# REALIZATION, VALIDATION AND OPERATION OF A FAULT-TOLERANT MULTIPROCESSOR : ARMURE

Yves DESWARTE*, Khadija ALAMI**, Olivier TEDALDI***

* INRIA/LAAS ** LAAS/CNRS
7, avenue du colonel Roche
31077 TOULOUSE CEDEX
FRANCE

*** CEIS-Espace
Rue des Frères Boudes
31084 TOULOUSE CEDEX
FRANCE

## ABSTRACT

ARMURE is the architecture of the data processing system choosen by CNES (the French Space Study Center) for the ground stations of the SARSAT/COSPAS search and rescue satellite program. The dependability requirements of the data processing system are much more strict for "real-time" functions (antenna control, telemetry acquisition) than for "deferred-time" functions (distress beacons, location computation), mainly with respect to error recovery time. For this reason, ARMURE presents a higher redundancy for the real-time units than for the deferred-time units.

The paper describes the development and verification/validation methods and tools which have been used for the realization of this system, and the operational dependability measures are presented and discussed. In particular, a detailed unavailability analysis shows that operation and maintenance staff training is essential in order to reach the availability aims which were fixed.

## INTRODUCTION

Dependability has always been a main concern for real-time computing system designers in as much as the cost of the consequence of a failure (loss of output, destruction of machinery, human casualties) is disproportionate to the cost of the faulty component (an electronic component, a program instruction) that caused the system failure. In most fault tolerant systems, the hardware redundancy of the computer architecture is determined by the most critical functions or phases. Thus, one may find duplex systems (Tandem "Non-stop computers"), triplex systems (August "Can't fail computers"), quadruplex systems (Stratus FT200 series). However, all the functions of one computer system are not equally critical: for instance, on commercial aircraft, it is a fact that coffee machine control is less critical than flight control! It is obvious that applying the same redundancy to all functions induces heavy overhead. Consequently, some systems are able to adapt their redundancy to the requirements of each function [Wensley 78, Deswarte 86]. The ARMURE architecture of the computing system for ground stations of the SARSAT/COSPAS program has been devised along these lines.

## I. DEPENDABILITY REQUIREMENTS OF THE SARSAT/COSPAS GROUND STATIONS

The purpose of the SARSAT/COSPAS program [CNES 84] is to aid search and rescue operations by using satellites to locate distress beacons. This program is the outcome of international cooperation between:

- the National Astronautics and Space Administration (NASA) in the USA,
- the Department of Communications of Canada,
- the National Space Studies Center (CNES) in France,
- the Ministry of Merchant Navy (Morflot) in the USSR.

CNES selected the CEIS-Espace french company for the design and development of the SARSAT/COSPAS ground stations. These stations are called Local Unit Terminals; their function is: to acquire the data transmitted by the beacons through the satellite, to locate the beacons and to report to the search and rescue services involved.

The dependability requirements are drastic since the survival of people in an emergency is dependent on the correct functional operation of the station. CNES fixed a strict target of 98.5% availability for a station working a full 24 hours a day. The scope of each station being limited to a few thousand miles, stations are likely to be installed anywhere on the earth. Mean time to repair (MTTR) must consequently be established at about a hundred hours so as to keep the cost of maintenance within reasonable limits. This induces a requirement in the mean time to failure (MTTF) of about 10 000 hours for the computing system, i.e. less than one failure in 14 months' working 24 hours out of 24.

From the operational point of view, the availability is given by the ratio between the number of satellite passes that are correctly tracked and processed, and the total number of satellite passes that could be observed. The consequences of failures on the measured availability thus depend on the functions involved. So, real-time functions (antenna control, acquisition of data sent by the satellite) must not be interrupted for more than about 20 seconds during the satellite pass, or else the pass must be considered as useless; this could be an "unavailability" of greater importance than the mere failure duration. In the same way, the storage of telemeasures and location results can be seriously

perturbed by a failure, even a short one, that would affect the stored data. On the contrary, the deferred time functions (computation of locations, computation of antenna control tables, printing of location results) can generally be interrupted for several minutes or more without impairing the overall operation of the station. In fact, the time necessary to calculate locations (about half an hour) is short compared with the time necessary to carry out a rescue operation (from a few hours to a few days).

Practically, we can deduce the following operational requirements:

- **for all functions:**
  - = transient faults must be tolerated without inducing any noticeable degradation in performance,
  - = at least one permanent fault - any fault - must be tolerated; if this causes a degradation in performance, this degradation must not reduce availability through losses of satellite passes, or inability to compute acquired passes owing to the lack of sufficient computing throughput,
- **for real time functions and storage of telemeasures:**
  - = these functions must be fulfilled even in case of a permanent fault; if a manual reconfiguration is required, the time necessary to reconfigure must remain within limits so as not to miss a pass as stated above (about 20 s.),
- **for deferred time and editing functions:**
  - = the failure of one of the units in charge of these functions must not prevent the station from working: it must be possible to process these functions on various units, the peripheral units must be mutually redundant: for instance, operator commands can be entered from the VDU or the teletype, result printouts can be done on the printer or on the teletype.

In the above, we have considered only the hardware induced faults, because:

1) the application is stable (the same software is used for long periods in the same conditions); so, if a sufficient validation is carried out, software faults should appear only in the early life of the system;
2) the repair time, in case of software-induced faults, is usually short (warm start up), compared to the time necessary for hardware repair (about a hundred hours); such faults are of little consequence as regards availability.

## II. DESCRIPTION OF THE ARCHITECTURE

### II.1.HARDWARE STRUCTURE

Such requirements as described above have led us to design a computer architecture that makes use of different redundancies according to the functions concerned. The different requirements concerning the time to detect and correct errors make us consider two classes of functions: for real time functions and storage devices, a duplex structure has been selected; for deferred time functions and input/outputs, a redundant structure with graceful degradation has been chosen [Deswarte 81]. This is characteristic of the ARMURE architecture (ARchitecture MUlti-processeur REdondante, in French) whose hardware structure is shown in figure 1.

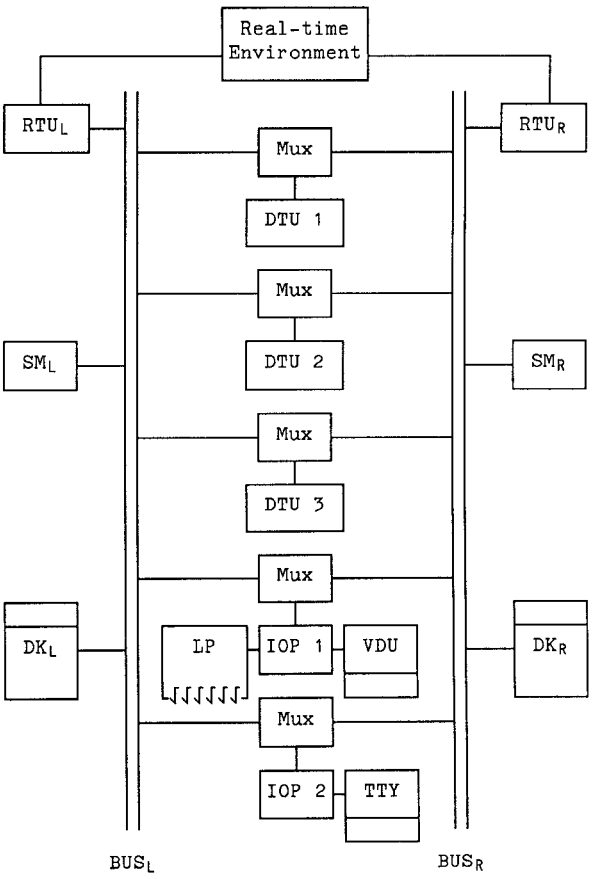


Figure 1
ARMURE hardware structure

The processing units (or processors) are of three different types:

- **Real-Time Units** (RTUs): their function is to drive the antenna control devices, to acquire the telemeasures transmitted by the receiver, to preprocess the telemeasures and record the preprocessing results on disks. To do so, the units have been doubled (RTU$_L$ and RTU$_R$) and each has access to a separate disk (DK$_L$ and DK$_R$) through its own separate bus (BUS$_L$ and BUS$_R$). The RTUs are hardware isolated so that a faulty unit does not perturb the other unit. Both RTUs get information from the same receiver and are macro-synchronized by a common 1 Hertz real-time clock. When free from faults, they both carry out the same processing of the same data, so they must output the same results. Orders sent from the two units are compared by the antenna control devices. In the event of a discrepancy, an alarm is signalled to the operator who can connect either RTU with the antenna devices according to the better reception level: if the reception level fades, it means he has selected the "wrong" unit, and he must connect the other one.
- The **Deferred-Time Units** (DTUs) process the data stored by the RTUs (computation of the location of distress beacons) and carry out the routines required by the working of the ground station (forecasting of satellite passes, computation of the antenna control tables for the next transists, correcting of satellite orbit references). For

that purpose, the DTUs use the data stored in two copies on left and right disks ($DK_L$ and $DK_R$), by connecting their own bus multiplexor (MUX), respectively on the left bus ($BUS_L$) then on the right bus ($BUS_R$).
- The **Input-Output processors** (IOPs) drive the peripheral devices. Like the DTUs, they have access to data duplicated on both disks ($DK_L$ and $DK_R$), by connecting their MUX to the left and right buses.

Each of these units is made up of a standard single board computer (Intel SBC 86/12A), including a 16-bit microprocessor, a local RAM and ROM memory and auxiliary circuits. Each micro-computer is connected to a standard bus (IEEE 796), but its own memory is usually large enough to execute most processing without interaction with the other units. Consequently, most of the time, the processors can be isolated by hardware (MUX) or by software. Such an isolation helps to reduce the risk of contamination of sane units by faulty units. Each unit has its own power supply, so that the failure of one supply causes the failure of only one unit. The DTUs and IOPs are not physically addressable, which allows the number of units to be increased or decreased in a software-transparent fashion, and to improve the virtual isolation of the units. The global data of the operating system or of the application is stored in two copies in the shared memories ($SM_L$ and $SM_R$) or on the disks $DK_L$ and $DK_R$, located on the RTU busses. It is through the shared memories and through the disks that the units communicate. Owing to their duplication, a single failure cannot prevent communication.

## II.2.OPERATING SYSTEM

### II.2.1.Characteristics of the operating system

To operate the particularities of the hardware structure, we had to develop a specific operating system [Deswarte 84]. This operating system is the software in charge of the management of the multiprocessor resources:
- processor management (task scheduling)
- data management (shared memory managment, file management, access rights management),
- input-output management.

The design of the operating system is influenced by three constraints due to the selected hardware structure:
- the processors are not addressable: it is not possible to assign a specific task or to send a message, or to report an event to a physical processor,
- there is at most one active task per processor ("mono-tasking"); tasks are not interruptible and they cannot wait for an event or a message,
- interactions between tasks are limited to the task creation and to the sharing of data in shared memory or in files, under supervision of the operating system, if necessary protected by "locks".

These constraints arise from a will to isolate the processors, which allows a reduction in the risks of error propagation, and from a will to decentralize and build up a modular structure in order to make it suitable dynamic reconfiguration.

### II.2.2.Error detection

Several error-detection facilities are included in the operating system:
- **Self-test program:**
  This is a subroutine called either when a processor is idle, or before executing an action likely to propagate a possible error (writing on file, creating a task). This subroutine [Abadir 82] is organized such that the processor can exit from the subroutine only if no error has been detected.
- **Comparison between left and right copies:**
  This comparison is done at every reading of the shared memory or of the files by a DTU or an IOP.
- **Check on resource-holding duration:**
  "Time-outs" are associated to locks, to task descriptors, and to files, in order to avoid blocking the system if a unit loops in the self-test, or if owing to a transient or permanent failure it does not release the resources it has obtained. These time-outs are checked only by the units that request the resource. There is no periodic checking of time-outs. Thus, for instance, a lock can remain blocked for ever if no unit tries to obtain the lock. This is precisely the aim of "time-outs": to prevent the blocking of a fault-free unit by a failing unit.
- **Watch-dog:**
  The watch-dog plays a similar rôle to that of the task time-out, but its effect is internal to the DTU executing the application task. It consists of a timer linked to an interrupt. The timer is initiated at the beginning of the task with a value equal to the max time forecast for the task. If the task ends before time is out, the timer is de-activated. On the contrary, if the interrupt happens before the end of the task, the latter is aborted. So, the watch-dog prevents disconnection of a sane unit which would loop due to a fault in the application program or in the data.
- **Likelihood tests in the operating system:**
  The operating system checks the consistency of the system data it handles. This data is duplicated in shared memories. In the case of inconsistency in one copy, the processing for this side only is abandoned, processing is normally continued with the other copy of data (i.e. on the sound side only). In case of an inconsistency in both copies, the unit which detects the inconsistency stops (it suspects itself of failure). Thus, step-by-step, all the units which try to use this inconsistent system data stop. To make the system sane again, it is necessary to reinitiate it (cold start up), and if necessary to roll back the aborted processing by an operator command.
- **Likelihood tests by the application:**
  Besides the error-handling specific to the application (elimination of erratic points, halting the execution of algorithms which are diverging or those whose data is insufficient), the application programs are able to signal to the OS any inconsistencies that they detect in the data they are processing. To do so, they call the automatic roll-back primitive which will initiate the same process as for the other means of detection.
- **Hardware error-detection devices:**
  The hardware includes a certain number of error-detection devices such as detection of inexistent addresses, parity errors or error-detecting code violations, transfer errors, invalid codes, blocked bus, etc.

II.2.3.Error handling

We have kept several aims in mind when designing the automatic error handling:
- to make diagnostics easier for the operator: to do so, messages that are as informative as possible must signal any error as soon as it is detected,
- to make application programming easier: error handling must be transparent for the application programmers,
- continue the processing as long as possible, even though there are errors: so, in case of discrepancy between the left and right copies of data, if there is no means of finding out which copy is wrong, processing must be pursued separately for each copy: it is better to obtain two locations of a distress beacon than no location at all.

According to the kind of detection and the state of the unit that detects the error, three sorts of processing are applied:
1) **Warning**: the error is signalled to the operator by editing a system message including all the known information concerning the symptoms of the error. After editing this message, processing is continued: it is a mere warning. In fact, in this case, the unit which detects the error deems itself sane and able to carry on the execution. This sort of error handling is to be applied for instance when lock, file or task time-outs are detected, or again when certain OS likelihood checks detect inconsistency.
2) **Task aborting**: the error is signalled (as above) and the current task on the unit which detects the error is aborted: this is the case when the error is recognized as fatal for the task, but not for the unit itself. Aborting a task will generally cause the automatic roll-back of this task by another unit. This sort of error handling can be induced by:
   - the detection of discrepancy between left and right copies,
   - the application likelihood checking (calling the roll-back primitive),
   - the watch-dog,
   - some OS likelihood checks.
3) **Unit blocking**: this is the case when the error is deemed fatal for the unit or for the system. This causes an instantaneous halt of all the activities of the unit and the blocking of the processor and its communications in order to avoid spreading the error. Such halted processing will be automatically rolled-back by other units. The blocking of the unit can be induced by the self-test or by some OS likelihood checks.

When a task is aborted or when the unit which executes the task is blocked (the task time-out is exceeded) an automatic roll-back is attempted on another unit (except in the case of RTUs: the task is processed concurrently on the other copy). However, if the roll-back is aborted too, there is no third automatic execution attempt; only the operator can ask for a new execution of the complete group of tasks: if the automatic roll-back fails, it is probably due to the task having been contaminated by other erroneous tasks; in this case, the only consistent checkpoint is the beginning of the group of tasks. Automatic roll-backs are activated in case of the blocking of a unit or of a bus (automatic reconfiguration). When an RTU or an RTU bus is concerned, the processing continues on the other unit, but only one copy of the global data and of the files is accessible: there is no longer any comparison between left and right copies. When a DTU is concerned, another DTU detects the task time-out and rolls-back the interrupted task. When an IOP is concerned, the blocking is detected by the other IOP which takes charge of printouts and operator dialogue on its own peripheral devices [Qadiri-Alami 83]. In all these cases, the OS adaptation is automatic and transparent for the application programs.

Thanks to the error reporting messages, the operator is kept informed about the current state of the computer system. He/she can manually reconfigure by blocking the busses and switching off the units, or by switching on the units and unblocking the busses. He/she can also modify the assignments of the peripheral devices to the various editings and to the operator dialogue (by means of typed commands). Manual reconfiguration induces the same handling as automatic reconfiguration with the exception of the roll-back of an RTU which requires the updating of files and global data; this is done by operator command and by complete reinitialization (cold start up).

The constraints which the architecture forces upon application programs are as reduced as possible. The OS hides all the error handling and the redundant hardware structure from the application. However, for the automatic roll-back to be efficient, no task must destroy its own entry data: this constraint is obeyed by the application programs developed by CNES. These programs, written in Fortran, were developed and tested by batch processing on the main CNES computers, independently from the target machine. Owing to the OS transparency, this large sized software (31 programs, 200 modules, about 600 Kbytes) was successfully integrated to the architecture and the OS in a relatively short time (3 months).

## III. VALIDATION OF THE ARCHITECTURE

Considering the high complexity of the architecture and specially of the OS, validation is a particularly important phase. This validation was achieved in four steps:
1) **Verification of the specifications**: this was done at the very beginning of the design, and was continued during most of the realization. It consisted of:
   a) computation of the expected system availability from the known failure rate of the hardware (components, boards, power supply), with the help of the SURF software package, developed at LAAS [Costes 81]; these computations confirmed our choice of architecture [Kanoun 82],
   b) modelling the most critical parts of the OS through Petri nets, and analyzing these nets by means of automatic tools developed at LAAS (OGIVE/OVIDE package) [Montel 83], in order to verify certain characteristics such as conflicts, absence of dead-lock and saturation, even though there are errors.
2) **Modular testing**: each of the OS modules has been tested interactively with all the combinations of its environment liable to modify its behaviour. This testing has been done following the layered

structure of the OS, from the innermost layers to the interfaces with the application or with the operator. To make this testing easier, specific interactive techniques were developed to read and modify the data or the parameters, to start the modules and to interrupt processing.

3) **Integration testing**: since the architecture is highly concurrent, and the units work asynchronously, it was not possible to check all the computing modes by exhaustive testing. Only two types of testing have been applied: "representative" testing, when a typical application ran on different configurations, nominal or degraded; "aggressive" testing when the system was put in "difficult" circumstances: conflicts, overload, reconfigurations, etc. Both these types of testing use the same interactive tools as for the modular testing at the level of each unit.
4) **Error injection**: it would have been advisable to inject errors as representative as possible in our system. This would have allowed us to verify that the system reacted as expected. Unfortunately, being short of time and means, we could only achieve this error injection on a small scale [Abadir 82]. It must however be pointed out that quite a number of hardware faults have appeared since the system has been in operation (see paragraph IV), thus allowing to verify its behaviour when dealing with actual faults.

It should be underlined that these validation methods have proved to be efficient since, once the modular testing was done, hardware, OS and application software integration lasted only three months. The first reception testing by CNES with a simulated real-time environment was in fact begun before the hardware and software were completely debugged: residual faults were sufficiently tolerated by the system, thanks to the automatic error handling. Test of the OS can be considered as satisfactory since it has not been necessary to modify the software since the integration was completed.

## IV. OPERATION RESULTS

Installation of the system at the CNES facility at Toulouse was achieved in June 1984, with actual operation testing. In July and August, 24 hours a day system operation has allowed tuning of some parameters (minimal time between successive satellites passes, archiving period, minimal number of measures for orbit correction, task time-outs, etc.), and correction of some application software faults. During that time, the operators have been trained to react according to preestablished procedures for forecasted errors, so as to rapidly put the system back to its normal operation.

Since September 1984, the system has been operational 24 hours a day, concurrently with a Canadian system sharing the same antenna. The period from September 1, 1984 to April 24, 1986 was marked by a number of hardware accidents such as:
- transient faults on a memory board,
- transient faults on a micro-computer board,
- transient and permanent faults on disks,
- permanent faults in power supplies.

A certain number of operator mistakes leading to a total shutdown of the station must be added to these hardware accidents. Over that period of operation, the records can be summarized as follows:

| Total number of passes | Suppressed passes | Non tracked passes | Badly processed passes | Correctly processed passes |
|---|---|---|---|---|
| 12292 | 2875 | 1326 | 88 | 8203 |

Suppressed passes correspond either to overlapping passes, or to deliberate will of the operators (for instance for antenna maintenance). So, the effective number of passes to be taken into account is 12292-2875 that is 9617 passes. Out of these 9617 passes, 1326 could not be followed, since the system was not available at the time of the satellite pass. The 88 badly processed passes correspond to passes which have been tracked by real time units, but for which deferred time processing did not work properly. On the whole, the global unavailability of the system is 1414/9617 passes, that is about 14.7%, which gives a 85.3% availability, definitely below our aims. We still have to scrutinize the causes of this unavailability, and for each type of cause, the consequences concerning the lost passes. This is shown on the following table:

| Double hardware faults | Badly tuned parameters | Trials out of specs | Hardware maintenance | Operartor mistakes |
|---|---|---|---|---|
| ≈ 2 | 53 | 131 | 383 | 844 |
| ≈ 0.02% | 0.55% | 1.4% | 4.0% | 8.8% |
| remark 1 | remark 2 | remark 3 | remark 4 | remark 5 |

**Remark 1**: These passes could not be tracked because of double hardware faults: a first fault led to the isolation of a faulty part of the hardware and continued processing in a degraded configuration; a second fault perturbed the working of the degraded system, which did not have enough redundancy to tolerate this second fault. This kind of double fault is all the more likely to happen as the duration of the operation in a degraded configuration is long (long time to repair), and as the faults are frequent (high failure rate). However, the conditions concerning these two points are particularly unfavorable: it often happens that the repair time is long (about 10 days or so), by lack of fast action of the operator, or by lack of availability of maintenance staff, or by lack of spare parts in good repair. Also, the failure rate is apparently 5 times higher than that which had been calculated during preliminary evaluations. This is partly explained by the fact that the hardware of the operational system is that which was used for four years for development, testing and hardware and software integration. Yet, in spite such unfavorable conditions, the number of passes lost because of these double faults is so small that it cannot be accurately discerned in the records: it can be estimated at about two passes during the considered period.

**Remark 2**: 53 passes have been lost owing to defective parameter tuning: too short time-out assigned to some tasks, or too long time-outs assigned to some locks. These can be considered as infant faults: they only happened during the first few months of operation, before the parameters were correctly tuned. This kind of faults should not happen again, unless deep modifications in the application software be carried out.

**Remark 3**: 131 passes have been lost owing to operation out of specification: either due to faulty satellites, or to an excessive number of nominal or

out-of-specification beacons (to measure the system limits), or again to test application software beyond the limits of nominal operation. This kind of unavailability can not be ascribed to our system.

**Remark 4**: Hardware maintenance appears to be an important cause of loss of passes. However, none of these passes would have been lost if normal maintenance procedures had been followed: it is possible to repair any unit while keeping the system in a degraded operational state, then putting it back into the nominal configuration between two passes. To save time, these procedures are not generally followed, owing to the presence of the Canadian system. Under these conditions, following the correct maintenance procedures would indeed have prevented the loss of passes, but would have taken more time. This kind of unavailability must not be taken into account in the calculation of the actual availability of the system since it would have been avoided if normal procedures had been followed.

**Remark 5**: The mistakes of the operators are the main cause of loss of passes. These mistakes were due either to a lack of attention or to a lack of efficient training. Yet, to avoid such mistakes, the operator dialogue is limited to selecting among menus, and numerous likelihood checks have been inserted into the routines of the management of this dialogue in order to reduce the risks of entering wrong values. It appears that most of these mistakes are connected with incorrect handling of disks and incorrect reinitializations. Even if these mistakes are infrequent, they can have serious consequences because of the high number of passes involved in only one mistake: for instance, during the last Christmas holidays, one of those mistakes provoked the loss of 520 passes for 20 days. Nevertheless, we can consider that such mistakes must not be directly ascribed to the design of the architecture.

The above remarks being considered, the unavailabilities actually ascribable to our system are 0.55% for the infant faults (tuning parameters), and about 0.02% for multiple hardware faults. The availability over the period of reference is 99.43%, higher than the ascribed target, in spite of an abnormally high hardware failure rate and a long mean time to repair. If infant faults are left out, the asymptotic availability reaches 99.98%.

## CONCLUSION

The operational period which started in September 1984 can be considered as unfavorable because of quite a number of infant faults and because of a measured hardware failure rate much higher than that which had been calculated during preliminary evaluation (an average of one failure of one component every 200 hours during the best periods, instead of about one failure per 1000 hours as estimated). This has enabled us to check the behaviour of the system confronted with actual faults and to justify our policy of fault tolerance. As regards the mean time to repair, taking into account the random characteristics of the failures, this time depends for a large part on the reaction time of the operators. At the beginning of the operation, this time was relatively short, owing to the attention given to this new system. Today, since experience has shown that the system continued to work even in case of faults, the tendancy of the operators is rather to wait for the system to be completely down before they act. The preliminary estimation of a mean time to repair which was about 100 hours was not excessively pessimistic.

Consequently, we may consider that the selected architecture is efficient in its fault tolerance, and that the availability targets would be easily reached if the failure rate was what it should be and if the designed procedures were applied both by the maintenance staff and the operators.

## Acknoledgements

The authors gratefully thank Mr. and Mrs. DEHOUX and Dr. David POWELL for their help in producing a readable English paper. The ARMURE study was partly supported by the SURF project of ADI.

## REFERENCES

**[Abadir 82]** J.ABADIR, Y.DESWARTE: "Run-time program for self-testing a 16-bit microprocessor", 1982 International IEEE Test Conference "Cherry-Hill", Philadelphia, November 1982, pp.205-213

**[CNES 84]** CNES: "SARSAT-COSPAS: global satellite system for distress positioning", Presentation Brochure, CNES, April 1984

**[Costes 81]** A.COSTES, J.E.DOUCET, C.LANDRAULT, J.C.LAPRIE: "SURF: A Program for Dependability Evaluation of Complex Fault-Tolerant Systems", 11th International Fault-Tolerant Computing Symposium (FTCS-11), Portland, June 1981, pp.72-78

**[Deswarte 81]** Y.DESWARTE, J.L.BOSSEBOEUF, P.COHEN, N.GARGIR, J.LEROUGE: "A fault-tolerant multimicroprocessor architecture for SARGOS", 11th International Fault-Tolerant Computing Symposium (FTCS-11), Portland, June 1981, p.251

**[Deswarte 84]** Y.DESWARTE, N.GARGIR: "Operating system for fault-tolerant multi-processors", 4th International Conference on Reliability and Maintenability, Perros-Guirec, France, May 1984, pp.320-326, in French

**[Deswarte 86]** Y.DESWARTE, J.C.FABRE, J.C.LAPRIE, D.POWELL: "A saturation network to tolerate faults and intrusions", 5th IEEE Symposium on Reliability in Distributed Software and Database Systems, Los Angeles, January 1986, pp.74-81

**[Kanoun 82]** K.KANOUN: "Previsional dependability evaluation of the SARGOS data processing system", internal CEIS-Espace report, December 1982, in French

**[Montel 83]** B.MONTEL, D.GRISSAULT, E.LE MER, C.ROBERT, A.SIVET, P.AZEMA, S.BACHMANN, B.BERTHOMIEU, M.DIAZ, B.PRADIN: "OVIDE, A Software Package for the Validation of Systems Represented by Petri Nets Based Models", 4th European Workshop on Applications and Theory of Petri Nets, Toulouse, France, September 1983, pp.292-308

**[Qadiri-Alami 83]** K.QADIRI-ALAMI: "SARGOS: design and realization of a fault-tolerant and reconfigurable input-output system", 3rd cycle thesis, March 1983, Université Paul Sabatier, Toulouse, in French

**[Wensley 78]** J.H.WENSLEY, L.LAMPORT, J.GOLDBERG, M.W.GREEN, K.N.LEVITT, P.M.MELLIAR-SMITH, R.E.SHOSTACK, C.B.WEINSTOCK: "SIFT: the design and analysis of a fault-tolerant computer for aircraft control", Proceedings of the IEEE, vol.66, N°10, October 1978, pp.1255-1268