the first spare if it is available, otherwise on the second if this one is available and so on.

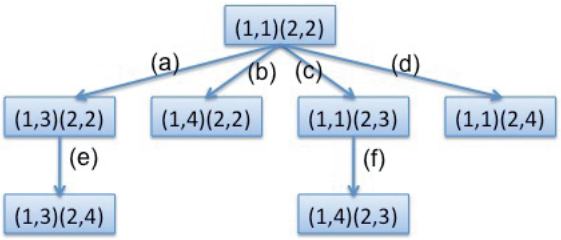
Let us consider the architecture $f=2$, $c=1$, $bm=2$, $sm=2$ where a module can host at most one application. For a module level reconfiguration we obtain several directed acyclic graphs of reconfigurations depending of the policy. For any of these graphs a node corresponds to a configuration, which is a list of pairs (number of application, number of hosting module). Below are shown the graphs associated with two policies that both order spare modules:

On the first graph (figure 4(a)), spares are not reconfigured, meaning that if a spare fails then hosted applications are lost. While on the second graph (figure 4(b)) they can be reallocated. For instance, transition (g) corresponds to a reconfiguration consecutive to the failure of the spare module 3. Reconfiguration graphs are different. Transitions (g) and (h) are missing on the left side graph because, according to this policy, when application 1 is running on spare 3 and this spare fails it is not possible to move 1 to spare 4.

• *Resource and real-Time constraints*

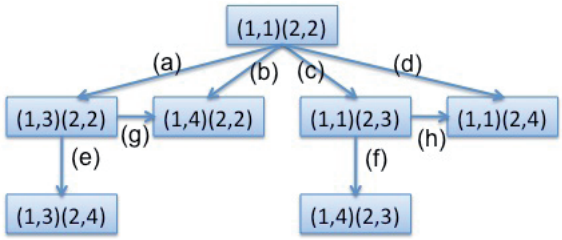
A configuration is safe if it satisfies some constraints. For instance, an application can be hosted on a module only if it provides adequate resources for the application such as processing power or memory. There are other kinds of constraints, such as segregation, that are described in [14].

A transition is safe if the intermediate steps are safe (they do not impact the integrity of the aircraft) and the duration of the transition is bounded by an appropriate value. The transition from one configuration to another is done by applying several basic procedures. All such elementary procedures are stored in some repository associated with a WCET (worst case execution time). For instance, data-loading a complete module is an elementary step and always takes less than a bounded amount of time, which is computed off-line. Globally, major reconfigurations done on the ground during two consecutive flights must take less than 15 minutes in



(a) Reconfiguration after module 1 failed
(b) Reconfiguration after modules 3 and 1 failed
(c) Reconfiguration after module 2 failed
(d) Reconfiguration after module 3 and 2 failed
(e) Second reconfiguration after module 2 failed
(f) Second reconfiguration after module 1 failed

(a) First policy: without spare reallocation



(a) Reconfiguration after module 1 failed
(b) Reconfiguration after modules 3 and 1 failed
(c) Reconfiguration after module 2 failed
(d) Reconfiguration after module 3 and 2 failed
(e) Second reconfiguration after module 2 failed
(f) Second reconfiguration after module 1 failed
(g) and (h) Second reconfiguration after spare module 3 failed

(b) Second policy: with spare reallocation

Figure 4. Examples of reconfiguration graphs

order to limit the flight delay. Minor reconfigurations done in-flight must complete more quickly according to the real-time requirements of the functions to reconfigure (generally less than several milliseconds).

- *Continuity of service*

When a module fails, the other avionics applications should not be impacted by a reconfiguration. In the case of partition level reconfiguration, the interactions between the supervisor and modules involved in the current reconfiguration must be transparent for the other partitions. For instance, data-loading and initialization must be realized during the partition time. In the case of distant reconfiguration, the routing of the impacted switches must be modified while ensuring the continuity of the remaining traffic.

3 - Reconfiguration execution

If a correct transition, with respect to the avionics constraints and the reconfiguration policy, has been found, the reconfiguration is performed. Basically, a reconfiguration is broken down into elementary steps: power up a spare module or a spare partition on a running module, test that the spare is fail-free and load the time application sequencing (at module level), data-load the code and initialize the partitions on the spare (during predetermined time slots for partition level), then verification by Monitoring and Fault detection that the spares are working correctly. A notification and report are sent to the maintenance terminals.

The sequence execution should be transactional and secure, i.e. the sequence should entirely succeed or totally fail. For this purpose, each step is acknowledged (succeeded, failed, not performed). Any step can be aborted without a side-effect on aircraft safety, performance and security. For instance, an access to an already allocated memory must be refused by Monitoring and Fault detection.

Discussion: mid-term and long-term perspectives

About the certification issue

Certification practices require safety assessments and showing real-time predictability for all configurations of a system. Two approaches can be considered. The first one consists in validating all the possible configurations. Currently, in an IMA-1G platform there is a unique configuration, which is completely certified. Because of the large number of reachable configurations (for c=4, bm=5, sm=1 there are 1296 configurations for local reconfiguration and 146001 for distant reconfigurations) the certification of an IMA-2G process must evolve in order to certify a family of configurations. Model based safety assessment, as described in [14], should be able to cope with the large number of reconfigurations. For real-time performance predictability it should be possible to consider that a local module reconfiguration in a cluster has no impact on performance.

A second approach could consist in certification of the tools used and the process followed for the generation of the configurations. This second approach is independent with the number of possible configurations, provided that the generation process is mainly automatic. This second method has not been explored by the recent work. However, it could be a promising alternative for the certification issue.

First lessons and next issues

The reconfiguration objectives explored by this preliminary work have the aim of enhancing the operational reliability of the aircraft. This is a different goal from the reconfigurable avionic systems proposed in the literature [15,11,16,9,3] where reconfiguration is one means to achieving fault tolerance. Similarly, the classic FDIR (Failure Detection Isolation and Recovery) procedure used in most space systems uses dynamic reconfiguration during the recovery phase. In those cases the system is statically configured with a set of may-be-redundant (but specialized) equipment which may be powered off/on when failure occurs. In the SCARLETT project we aim at configuring generic resources, i.e. IMA modules, with uploadable software functions.

However, even if our primary objectives are different, the methodology, the software and/or hardware architecture design [15,9,11] are helping us toward our goals:

- The steps of reconfiguration process are usually the same (error diagnostic, select new configuration, apply configuration, etc.)
- The widespread idea of precomputed and identified configuration as a means to certifying configuration seems appealing,
- The need for timing constraint considerations in the reconfiguration process for real-time application. Furthermore, the reconfiguration principles presented in the ASAAC standard (see [3]) for military aircraft IMA will be of particular interest if we want to explore the distributed implementation of the reconfiguration supervisor.

Mid-term perspectives would be the implementation of an on-the-ground module and partition level reconfiguration. We also plan to detail the analysis of the proposed solution in terms of operational reliability, safety and certification. A longer-term perspective is to study other scenarios including in flight reconfiguration and reconfiguration for safety.

Second challenge: integrating high performance processors for time critical functions

Multicore architectures have reached the embedded systems market quite recently. They feature high integration and a good performance-per-watt ratio thanks to resource sharing between cores. Standard multicore include 2 to 8 cores on the chip. Cores usually have one or two levels of private instruction and data caches and share an additional level, as well as a common bus to the main memory. Such architectures often implement hardware cache coherency schemes that allow running parallel applications designed under the very convenient shared memory programming model.

Unfortunately, resource sharing makes the timing analysis of critical software very complex if not infeasible. This is due to the difficulty of taking all the possible inter-task conflicts into account, in particular when the cache coherency controller generates implicit communications. For these reasons we believe that multicore processors will be difficult to use for the design of safety critical systems.

On the other hand, many-core architectures limit resource sharing and favor explicit communications between cores. Many-core chips include numerous cores (more than 16) with distributed memories (so the conflicts are limited) and a complex communication network based on a network-on-a-chip (NoC) technology. They do not feature hardware cache coherency and the inter-core communications must be explicit, either to implement software-controlled cache coherency or the message-passing model.

We are convinced that such features are more suitable for safety critical systems in the sense that inter-core communications are

software-managed and thus more predictable. However, conflicts on the network still may occur due to implicit accesses of the main memory on cache misses and must be considered when performing timing analysis.

The challenges

Many-core platforms involve several non predictable mechanisms for managing the resource sharing which make it hard to ensure time predictability. Most of the works in the literature on many-core architectures are concerned with high performance: the idea is to extend and adapt current operating systems to the new architectural organization [6,17,13]. The main concern of an embedded system designer is somehow different: he/she wants to determine the worst-case behaviors in order to verify that the hard real-time requirements are always met, rather than to study the average performance. Three challenges arise.

First challenge: processing timing analysis

The first requirement for implementing a safety critical system on a platform is the ability to determine the worst-case execution time (WCET) for any task. The possible interactions and resource contentions due to the task concurrency must be taken into account. The widespread method based on measuring the execution time of a function on the target has been demonstrated to be generally speaking unsafe. The preferred approach is static timing analysis based on modeling techniques, which determines a safe upper bound for the WCET [4]. However, this approach is not yet suitable for many-core architectures. Improving this approach for many-core platforms and for time critical applications is the first next challenge to solve for integrating high performance processors in avionic architectures.

Second challenge: communication timing analysis

Knowledge of the timing behaviors of the accesses on the internal network is fundamental. The user must determine at what time an access is made, where and when a message crosses the network or memory components. The algorithms to compute communication time bounds are, of course, dependent of the communication technology. And this technology has dramatically changed moving from monoprocessor to many-core systems. As the number of components on a chip was low, the medium was a simple bus. In that case, only one device can drive the medium at a time, with a bus arbiter and an arbitration policy (such as round-robin, static priority or TDMA). But with the many-core generation the communication architecture has been enriched to allow a routing network, leading to the so-called networks-on-chip (NoC). In that case, there are links and switches, using either packet-oriented switching (store and forward), circuit-oriented switching (cut-through), or some intermediate policy (like the wormhole).

Most of the research on worst-case communication time have been made for external networks and bus [7]. These results are expected to be reusable for the many-core internal network.

Third challenge: memory access timing analysis

The last components that are difficult to predict are the memory controller and the memory. A RAM is a 3-dimensional storage component organized in banks, rows (or memory pages) and columns. An access (a read or a write) refers to a reference (num bank, num row, num column). The memory controller sends the command to the memory, it can ask for an activation which consists in storing a row in a buffer, a read or a write on an opened row, or a precharge which consists in saving and closing a row. All these commands require some timings. The memory response for a read also takes some time. The memory controller FIFO can sometimes reorder the references. To be able to predict worst case Memory access time for IMA many-core modules with several avionic applications becomes a real challenge.

A first solution for improving many-core predictability in avionic architectures: time oriented approaches

Any critical systems designer has to cope with these problems and has mainly two approaches to safely embed a many-core architecture. The first involves designing specific predictable processor architectures. In this way it is possible to greatly improve worst-case analyses. The cost of such specific hardware developments may often prevent their use and may force the designer to rely on a COTS. The second approach is therefore to apply an execution model: the idea is to define some rules that constrain and reduce the number of non predictable behaviors. If the rules are well chosen, the system may be analyzed without too much pessimism. The basis of this is applying time oriented mechanisms by constraining the behaviors within timing slots.

Time oriented approaches help the arbitration to shared resources. These solutions are well suited within the context of critical embedded systems since they make the formal verification easier and simplify the programming. The idea is to off-line allocate timing slots where the behaviors are constrained and thus analyzable.

Onera and Airbus have proposed a time oriented execution model on multicore to force the COTS to be deterministic [5]. The idea is to distinguish on each core the moments of functional computation and the moments of read/write from/to the memory. These two types of accomplishments occur separately in distinct pre-defined slices. Each core is assumed to have a local clock physically derived from a common hardware clock. Consequently, the concept can apply on a COTS. The separation of behaviors leads to several interesting real-time properties: the functional behavior is fully deterministic; the static WCET evaluation of an execution slice is reduced to a static WCET evaluation of a non preemptive sequential code on a monoprocessor, which is a well-known problem; and the worst case interference on the communication network between the cores and the memory is predictable.

Discussion, mid-term and long-term perspectives

These last results obtained by Onera and Airbus seem to be promising. They allow the embedding of many-core processors for time critical avionic computers to be considered. However, a new question arises: how to embed such a component into an IMA architecture. The challenge is two-fold. Firstly guaranteeing that different avionic functions implanted on the same module do not interfere, which could be done by an extension of the previous time oriented execution model. The second question is more promising and concerns the assessment of the worst-case behavior of each function. The question is: it is possible to determine this worst-case behavior for a given function without knowing the internal behavior of the other functions running on the same module. This modularity issue is central in the IMA development process. This will be the one of main issues for the next IMA-2G architecture ■

Acknowledgements

The authors are grateful to the French Ministry of Civil Aviation and the European Commission for partial funding of this work. The authors would like to thank J. Foisseau for his valuable contributions.

References

[1] Aeronautical Radio Inc – ARINC 653: Avionics Application Software Standard Interface. 1997.

[2] Aeronautical Radio Inc – ARINC 664: Aircraft Data Network, Part 1: Systems Concepts and Overview. 2002.

[3] ASAAC Consortium - ASAAC final draft of proposed guidelines for system issues - volume 4: System configuration and reconfiguration. 2004.

[4] C. BALLABRIGA, H. CASSÉ, C. ROCHANGE, P. SAINRAT - *Otawa: An open Toolbox for Adaptive wcet Analysis.* 8th International Workshop on Software Technologies for Embedded and Ubiquitous Systems (SEUS'10), LNCS vol. 6399, 2010.

[5] F. BONIOL, H. CASSÉ, E. NOULARD, C. PAGETTI - D*eterministic Execution Model on COTS Hardware*. International Conference on Architectue of Computing Systems (ARCS), München, Germany, 2012.

[6] S. BOYD-WICKIZER, H. CHEN, R. CHEN, Y. MAO, F. KAASHOEK, R. MORRIS, A. PESTEREV, L. STEIN, M. WU, Y. DAI, Y. ZHANG, Z. ZHANG - *Corey: An Operating System for Many Cores.* 8th Symposium on Operating Systems Design and Implementation, 2008.

[7] M. BOYER, N. NAVET, X. OLIVE, E. THIERRY - *The Pegase Project: Precise and Scalable Temporal Analysis for Aerospace Communication Systems with Network Calculus.* LNCS vol. 6415 pp. 122–136. Springer, 2010.

[8] R.-L. CRUZ - *A Calculus for Network Delay.* Part I: network elements in isolation. IEEE Trans. on Inf Theory 37(1), pp. 114–131, 1991.

[9] S.-M. ELLIS - *Dynamic Software Reconfiguration for Fault-Tolerant Real-Time Avionic Systems.* Microprocessors and Microsystems, Proc. of the 1996 Avionics Conference and Exhibition, vol. 21, issue 1, pp. 29–39, 1997.

[10] J.-Y. LE BOUDEC, P. THIRAN - *Network Calculus: a Theory of Deterministic Queuing Systems for the Internet.* LNCS vol. 2050. Springer, 2001.

[11] C. METRA, A. FERRARI, M. OMANA, A. PAGNI - *Hardware Reconfiguration Scheme for High Availability Systems*. IOLTS '04: Proc. of the Int. On-Line Testing Symposium, Washington, DC, USA, 2004.

[12] J. MOORE - *The Avionics Handbook*. Spitzer, C.R. (ed.) Advanced Distributed Architectures, pp. 33-1–33-12. CRC Press, Boca Raton, 2001.

[13] S. PETER, A. SCHPBACH, D. MENZI, T. ROSCOE - *Early Experience with the Barrelfish OS and the Single-Chip Cloud Computer.* Proc. of the 3rd Intel Multicore Applications Research Community Symposium (MARC), Ettlingen, Germany, 2011.

[14] L. SAGASPE, P. BIEBER - *Constraint-Based Design and Allocation of Shared Avionics Resources.* 26th AIAA-IEEE Digital Avionics Systems Conference, 2007.

[15] K. SEELING - *Reconfiguration in an Integrated Avionics Design*. Digital Avionics Systems Conference, 15th AIAA/IEEE, 1996.

[16] C. Sriprasad, M. Harvey - *Dynamic Software Reconfiguration Using System-Level Management*. 14th Digital Avionics Systems Conference (DASC), 1995.

[17] D. WENTZLAFF, C. G. III, N. BECKMANN, K. MODZELEWSKI, A. BELAY, L. YOUSEFF, J. MILLER, A. AGARWAL - *A Unified Operating System for Clouds and Manycore: Fos.* 1st Workshop on Computer Architecture and Operating System co-design (CAOS), 2010.

[18] R. ZURAWSKI - *Embedded Systems Handbook.* CRC Press, Inc., Boca Raton, FL, USA, 2004.

Acronyms

AFDX (Avionics Full Duplex Ethernet)
BAG (Bandwidth Allocation Gap)
COTS (Commercial Off-The-Shelf)
FCPC (Flight Control Primary Computer)
FCS (Flight Control System)
FCSC (Flight Control Secondary Computer)
FDIR (Failure Detection Isolation and Recovery)
IMA (Integrated Modular Avionics)
I/O (Input / Output)
MAF (Major Frame)
NoC (Network on Chip)
VL (Virtual Link)
WCET (Worst Case Execution Time)

Pierre Bieber holds a Ph. D. in computer science from the University of Toulouse. He works at Onera DTIM, where he is involved in research projects developing safety and security assessment techniques for avionics systems.

Frédéric Boniol graduated from a French Grand Ecole for Engineers in Aerospace Systems (Suapero) in 1987. He holds a PhD in computer science from the University of Toulouse (1997). Since 1989 he has been working on the modeling and verification of embedded and real-time systems. Until 2008 he was professor at ENSEEIHT. He now holds a research position at Onera. His research interests include modeling languages for real-time systems, formal methods and computer-aided verification applied to avionics systems.

Marc Boyer obtained his Engineer Degree in Computer Science at ENSEEIHT in 1996, and his PhD thesis from Toulouse III University in 2001. He is a research scientist, in the DTIM department at Onera. He works on architectures and formal methods for embedded communicating systems.

Eric Noulard graduated from a French Grand Ecole for Engineers in Computer Science (ENSEEIHT) in 1995 and obtained his PhD in computer science from Versailles University in 2000. After 7 years working in the Aerospace & Telecom field for BT C&SI, principally building high performance tests & validation systems, he joined the Onera research center in Toulouse as a Research Scientist. He works on distributed and/or embedded real-time systems and he is actively involved in the development of the CERTI Open Source project.

Claire Pagetti has a mathematical background and holds a PhD in computer science from the Ecole Centrale de Nantes (2004). She is currently a research engineer at Onera Toulouse and associate professor at ENSEEIHT. Her field of interest is mainly in safety-critical real-time systems. She is contributing to, and has participated in, several industrial and research projects in aeronautics.