ICAS-84-4.5.3

REVIEW OF AIRCRAFT-ENGINE LINKS AS A CONSEQUENCE OF ENGINE DIGITAL CONTROL

D. Rambach
Société Nationale d'Etude et de
Construction de Moteurs d'Aviation
Paris, France

Abstract

Integration consists in operating several accessories simultaneously and in providing the best possible sharing of hardware and software tasks between such accessories. Digital control of the engine through integration opens the way to new services in the field of control and display, and even in the field of performance. However, this evolution may lead to such a complex software package that its validation would be very difficult. Aircraft/engine integration is made at three levels depending on the period covered by the corresponding optimization : instantaneous optimization or control modes, intermediate optimization or propulsive modes, overall optimization or aircraft modes. Control modes cover the piloting of each engine, propulsive modes the piloting of the whole powerplant installation, inter-connection with air inlets and maintenance modes. Finally, the aircraft modes place the propulsive system under the command of the Flight Control System (FCS).

To validate the corresponding software, a classification of software as critical - essential - non essential is required. If one software module is critical the whole software could become critical.

Three approaches are possible :

to maintain the hydromechanical and the analog technologies for critical functions, to use several software sources, to limit the size of the critical software. These approaches can be applied separately or in combination. With an analytical method called pyramidal integration a hardware and software architecture can be built up based upon failure propagation laws such that the system will perform like a complex system but will switch to a "standby" configuration, more easily validated, in case of a failure.

I. Introduction

Digital computers are more and more frequently used in aircraft. The capacity of these computers to dialogue also increases and big complex systems progressively replace small independent systems. Our intention is neither to criticize this evolution nor to compare these two approaches. As a matter of fact, such a comparison would be very difficult as it would lead to compare non comparable concepts. The purpose of this paper is to bring out an evolution, to stress some difficulties and propose an appropriate approach to solve these.

I.1. Definition of Integration

We call integration the will to simultaneously perform a number of tasks, which will require using the data generated for each of the basic tasks.
This functional integration can be based upon various hardware architectures, centralized for instance, but also distributed .
Finally, for a given hardware architecture involving several digital systems, the problem arises to know how to share the software tasks and how to define exchanges. This problem will be called software integration.

I.2. Examples of Engine-Aircraft Integration

We will give some examples showing that a performance improvement can be envisaged due to the introduction of digital accessories in the engine and the aircraft.

Engine data display and monitoring : By means of a cathode-ray tube with alphanumeric display a much finer investigation of the engine condition can be carried out upon request.

Fuel consumption : With a better knowledge of the conditions external to the engine (temperature, loads, etc...), it can be envisaged not to always select the "worst case" condition, without jeopardizing safety.

Electrical throttle : In its current functions or in extended functions, the auto-throttle can be operated by the communication between engine control software and aircraft software, the throttle role needing then to be redefined.

I.3. Validation Problems

An important problem raised by digital systems is their validation. It is very difficult to prove that a software has no defect, as it is also delicate to guarantee that, if there is a failure, this failure will have no severe effect.
This demonstration seems to be far more problematic when several software modules of a different design are involved. We think that an approach currently used today is not appropriate to solve this problem. This approach consists in considering that the software dedicated to an accessory can be fully specified with this accessory, the system only consisting, in this case, of the sum of individual accessories.
In fact, this approach tends to deal with digital accessories as if they were analog accessories.

The approach we think would be more sensible would consist in saying that, in parallel with the development of programable hardware components including their own operating software, software equivalent to virtual accessories should be developed.

When several of these virtual accessories must communicate, tests specific to the software must be carried out, which include the whole software involved. This hardware/software separation seems essential to us if we must be capable of proving the system ability to perform its task.

The consequence of this distinction - hardware and virtual accessory - is that the design responsibilities of each of these accessories are not necessarily merged and the main contractor can and probably must have a different approach for each :

- the design of a physical accessory must be defined as clearly as possible by its interfaces and its environment,

- the design of a virtual accessory or software can not be isolated from the system and it must be considered within the whole system under a single leadership even if partial jobs can be entrusted to external sub-contractors, which may be the hardware suppliers. This solution is convenient from the industrial point of view but does not help the demonstration of the system quality.

II. Engine-Aircraft Integration

Most functions newly introduced by digital systems mainly aim at :

- either providing the pilot with a greater freedom,

- or automatically carrying out tasks usually performed by the pilot.

Generally, these new functions simultaneously aim at fulfilling both purposes by carrying out piloting work automatically and more finely. In case of specific problems, the pilot can get more information and then make more appropriate decisions.

In any case digital technology and integration aim at introducing closed loops and, thus, optimization, which used to be the pilot's task.

A more general classification can be applied to the engine-aircraft relations : instantaneous optimization (time constant of a fraction of second), optimization during a flight phase, optimization through a succession of flight phases.

II.1. Instantaneous optimization or control modes

An engine control system consists of loops and of a set of limitations. Choosing a control system consists in defining criteria (minimum fuel consumption, minimum response time, etc...), and in optimizing the control function by finding a compromise between these criteria.

When the control system was basically hydromechanical, this compromise was the same in all flight phases.

Analog electronics has not much questioned the principle of a single compromise but it improved its performance. However, it was also possible to introduce variations for specific flight phases (e.g. overload).

With a digital system, different optimizing criteria depending on the flight phase (e.g. cruise or take-off) can be chosen using the same control accessories, i.e. without requiring an excessive increase of the hardware. The gain will be even greater when the engine incorporates variable geometry features and when the same thrust can therefore be obtained through various combinations of the degrees of freedom : N1, N2, WF, LP STATOR α, HP STATOR α, etc... The various optimizations, specific to engine thermodynamics, are the control modes. To be logical, these permanent modes should be supplemented by temporary or specific modes, such as starting, thrust reversing, fuel chopping, etc...

By analogy, engine functional monitoring should also be mentioned (detailed display upon request and prediagnosis).

II.2. Intermediate optimization or propulsive modes

Control modes cover optimizations internal to each engine. A second level of optimization applies to all engines of the aircraft.
The types of operations the intermediate optimizations or propulsive modes can take in charge are:

- engine synchronization,

- compensation on one engine when another one fails or drifts,

- control of a measured/computed parameter such as thrust, speed, etc...

By analogy, other types of functions can be considered as propulsive modes :

- input into the aircraft of data such as thrust, thrust reserve and, more generally, information about the engine condition,

- incorporation of data such as air inlet, airbleed and power take-off etc...

Propulsive modes operate at the engine level by selecting the applicable control mode.

II.3. Overall optimization or aircraft modes

The propulsive system itself is a part of an overall system including navigation, flight controls, etc...

Then an automatically controlled engine (power lever) is conceivable according to the flight phase and a succession of flight phases.

The objective of the aircraft modes is to limit the pilot's work to the choice of a mode when everything is O.K. without any further action on the power lever.

To summarize :

- Control Modes assure the internal operation of each engine,
- Propulsive Modes assure the overall operation of the engines and associated services,
- Aircraft Modes automatically "conduct" the engines according to the mission.

II.4. Typical Architecture

As we shall see later in this paper, a certain correspondence between hardware architecture and function distribution is necessary for safety reasons.

This distribution is required to perform failure analysis but also to clearly define design (and manufacturing) responsibilities.

A typical example of such an architecture, where the engine is controlled by a redundant full authority digital system, is illustrated in Fig. 1.

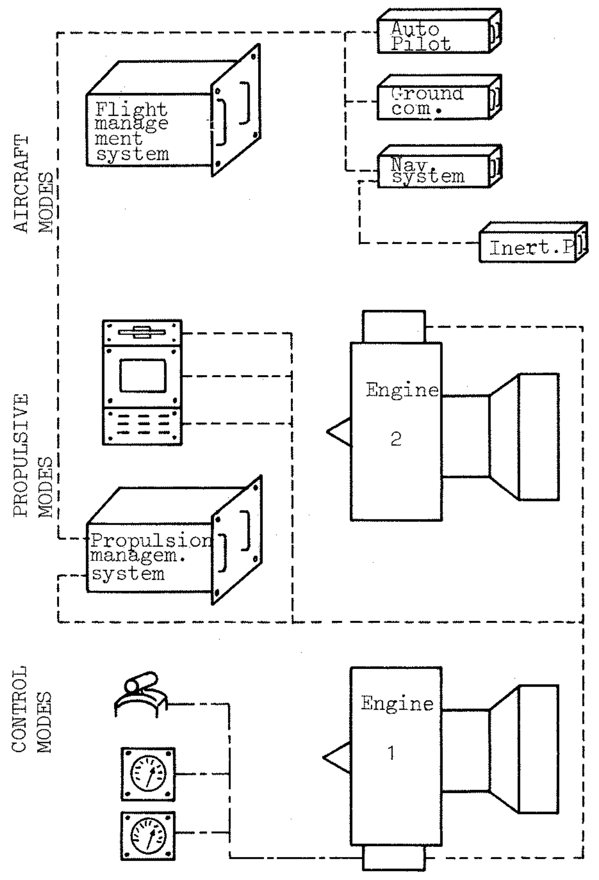

Fig. 1 Integration modes

III. Validation and architecture

III.1. Principle

A system is validated by adding an overall validation to partial validations (validations of accessories). Partial validations cannot be made without considering the organization of the system which the accessory is part of, as the characteristics to be checked cannot be defined apart from the system organization. All these characteristics are defined in a technical specification which is used as reference for validation.

The purpose of the overall validation is to check that cumulating the partial specifications is sufficient to guarantee a good operation of the whole assembly.

The problem is the same for the virtual accessory : i.e. the software. A software external specification must define its characteristics and overall tests must guarantee the satisfactory operation of the whole.

On the other hand, the type of checks to be performed during partial tests depends on failure effects and, therefore, on the system architecture. Therefore a relation (a compromise ?) must be found between the architecture and the validation level.

III.2. Validation levels

The three levels defined in the initial version of the RTCA - DO 178 are :

Level 1 : critical software "for which the occurence of any failure condition or design error would prevent the continued safe flight and landing of the aircraft".

Level 2 : essential software "for which the occurence of any failure condition or design error would reduce the capability of the aircraft or the ability of the crew to cope with adverse operating conditions".

Level 3 : non-essential software.

These definitions may vary but there is always the concept of a level 1 when safety is concerned.
We think that, in the propulsion system, safety is very often concerned, especially at take-off, and even when certification tests have demonstrated that an engine flameout was acceptable with regard to safety, functions such as the overspeed limitation remain critical.

Three options are therefore available :

a) Not to use software to carry out these functions,
b) to use 2 different software packages,
c) To use a simple software, protected enough against any other software, so that quality can be demonstrated.

We do not think that any one of these options is better than the others but the choice depends on each specific case.

Option a) can be chosen in case of a simple non digital system but if its application was to be generalized on an engine, it would mean a return to the hydromechanical control with an electronic superviser.

Option b) is probably appropriate for systems where the variable to be controlled can be obtained through several position actuation alternatives, and where safety can be maintained by freezing the defective actuator (if any). Therefore, the work sharing between the digital computers has to be based on this logic.

When this possibility is not available, the computing unit should be duplicated by differing hardware and software. This option is expensive and makes mutual testing between the two channels more difficult and hence less efficient.
Now, the efficiency of such mutual testing and, above all, the very short time available to detect and correct a failure are the most critical two aspects of a dual channel full authority digital control. In that case, option b) brings therefore few advantages.

Option c) is based on a software reduced in size, possibly combined with partitioning. Compared with most systems, the control system is a "small size software" and it is possible to single out an even smaller "core" dedicated to critical functions.

III.3. Partitioning and critical software

The simplest partitioning would consist in using several processors. Today this approach does not appear as unrealistic. However, the development of the partitioning concept in electronics/computer applications would also be interesting so that the hardware reliability can be computer in a conventional way and the software failure probability be considered as negligible. As a matter of fact, the major doubt remaining today in software test is related to interference between modules. If independence between modules is still difficult to achieve, is must be possible at least to single out one "critical" module completed at the beginning of each cycle, independently of what comes after.

IV Software Integration

IV.1. Pyramidal integration

The combination of several modules makes it very difficult to validate a software. The combination of several software modules can lead to a system which cannot be validated. It is therefore necessary to proceed with precautions when building the system. In a previous paper AGARD n° 349, we presented an hardware architecture philosophy which we called pyramidal integration. This architecture defines levels, the lower levels corresponding to elementary functions whereas the upper levels are obtained by grouping functions of lower levels and new functions. The pyramid building shall satisfy the following law : a failure affecting a function of a given level can generate failures in related functions of an upper level only.
The pyramidal architecture is schematically represented in Fig. 2.

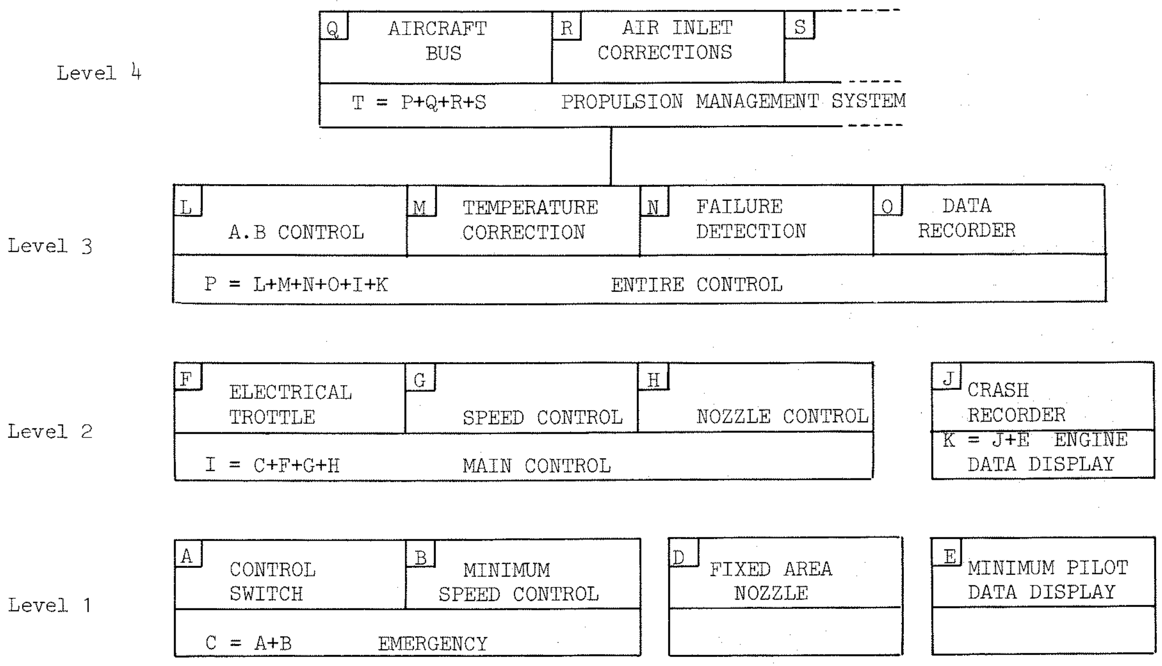

Fig. 2 Pyramidal integration

IV.2. Software pyramidal integration

Formal methods used to describe the requirements can be applied either through function blocks exchanging data, or through data related by functions (see Fig. 3).

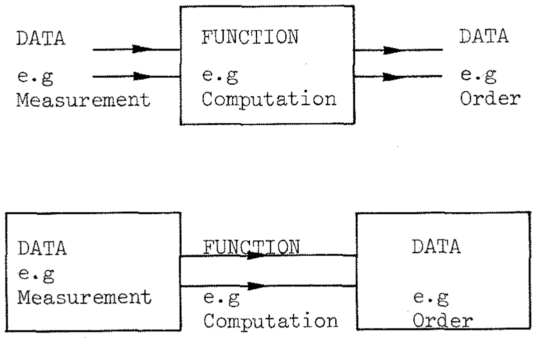

Fig. 3 Dual representation

In the pyramidal integration, the connections between functions imply a flow of data (up and down). In the dual description of the pyramidal integration each block represents a set of data and each link a processing. The pyramid law becomes : any erroneous data can only produce errors on upper level data generated from the erroneous one.

Pyramidal architecture through functions is quite appropriate for the analysis of a complex system and leads to a clear hardware architecture, which is compatible with safety.

Being rigorous and easy to computerize, pyramidal architecture through data is well adapted to automatic analysis. It is also probably more independent from the hardware than the pyramidal architecture through functions. The pyramidal architecture must help choose the task assignment to the various processors while meeting the safety rule of the pyramidal architecture. Besides, this rule may be considered as a constraint which could be taken into account in a multiprocessor scheduler.

IV.3. Downgraded operation

Due to the shape of the pyramid, any failure obligatorily affects the top of the pyramid. That is normal since, in this case, the system integrity is no longer hundred percent maintained. However, if this should lead to the loss of the upper level availability, this level might not be often operational in complex systems and complexity might become an "absolute" limit to integration.

To reduce the unavailability rate, it should be desirable that, when a failure occurs, and as far as it can be detected, a downgraded operation would substitute for the initial operation in order to :

- on the one hand, and when possible, perform an intermediate job between the lost function and the functions of the lower level,

- on the other hand, make it possible for the upper levels to carry out the essential part of their mission.

The concept of downgraded operation, which is very expensive without a digital system, can be introduced through the software : measurement replaced by a tabulated evaluation, simplified computations, etc...

V. Conclusion

A great number of the principles, which have been presented here, extends beyond the field of aircraft engine control. However, they result from specific concerns of the engine manufacturer :

- absolute priority to safety,

- attempts to reduce weight and overall dimensions,

- the need to keep the engine as a clearly identified entity,

- importance of the maintenance problems.

At first sight, meeting these requirements does not favor the integration. However rejecting integration would mean to deprive oneself (and the pilot) of a series of options, which will be soon considered as mandatory.

Digital control will therefore be a part of a complex software, but which must be progressively built in order to preserve the possibility of progressive switching to downgraded modes in case of failure to achieve more and more independent and "manual" operation if necessary.